Approche sémantique basée sur les intentions pour la modélisation, la négociation et la surveillance des contrats de qualité de service

Kaouthar Fakhfakh

THÈSE

En vue de l'obtention du

Doctorat de l'université de Toulouse

Délivré par *l'Université Toulouse I et l'Ecole Nationale d'Ingénieurs de Sfax*

Discipline : *Informatique*

Présentée et soutenue par *Mme. Kaouthar FAKHFAKH*

Le : *21 Mai 2011*

Titre : *Approche sémantique basée sur les intentions pour la modélisation, la négociation et la surveillance des contrats de qualité de service*

Jury

Président	
Mme. BELGUITH Lamia	Maître de conférences (HDR) à FSEG-Sfax, Tunisie
Rapporteurs	
M. BENSLIMANE Djamal	Professeur des universités à Lyon, France
M. GARGOURI Faïez	Professeur des universités à l'ISIM-Sfax, Tunisie
Examinateurs	
M. DRIRA Khalil	Directeur de Recherche au LAAS-CNRS
M. CHAARI Tarak	Maître assistant à l'ISEC-Sfax, Tunisie
Directeurs de thèse	
M. TAZI Saïd	Maître de conférences (HDR) à l'UT1, France
M. JMAIEL Mohamed	Professeur des universités à l'ENI-Sfax, Tunisie

---

Unités de recherche : *LAAS-CNRS et ReDCAD*

Approche sémantique basée sur les intentions pour la modélisation, la négociation et la surveillance des contrats de qualité de service

---

Intention-based Semantic approach for service level agreements modeling, negotiation and monitoring

Kaouthar FAKHFAKH

Université de Toulouse 1 & Université de Sfax

À MON MARI POUR SA PATIENCE,

SON SOUTIEN ET SES ENCOURAGEMENTS PENDANT CES ANNEES DE THESE.

POUR TOUTES LES CONCESSIONS QU'IL A DÛ FAIRE ET TOUTE L'AIDE QU'IL A SU

M'APPORTER, JE LUI DÉDIE CE TRAVAIL QUI N'AURAIT PU ABOUTIR SANS SON SOUTIEN.

A MA FAMILLE ET A TOUS CEUX QUE J'AIME.

REMERCIEMENTS

Je souhaiterais tout d'abord remercier mes directeurs de thèse M. Said TAZI et M. Mohamed JMAIEL pour l'encadrement de mon travail et pour leurs conseils avisés et leurs remarques pertinentes afin d'améliorer la qualité de ma thèse et de bien cerner toute la problématique.

Je remercie tout particulièrement M. Djamal BENSLIMANE et M. Faïez GARGOURI d'avoir acceptés d'être rapporteurs de mon manuscrit. Je remercie également Mme Lamia BELGIUTH d'avoir bien voulu présider le jury de ma soutenance de thèse ainsi que M. Khalil DRIRA et M. Tarak CHAARI d'avoir accepté d'être examinateurs de ce travail.

Je remercie tous les chercheurs, enseignants et membres du personnel du laboratoire ReDCAD à Sfax et du laboratoire LAAS à Toulouse pour leur amitié et leur aide pendant ces années de thèse.

Je tiens à remercier mes collègues et amis pour les bons moments que nous avons vécus ensemble.

Titre

Approche sémantique basée sur les intentions pour la modélisation, la négociation et la surveillance des contrats de qualité de service.

Mot clés

Contrats de qualité de service; Alignement d'objectifs sémantiques; Ontologies; Intentions; Architectures Orientées Services; Surveillance de qualité de service.

Résumé

Nos travaux de thèse s'inscrivent dans le cycle de vie des contrats de qualité de service (QdS) pour les architectures orientées services. Le cycle de vie de ces contrats se caractérise principalement par une phase de spécification et de modélisation des niveaux requis de qualité, une phase de négociation de ces niveaux et une phase de surveillance du contrat de qualité de service élaboré entre le client et le fournisseur. Concernant la première phase, nous avons remarqué que les approches existantes de modélisation des contrats de qualité de service se limitent à une description incompléte notamment dans le cas où les niveaux de service requis sont composés à partir d'autres métriques élémentaires de qualité. Pour remédier à cette insuffisance, nous avons défini un modèle nommé « SLAOnt » caractérisé par sa richesse sémantique et par sa complétude. Ce modèle de contrats de QdS est le fondement de base pour la phase de négociation et la phase de surveillance. Quand à la deuxième phase, nous avons constaté que les approches existantes de négociation consistent à choisir un sous-ensemble de clauses et de valeurs parmi des choix prédéfinis par le fournisseur. Cependant, le client peut ne pas comprendre ces choix dans le cas où il n'est pas expert du domaine. En conséquence, il n'a pas la possibilité d'exprimer ses exigences avec ses connaissances et son langage. Pour cela, nous avons proposé une approche de négociation basée sur un alignement d'objectifs sémantiques de qualité de service pour l'élaboration de ces contrats. Cette approche d'alignement se base sur des inférences sémantiques dans toutes ses étapes permettant de détecter des correspondances non détectables par les techniques classiques de négociation et d'alignement offrant ainsi une précision supérieure par rapport aux approches existantes. Cette phase d'alignement se termine par une génération complète d'un contrat de qualité de service en cas de compatibilité. Concernant la troisième phase traitant la surveillance des contrats de qualité de service, nous avons remarqué l'absence d'une approche automatique et générique pour faciliter la tâche d'observation, d'analyse et de détection des violations de contrats de QdS. Pour remédier à ces insuffisances, nous nous sommes basés sur les mêmes capacités sémantiques que la phase de négociation. En effet, l'approche de surveillance de QdS que nous avons définie se base sur un raisonnement par inférence qui nous permet de détecter des violations et même des dégradations non détectables sans sémantique.

Title

Intention based Semantic approach for service level agreements modeling, negotiation and monitoring.

Keywords

Service Level Agreements; Semantic objectives Matching; Ontologies; Intentions; Service Oriented Architectures; Quality of Service monitoring.

Summary

This work adresses the service level agreements (SLA) life cycle in service-oriented architectures. This life cycle starts with a specification and a modeling phase of the required quality levels. The second phase of this life cycle consists in negotiating these levels. This phase is accomplished by signing a service level agreement by the client and the provider. The last phase consists in monitoring the obligations defined in the SLA until the termination of its validity period. Concerning the first phase, we noticed that the existing SLA models are incomplete especially if the required service levels have to be indirectly computed according to other elementary quality metrics. To overcome this problem, we defined a new SLA model named SLAOnt. This model is characterized by its semantic richness and completeness. Regarding the second phase, we noticed that the existing SLA negotiation approaches consist in choosing a subset of clauses and values among choices predefined by the provider. However, the client may not understand these choices if he is not an expert of the IT domain. In this case, he has not the possibility of expressing his needs with his own knowledge and language. For this reason, we propose a novel negotiation approach based on a semantic matching of quality of service objectives between clients and providers. This approach is based on semantic inferences in all its stages to detect indirect correspondences that are not detectable by the classic negotiation and matching techniques. Our approach provides higher precision when compared to the existing ones. This phase ends with a generation of a complete service level agreement in case of compatibility. Concerning the monitoring phase, we noticed a lack of an automatic and a generic approach to facilitate the task of supervision, analysis and detection of the SLA violations. To address these shortcomings, we relied on the same semantic capacities as the negotiation phase to establish a complete monitoring approach of the signed service level agreements between service providers and consumers. Indeed, our QoS monitoring approach is based on inferential reasoning allowing us to detect violations and even undetectable degradations without using semantics.

SOMMAIRE GENERAL

LISTE DES FIGURES

LISTE DES TABLEAUX

INTRODUCTION GENERALE

SOMMAIRE

I. CONTEXTE DU TRAVAIL

Les architectures orientées services (SOA) sont aujourd'hui envisagées par de nombreuses entreprises dans le cadre de l'évolution de leur système d'information. Au travers de cette architecture, une nouvelle souplesse et une meilleure résilience sont notamment apportées en termes de disponibilité de ces systèmes. Cependant, les pressions qui s'exercent sur le marché, la mondialisation, les exigences de la conformité réglementaire, la concurrence féroce, la demande accrue de services de la part de la clientèle présentent de nombreuses difficultés en termes d'établissement de contrat entre le client et le fournisseur de service et le degré de la qualité de service (QdS) offert par le fournisseur. De ce fait, la garantie des performances dans les architectures SOA est devenue indispensable pour protéger les droits des clients vis-à-vis de leurs fournisseurs. Dans ce cadre, les accords de qualité de service (SLA : Service Level Agreements) forment une solution adéquate pour spécifier ces garanties de qualité de service. La négociation et la surveillance de ces accords de qualité de service pour la gestion des relations entre les fournisseurs de services et leurs clients est un domaine qui attire l'attention de plusieurs chercheurs et industriels dans les architectures orientées services. C'est dans ce contexte que s'inscrit la problématique de cette thèse.

II. PROBLEMATIQUE

Dans ce travail de thèse, nous nous intéressons principalement aux étapes de négociation et de surveillance de SLA. Concernant la phase de négociation, nous nous focalisons précisément sur la première étape de cette phase qui traite la réception de la demande du client chez le fournisseur. Le traitement de ces demandes reste une tâche assez fastidieuse qui nécessite un temps d'analyse conséquent pour vérifier si le produit du fournisseur correspond aux besoins du client. Dans la procédure de négociation habituelle, cette première étape consiste généralement à sélectionner un sous-ensemble de clauses et de valeurs parmi des choix prédéfinis par le fournisseur. Cependant, le client peut ne pas comprendre ces offres surtout lorsqu'il n'est pas un expert du domaine.

Concernant la phase de la surveillance, nous nous focalisons sur le suivi et la vérification des obligations spécifiées dans les contrats de qualité de service afin de réagir aux problèmes de violations ou de non respect de ces contrats. Cette phase a une grande importance pour le maintien des accords de la qualité de service élaborés entre le client et le fournisseur.

Vu ces constatations, les questions suivantes s'imposent :

- Comment donner la possibilité au client d'exprimer ses demandes avec son propre langage ?
- Comment vérifier la correspondance des besoins du client avec les offres du fournisseur ?
- Comment assurer la génération complète du contrat de qualité de service en cas de compatibilité ?
- Comment assurer la surveillance de qualité de service des contrats en cas de correspondances ?

Pour apporter des éléments de réponse à ces questions, nous nous sommes fixés des objectifs que nous présentons dans la section suivante.

III. OBJECTIFS

Le premier objectif de notre travail est d'offrir un moyen qui permet l'expression facile des besoins des clients par leurs propres langages et leurs propres connaissances. Ceci crée une tâche supplémentaire considérable du côté des fournisseurs qui consiste à comprendre les besoins des clients et à vérifier leurs correspondances avec ses offres. Ainsi, notre deuxième objectif consiste à automatiser au maximum cette tâche pour les fournisseurs en vue d'une génération complète d'un contrat en cas de compatibilité. Ce contrat doit être généré selon une structure riche et complète pour protéger les droits des parties concernées. La définition de cette structure constitue notre troisième objectif. Une fois le contrat élaboré et approuvé, notre quatrième objectif consiste à offrir tous les moyens nécessaires pour la surveillance de ses différentes clauses et à déclencher les actions convenables en cas de violation. Suite à ces objectifs, nous présentons les principales contributions de cette thèse dans la section suivante.

IV. CONTRIBUTIONS

La première partie de cette thèse concerne la négociation de contrats de QdS. Les approches existantes de négociation consistent à choisir un sous-ensemble de clauses et de valeurs parmi des choix prédéfinis par le fournisseur. Cependant, le client peut ne pas comprendre ces choix. En plus, il n'a pas la possibilité d'exprimer ses besoins avec ses connaissances et son langage. Pour cela, nous avons choisi de résoudre cette problématique par unification d'objectifs sémantiques de QdS pour l'élaboration de ces contrats. Ces objectifs se basent sur une représentation sémantique des intentions du client et du fournisseur. Chaque intention est exprimée librement dans le domaine de compétence de l'un ou de l'autre. Le processus d'unification de ces intentions se base sur des correspondances linguistiques, heuristiques et sémantiques. Nous avons clôturé cette partie par le développement d'un outil qui implante ce processus d'unification permettant de générer automatiquement un contrat de qualité de services en cas de compatibilité des intentions.

La deuxième partie de cette thèse traite la surveillance des contrats élaborés entre le client et le fournisseur pour détecter toutes leurs violations. Les mécanismes d'observation, d'analyse et de détection de violations de cette phase bénéficient de la même capacité sémantique que la phase de négociation. En effet, l'approche sémantique que nous avons définie se base sur un raisonnement par inférences qui nous permet de détecter des violations et même des dégradations non détectables sans sémantique. Dans cette partie, nous avons élaboré un modèle de SLA nommé *SLAOnt* pour la surveillance des différentes clauses définies dans le contrat. Dans ce modèle, nous pouvons exprimer

des dépendances sémantiques entre les paramètres de QdS. Ces dépendances peuvent être de type statistique (comme le calcul de moyennes par exemple), de type logique (comme des opérateurs existentiels) ou de type inférence (paramètre déduit par inférence à partir d'autres paramètres). Les obligations qui définissent les clauses de ce contrat sont exprimées en utilisant un langage d'inférence sémantique. Ceci nous a facilité le processus de leur surveillance et l'application automatique des actions à effectuer en cas de violations ou dégradations. Nous avons couronné cette deuxième partie par le développement d'une architecture orientée services qui réalise la surveillance automatique des contrats définis selon notre modèle (instances de *SLAOnt*).

Dans la section suivante, nous présentons un exemple de motivation pour mieux illustrer nos objectifs et nos contributions.

V. EXEMPLE DE MOTIVATION

Avec l'évolution de la technologie, les clients peuvent louer et même télécharger des films via internet sans avoir besoin à se déplacer. Ces services de location et de téléchargement offrent différents débits de transfert. Les tarifs de ces prestations varient selon le débit offert par le fournisseur à ses clients pour la visualisation ou le téléchargement d'une part, et du type et de la date d'apparition des films d'autre part. Par exemple, un fournisseur de bibliothèques numériques offre deux services : un service de téléchargement de films et un service de recherche de films. Le fournisseur impose deux contraintes sur ses offres : une contrainte fonctionnelle sur les types de ses films qui peuvent être de type comédie ou d'action et une contrainte non fonctionnelle concernant le *débit* qui est fixé à 20 Mb/s. Les prix des films sont fixés par ce fournisseur à 2 Euros. Ces films peuvent avoir deux tailles différentes : 700 Mo et 900 Mo. Nous supposons que le client qui n'est pas un expert dans le domaine des technologies informatiques veut télécharger des films de type comédie avec un *temps de téléchargement* inférieur à dix minutes et un prix maximum égal à trois euros par film. Nous constatons que ce fournisseur peut satisfaire la contrainte du prix proposée par le client, qui est égale à deux euros et la contrainte du type de film comédie. En ce qui concerne la contrainte du temps de téléchargement, comme nous remarquons dans ce cas, le client ne peut exprimer ses exigences qu'en termes de *temps de téléchargement* puisqu'il ne comprend pas les aspects techniques et les significations du terme *débit*. La correspondance entre ces deux qualités de service : *temps de téléchargement* et *débit* se base sur une dépendance en termes de rapport mathématique pour en déduire la valeur de l'un ou de l'autre. La problématique qui se pose est donc comment détecter ce type de correspondance et comment évaluer les offres du fournisseur par rapport aux demandes du client ? En cas de compatibilité entre les besoins du client et les offres du fournisseur, comment assurer un moyen automatique confiant au client par rapport aux qualités exigées dans le contrat ?

VI. ORGANISATION DU DOCUMENT

Après cette introduction générale, nous présentons dans cette section l'organisation de notre document. Ce mémoire de thèse comporte quatre chapitres principaux. Dans le premier chapitre, nous commençons par présenter un aperçu sur les travaux existants du domaine des accords de qualité de service. Le deuxième chapitre traite l'intégration de la sémantique dans le cycle de vie des accords de qualité de service. Dans le troisième chapitre, nous détaillons nos contributions dans ce domaine en termes de modélisation et de négociation de contrat de qualité de service entre le client et le fournisseur et en termes de surveillance des obligations de ce contrat. Avant de conclure, dans le quatrième chapitre, nous validons nos contributions par une implantation pour mettre en œuvre notre travail.

CHAPITRE I - ETAT DE L'ART : TRAVAUX RELATIFS AU CYCLE DE VIE DES CONTRATS DE QDS

SOMMAIRE DU CHAPITRE

I. INTRODUCTION

Dans ce chapitre, nous présentons quelques notions de bases ainsi que les travaux existants relatifs aux contrats de qualité de service *(SLA : Service Level Agreements)* dans les architectures orientées services *(SOA : Service Oriented Architecture)*. Nous commençons, dans la section II, par donner une introduction sur ces architectures et leurs caractéristiques. Après, dans la section III, nous décrivons les principes des accords de qualité de service dans les architectures *SOA*. Nous analysons, ensuite, les spécifications existantes des accords de qualité de service. Dans la section V, nous exposons quelques protocoles et projets existants de négociation des contrats de qualité de service. Avant de conclure, dans la section VI, nous présentons quelques approches existantes relatives à la surveillance des SLA. Dans la conclusion, nous soulignons l'importance de la sémantique dans le processus d'élaboration des accords de qualité de service.

II. ARCHITECTURES ORIENTEES SERVICES

1. Qu'est ce qu'un service Web ?

Dans le cadre de la programmation orientée service, un service peut être défini comme une entité fonctionnelle auto-contenue, auto-décrite, indépendante des plateformes, et pouvant être décrite, publiée, découverte, invoquée, composée à l'aide de protocoles standards. La technologie des services Web constitue une approche pour la concrétisation du paradigme de service sur le Web. Par le biais de cette technologie, il devient possible de délivrer et de consommer des services logiciels sur le Web, détournant l'usage premier de celui-ci qui ne consistait initialement qu'à publier et à lire des documents et des informations. Le W3C *(World Wide Web Consortium)* définit le service Web de la manière suivante [1] : Un service Web est un système logiciel identifié par un identificateur uniforme de ressources *URI*, dont les interfaces publiques et les liens *binding* sont définis et décrits en XML. Un service Web peut être découvert et sélectionné dynamiquement par d'autres systèmes logiciels. Ces derniers peuvent, ensuite, interagir avec le service sélectionné en utilisant des messages XML transportés par des protocoles Internet.

Les services Web se basent sur un modèle de trois couches : un protocole de communication permettant de structurer les messages échangés entre les composants logiciels, une spécification de description des interfaces des services et enfin une spécification de publication et de localisation de service. Les services Web sont le fondement de base des architectures orientées services.

2. Qu'est ce que une architecture orientée services ?

Une architecture orientée services *(SOA)* est un style d'architecture fondé sur la description de services et de leurs interactions. Les caractéristiques principales d'une architecture orientée services sont le

couplage faible, l'indépendance par rapport aux aspects technologiques et l'extensibilité. La propriété de couplage faible implique qu'un service n'appelle pas directement un autre service. Les interactions sont gérées par des fonctions externes d'orchestration. L'indépendance par rapport aux aspects technologiques est obtenue grâce aux interfaces d'utilisation offertes par le fournisseur du service. Enfin, l'extensibilité est rendue possible par le fait que de nouveaux services peuvent être découverts et invoqués à l'exécution.

La notion d'architecture orientée services n'est pas nouvelle, elle est apparue dès le développement des approches client/serveur. Les définitions sont nombreuses et nous retiendrons les suivantes :

– "Une architecture orientée services est un style d'architecture qui aide les organisations à partager la logique métier et les données entre plusieurs applications et selon différents modes d'utilisation.", donnée en 1996 par le groupe Gartner [2].

– "SOA permet une intégration flexible des applications et des ressources: (1) en représentant chaque application ou ressource en tant que service avec une interface normalisée, (2) en permettant aux services d'échanger des informations structurées (messages, documents, objets métier »), et (3) en coordonnant les services afin de s'assurer qu'ils peuvent être invoqués, utilisés et modifiés de manière efficace par des utilisateurs ou par d'autres services [3]."

– Dans le modèle de référence OASIS [4], "SOA est un paradigme pour organiser et utiliser des ressources métiers distribuées qui peuvent être sous le contrôle de différents domaines. Il fournit un moyen uniforme pour offrir, découvrir, interagir et utiliser ces ressources afin de produire des résultats souhaités compatibles avec des conditions et des attentes mesurables".

Malgré le manque de spécification officielle pour définir une architecture orientée services, trois rôles clés sont communément identifiés : producteur de services, annuaire de services et consommateur de services. L'interaction entre ces trois rôles est décrite par la Figure 1 [4].

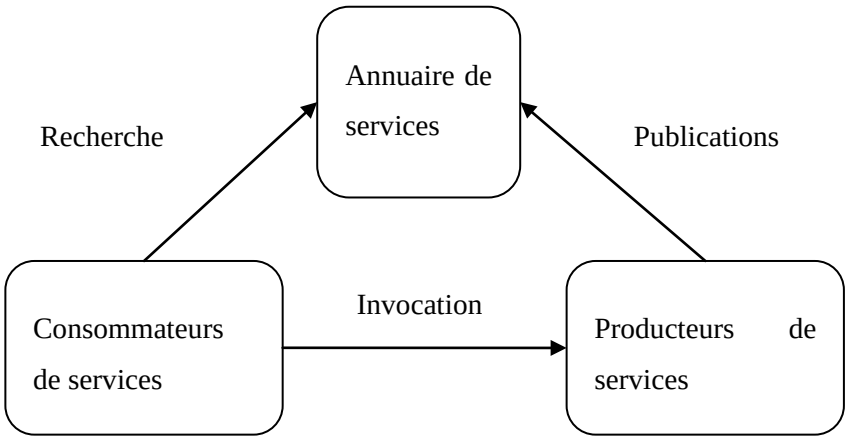

Figure 1 – Architecture orientée services

Le producteur de services a pour fonction de déployer un service sur un serveur et de générer une description de ce service. Ce dernier précise à la fois les opérations disponibles et leur mode

d’invocation. La description de ce service est publiée dans un annuaire. Les consommateurs de services peuvent découvrir les services disponibles et obtenir leur description en lançant une recherche sur l’annuaire. Un consommateur peut alors utiliser la description du service obtenue pour établir une connexion avec le fournisseur et invoquer les opérations du service souhaité.

Bien que les *SOAs* n’aient vraiment pris leur essor que ces dernières années, il existe des propositions plus anciennes sur ce même modèle telles que *CORBA* [5] ou *DCOM* [6]. Les *SOAs* sont en effet plus connues sous leur version services Web *(Web Services Oriented Architectures ou WSOA)* car elles reposent sur des technologies basées sur des standards *XML* et offrent ainsi une solution indépendante des implantations, plateformes, langages ou encodage de données propriétaires. Les trois protocoles de base sont *WSDL* [7], *UDDI* [8] et *SOAP* [9]. Le standard *WSDL* (*Web Service Description Language*) est un langage reposant sur la notation XML permettant de décrire les services Web. *WSDL* permet ainsi de décrire l'emplacement du service Web ainsi que les opérations (méthodes, paramètres et valeurs de retour) que le service propose. Le standard *UDDI (Universal Description Discovery and Integration)* défini par l'OASIS vise à décrire une manière standard de publier et d'interroger les services Web au sein d'un service d'annuaire. Et enfin, *SOAP* est un protocole défini à l'origine par Microsoft, puis standardisé par le W3C, utilisant la notation *XML* permettant de définir les mécanismes d'échanges d'information entre des clients et des fournisseurs de services Web. Dans la section suivante, nous présentons les principes de base de la qualité de services dans les architectures *SOA.*

3. Notions générales sur la qualité de service (QdS)

Avec la prolifération des services Web, la notion de QdS émerge aujourd’hui. Son importance pour les fournisseurs et les clients de services devient de plus en plus importante. En parcourant la littérature, nous avons remarqué qu’il n’existe pas de consensus sur la définition de la qualité de service [12]. La recommandation ITU-X.902 [10] définit la QdS comme un ensemble d’exigences dans le comportement collectif d’un ou plusieurs objets. Dans le contexte des technologies de l’information et multimédia, la QdS a été définie par Vogel et al. [11] comme l’ensemble des caractéristiques quantitatives et qualitatives d’un système multimédia, nécessaires pour atteindre la fonctionnalité requise par l’application. On peut aussi dire [12] que la qualité de service représente l’aptitude d’un service à répondre d’une manière adéquate à des exigences, exprimées ou implicites, qui visent à satisfaire ses usagers. Ces exigences peuvent être liées à plusieurs aspects d’un service, par exemple : sa disponibilité, sa fiabilité, etc.

III. PRINCIPES DES CONTRATS DE QUALITE DE SERVICE DANS LES SOAS

Compte tenu de l'importance des services et de la dynamicité des systèmes dans les architectures orientées services, il est nécessaire de spécifier les relations qui existent entre fournisseurs et consommateurs de services. À la base, ces relations contractuelles ne sont pas réellement explicitées. Ensuite, des contrats (ou *Service Level Agreement - SLA*) sont justement définis pour expliciter et identifier les besoins et les engagements des différentes parties sur la réalisation des services. Ces contrats sont généralement bilatéraux et traduisent ainsi d'une façon formelle le fait que, d'un côté, les fournisseurs s'engagent à fournir des services aux qualités spécifiées à la condition que les directives de l'utilisation des services soient suivies et d'un autre côté, les consommateurs acceptent de respecter les directives d'utilisation pour pouvoir alors bénéficier des qualités de service garanties.

1. SLA : Définition

SLA *(Service Level Agreement)* est une expression anglaise qui peut être traduite par « contrat de qualité de Service » ou par « Engagement de qualité de Service », ou encore plus simplement « Convention de Service » [13]. Nous retenons qu'un SLA est un contrat définissant les engagements du fournisseur quant à la qualité de sa prestation et les pénalités engagées en cas de manquement. Cette qualité doit être mesurée selon des critères objectifs acceptés par les deux parties (client et fournisseur) comme par exemple, le temps de rétablissement du service en cas d'incident [14]. Un SLA contient donc un ensemble d'accords communs entre le client et le fournisseur sur les niveaux de qualité de la prestation de ce dernier.

Les SLA ne constituent pas un nouveau concept et sont déjà traditionnellement utilisés dans plusieurs domaines lorsqu'il s'agit, par exemple, de contractualiser la réservation de ressources. En revanche, ces contrats sont établis manuellement par des échanges et des signatures de documents papiers. Ainsi, un défi majeur dans les *SOA* réside dans l'automatisation du processus complet de contractualisation : de la création des contrats dans des formats traitables par la machine jusqu'à leur surveillance et leur gestion au cours de l'exécution des services, en passant par des mécanismes automatisés permettant de négocier ces contrats pour établir ou maintenir des formes d'accord.

2. Structure des accords de qualité de service

Un SLA est composé généralement de 3 sections [15] (voir Figure 2) :

i. La première section (*Parties*) contient les parties impliquées dans le contrat : les parties signataires (le fournisseur de service et son client) et les parties tierces qui participent à la surveillance de SLA.

ii. La deuxième section (*ServiceDescription*) présente la description des services. Cette partie contient les opérations du service, leurs messages d'entrée/sortie et l'encodage du transport de ces messages.

Elle contient aussi les paramètres de SLA représentant les variables de qualité de service qui vont être utilisées dans la spécification des obligations du contrat. Ces paramètres se basent sur des métriques évaluées par des directives de mesure. Ces dernières définissent comment les métriques sont mesurées. Le dernier élément (*schedule*) de cette deuxième partie du contrat spécifie la durée et la fréquence des mesures des qualités de service.

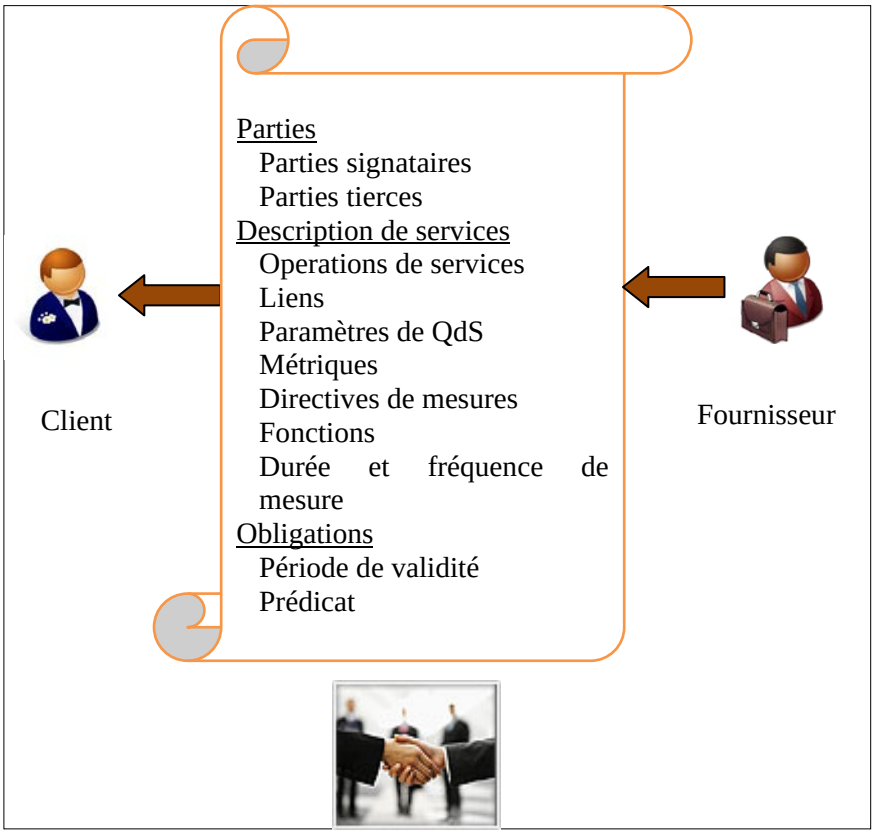

Figure 2 – Structure d'un SLA

iii. La troisième section (*Obligation*) présente les obligations du contrat : sa période de validité, les conditions qui spécifient ces obligations et les actions à prendre en cas de non respect du contrat. Le SLA est accompagné d'une partie technique, appelée *SLS* (*Service Level Specification*) [125], qui définit les paramètres négociés entre les différents domaines afin de se mettre d'accord sur un niveau de QdS. Les principaux paramètres utilisés dans un SLA sont :

• Temps de service : Spécifie le temps pendant lequel la QdS négociée doit être garantie.

• Paramètres de QdS : Délai, gigue, taux de perte, bande passante.

• Profil du trafic : Taille des paquets.

• Traitement d'excès : Spécifie le traitement que le réseau applique aux paquets hors profils.

• Identification du trafic : Adresse IP source et destination, port source et destination, identification du protocole.

• Identification du client : Utilisée pour les fonctions d'authentification, d'autorisations et de calcul de

coût (AAA *: Authentication, Authorization, Accounting*).

• Mode de négociation : Prédéfini ou non, c'est-à-dire que les paramètres de SLS sont contraints par le fournisseur ou prennent des valeurs quelconques.

• Intervalle de renégociation : L'intervalle de temps pendant lequel un SLS négocié ne peut être renégocié.

• Fiabilité : Temps d'inaccessibilité et temps de rétablissement d'un service suite à une panne.

3. L'établissement des accords de qualité de service

Bien que les contrats électroniques aient pour but de formaliser des accords acceptés mutuellement par les deux parties (fournisseurs et consommateurs de services) le processus d'établissement des contrats reste cependant bien souvent asymétrique [16] et contrôlé par les fournisseurs. Ce processus, sous sa forme la plus générale, se décompose selon la Figure 3 en :

– Fournisseurs de services qui créent des gabarits de contrats (*template*) définissant notamment la liste des services offerts et les niveaux de services associés et intègrent parfois les coûts financiers des services ainsi que des pénalités en cas de violation des termes du contrat. Ces gabarits de contrats sont envoyés aux consommateurs,

– Consommateurs qui sélectionnent les services et créent une instance de contrat. Ce contrat est renvoyé et soumis à validation par les fournisseurs.

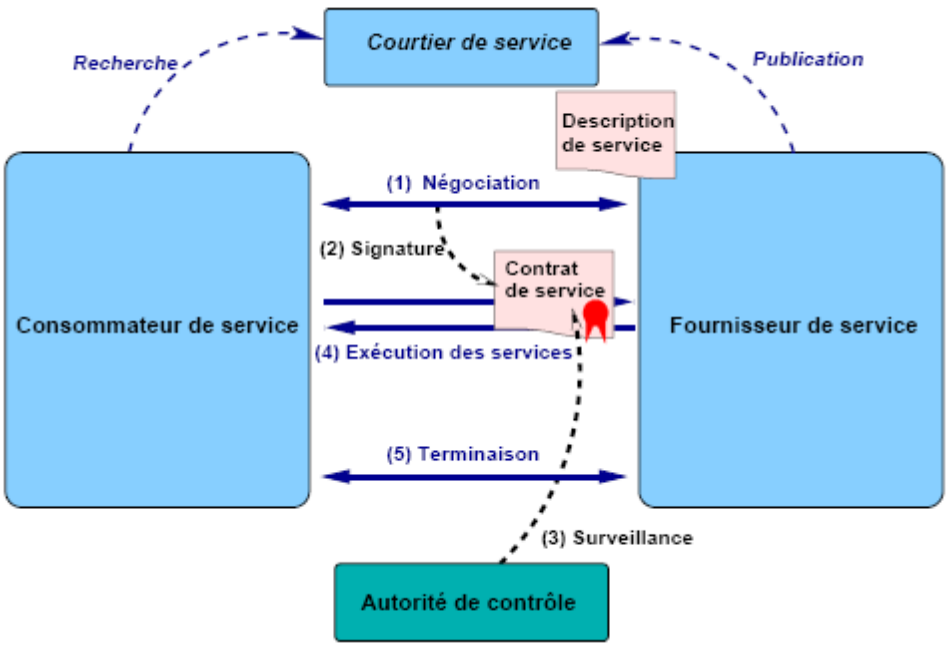

Figure 3 – Architecture de gestion d'un SLA [17]

Ainsi, les fournisseurs et les consommateurs peuvent entrer dans un processus de négociation basé sur des échanges successifs de tels documents qui expriment les contrats souhaités par les consommateurs

et ceux acceptés par les fournisseurs. Bien évidemment, cela implique que les fournisseurs de services aient les capacités pour supporter et proposer différents types de services et que les deux parties puissent dialoguer durant ce processus de négociation. Par la suite, un ensemble de systèmes sont intégrés pour surveiller et gérer les contrats négociés lors de l'exécution des services.

Le cycle de vie habituel [18] appliqué à cette architecture est composé des phases suivantes (comme présenté dans la Figure 4) :

1**.** Le développement d'un modèle (*template*) du contrat

2. La négociation du contrat

3. L'implantation du contrat et son établissement

4. L'exécution du contrat et la surveillance des performances engagées

5. La terminaison ou la fin du contrat.

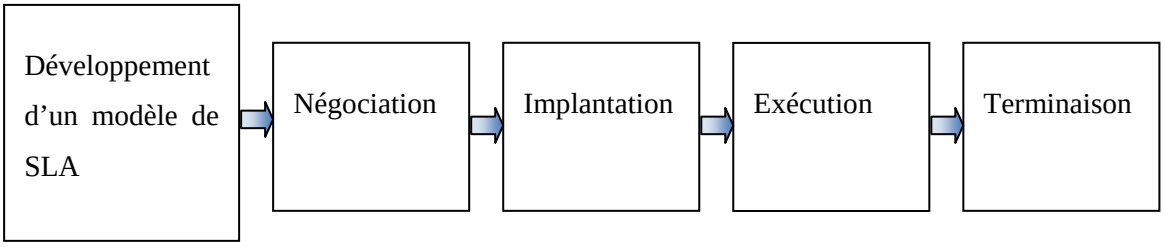

Figure 4 – Le cycle de vie d'un SLA [18]

Il est évident que tout le cycle de vie d'un SLA repose sur un *template* ou un modèle utilisé pour la représentation de ses différentes clauses. Dans la section suivante, nous analysons les spécifications et les modèles existants relatifs aux accords de qualité de service.

IV. MODELISATIONS DES ACCORDS DE QDS

1. Spécifications existantes des accords de QdS

Différents langages ont été proposés pour implanter les spécifications des *SLAs*. Ces langages se focalisent notamment sur l'expression des capacités des fournisseurs de services et des contraintes de qualité de service. Parmi ces langages, nous citons *WSLA* [15], *WSOL* [19], *SLAng* [20] *et WS-Agreement* [21].

1.1 WSLA [15]

Le canevas *WSLA* (Web Service Level Agreement) propose un langage de spécification de SLA basé sur les XML Schémas [22]. Il décrit les parties impliquées, la description des services (caractéristiques et paramètres observables) et les obligations. Le méta modèle sous-jacent [23] au canevas *WSLA* est présenté dans la Figure 5. Le langage *WSLA* introduit en particulier :

– des paramètres de SLA, qui sont des propriétés (généralement non fonctionnelles) du service. Ces

paramètres sont l'équivalent des QdS (disponibilité, temps de réponse etc.).

– des métriques de QdS qui agrègent d'autres métriques d'une façon récursive. Une métrique peut être définie par une directive de mesure qui est une fonction permettant de récupérer la valeur de la métrique.

A partir de ces éléments, il est possible d'exprimer des objectifs de niveaux de *services (Service Level Objectives - SLO)* et des actions à effectuer. Ces objectifs de niveaux de services sont exprimés dans une logique propositionnelle, donc avec une expressivité limitée par rapport à des assertions exécutables classiques. La Figure 6 présente un exemple d'objectif d'un niveau de services dans un scénario de réservation de voyage. Cet exemple indique que le temps de réponse du service de réservation doit être inférieur à 0.5 seconde.

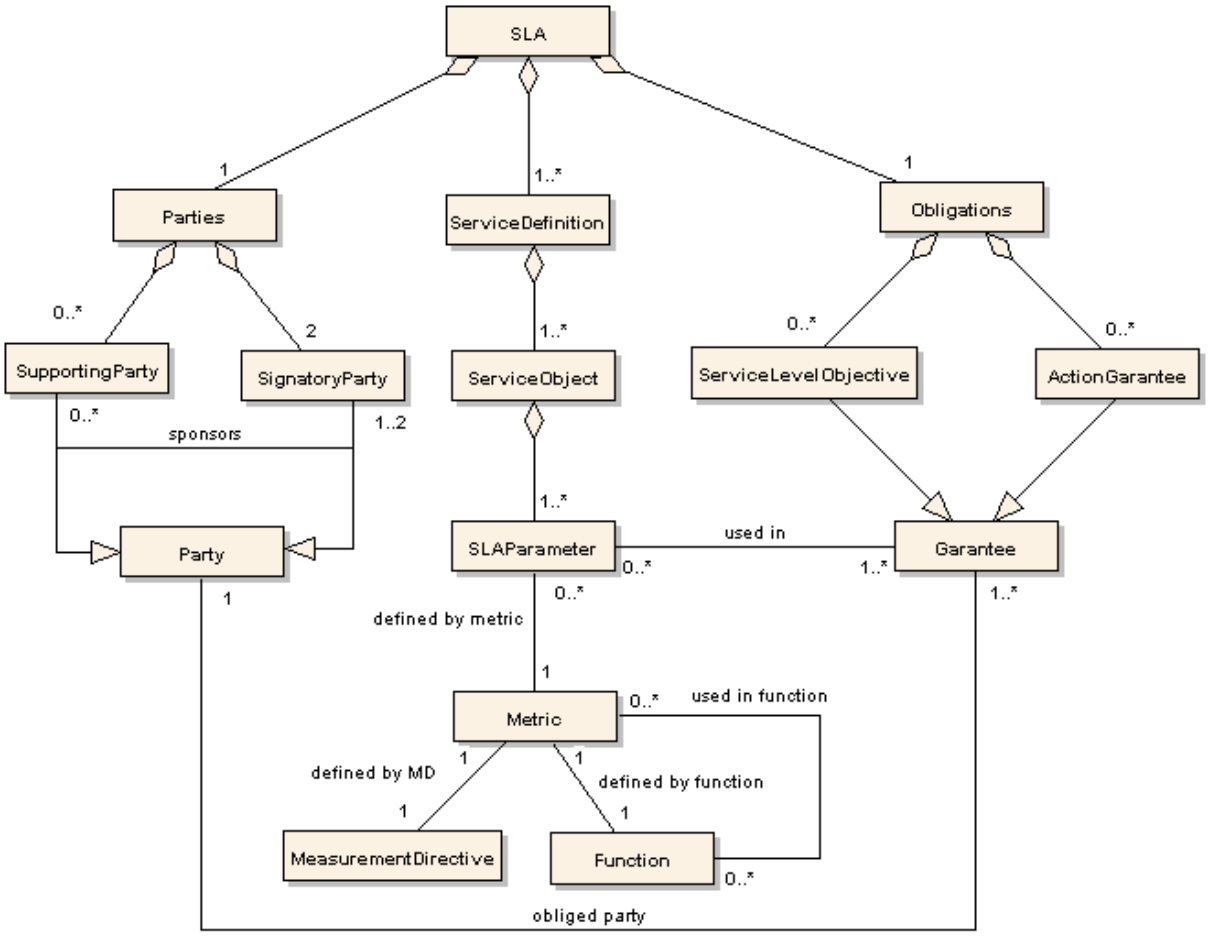

Figure 5 – Modèle de contrat WSLA [23]

```
1.    <Expression>
2.      <Predicate xsi:type="Less">
3.        <SLAParameter>ResponseTime</SLAParameter>
4.        <Value>0.5</Value>
5.      </Predicate>
6.    </Expression>
```

Figure 6 – Un exemple d'un objectif de qualité de service exprimé avec WSLA

1.2 WSOL [19]

WSOL (*Web Service Offerings Language*) est un langage basé sur *XML* qui permet de spécifier des niveaux de services rattachés aux descriptions *WSDL* des services. *WSOL* s'appuie sur la notion de classes de services qui représentent des SLA simples dans lesquels les consommateurs choisissent un service donné parmi ceux proposés par les fournisseurs. *WSOL* s'appuie sur la définition des dimensions de qualité, des métriques et des contraintes de qualité. La Figure 7 présente un exemple exprimé avec le langage *WSOL*.

```
<wsol:serviceOffering name="SO1" service= "buyStock:buyStockService"
   accountingParty="WSOL-SUPPLIERWS">
 ...
 <wsol:instantiate CGTName="CGT2" resService="..."
   resPortOrPortType="..." resOperation="..." resCGName="CG5">
  <wsol:parmValue name="maxResTime">
    <wsol:numberWithUnitConstant>
     <wsol:value>30</wsol:value>
      <wsol:unit type="QoSMeasOntology:millisecond"/>
    </wsol:numberWithUnitConstant>
   </wsol:parmValue>
 </wsol:instantiate>
</wsol:serviceOffering>
```

Figure 7 – Exemple d'un service offert de WSOL [19]

Cet exemple décrit le temps de réponse d'un service. Ce temps a une valeur maximale qui ne doit pas dépasser les 30 millisecondes. En effet, les contraintes de QdS ne sont pas extensibles et sont prédéfinies à l'avance comme *maxResTime* et *minThroughput*. Ainsi, la définition de nouvelles contraintes oblige le développeur à ajouter d'autres éléments dans ce langage et d'autres entités logicielles pour pouvoir comparer les valeurs mesurées par rapport aux seuils définis. Ceci est dû principalement à l'absence des opérateurs de comparaison dans la définition de ces contraintes.

1.3 SLANG [20]

SLAng est un langage de définition d'accords de qualité de service [24] avec une focalisation sur les services réseaux et middleware et sur le maintien de la QdS de bout en bout. Dans *SLAng*, nous retrouvons des caractéristiques classiques de définition de paramètres pour les SLA et un schéma XML décrivant les parties signataires et les contraintes de qualité de service. Les objectifs de *SLAng* sont multiples. Ce langage tente de fournir des moyens de définir les propriétés de QdS de manière non ambigüe et de les intégrer dans des accords. Un des avantages de *SLAng* est l'expression des accords de niveau de service dans un langage XML, ce qui offre une compatibilité avec les technologies Internet. Toutefois, *SLAng* n'est pas un langage extensible car il se base sur un ensemble

de QdS prédéfinis. Ceci limite son utilisation vu le manque de flexibilité et l'effort considérable pour la définition de nouvelles QdS.

1.4 WS-AGREEMENT [21]

WS-Agreement est une autre spécification qui s'intéresse à la création de contrats, d'accords et de garanties à partir des offres entre un fournisseur de services et un client. C'est un langage basé sur *XML* décrivant un SLA *(Service Level Agreement*) dans un environnement de *Grid Computing*. Il constitue un effort de standardisation et d'unification des nombreux langages existants dans le but de définir un langage et un protocole qui permettent de publier les capacités des fournisseurs de services (*templates*), de créer les accords à partir des offres et de surveiller les contrats établis. Toutefois, ce langage se basant sur des chaînes de caractères libres sans aucune contrainte pourrait engendrer des incompatibilités et des incompétudes lors des spécifications des accords de QdS. Par exemple, dans l'extrait suivant : *<wsag:Item> <Location>FileSizeLimit</Location> <xs:restriction base="xs:positiveInteger"> <xs:maxInclusive value="524288000"/> </xs:restriction> </wsag:Item>*, l'expression *FileSizeLimit* ne donne pas une précision sur la nature du fichier concerné et ne fournit aucun moyen pour obtenir sa taille réelle pour pouvoir la comparer avec la limite donnée. En outre, l'unité de cette limite n'est pas précisée. Ceci restreint les possibilités d'une utilisation automatique (par des outils informatiques) d'un accord de QdS exprimé avec WS-Agreement.

2. Bilan sur les spécifications existantes des accords de QdS

Parmi les langages cités ci-dessus, la contribution la plus aboutie et la plus répandue est celle du langage *WSLA* proposé par *IBM* [25]. *WSLA* est une spécification flexible, utile et assez simple pour être utilisée dans des cas réels. C'est le seul langage existant qui est applicable sur plusieurs types de services comme les services Web et les services de stockage. Cependant, les contrats de qualité de service dépassent rarement le niveau syntaxique alors que les fournisseurs et les clients n'ont pas le même degré de connaissances. Ils peuvent ne pas partager le même langage. En conséquence, la négociation du contrat de QdS reste une tâche assez difficile à réaliser avec ce genre de langages.

V. NEGOCIATION DES ACCORDS DE QUALITE DE SERVICE

Dans cette section, nous commençons par présenter quelques définitions existantes du terme négociation en cas général. Par la suite, nous exposons les principaux protocoles existants de négociation de qualité de service. Enfin, nous dressons un bilan où nous illustrons le manque de sémantique dans la structure des messages échangés au sein de ces protocoles.

1. Négociation : Définition et principe

Nous pouvons citer plusieurs définitions pour le terme négociation. Smith (1980) dit que [26] : Par

négociation, on entend une discussion dans laquelle des individus intéressés échangent des informations et arrivent à un accord en commun. Pruitts (1981) [27] donne une définition qui s'appuie sur des considérations psychologiques et pour laquelle le conflit est l'élément de base. La négociation est le processus par lequel plusieurs individus prennent une décision commune. Les participants expriment d'abord des demandes contradictoires, puis ils essaient de trouver un accord par concession ou par la recherche de nouvelles alternatives.

Si on regarde les deux définitions présentées, on peut identifier deux aspects essentiels de la négociation : la communication et la prise de décisions.

Pour modéliser la négociation, il faut alors prendre en compte les aspects suivants :

- Le protocole de négociation : c'est l'ensemble des règles qui régit la négociation : les participants possibles dans la négociation, les propositions légales que les participants peuvent faire, les états de la négociation (par exemple l'état initial où commence la négociation, l'état où on accepte des soumissions ou la fin de la négociation) et une règle pour déterminer quand on est arrivé à un accord ou quand il faut s'arrêter parce qu'aucun accord n'a pas pu être atteint.
- L'objet de négociation : c'est un objet abstrait qui comprend les attributs qu'on veut négocier. Dans certains cas, il s'agit de négocier uniquement le prix, mais, dans d'autres cas, il faut aussi négocier plusieurs attributs comme le temps nécessaire pour satisfaire une commande, la qualité des produits etc.
- Le processus de décision est le modèle utilisé pour prendre des décisions pendant la négociation. La partie la plus importante de la prise des décisions, dans ce cas, est la stratégie de négociation qui permet de déterminer quelle primitive de négociation l'agent doit choisir à un certain moment. Le processus de décision revient à répondre à la question : "Que dois-je faire maintenant ?", par exemple liciter enchérir, abandonner, etc.

Dans notre travail, nous nous intéressons principalement aux protocoles de négociation présentés dans la section suivante.

2. Protocoles de négociation

Dans la littérature, nous distinguons différents protocoles de négociations de QdS. Une brève présentation de ces protocoles est donnée ci-dessous [28] :

2.1 Le protocole SrNP

Le protocole SrNP (*Service Negotiation Protocol*) a été défini dans le cadre du projet TEQUILLA [29] pour la négociation de la structure d'inscription des services (SSS *Service Subscription Structure*) qui représente un ensemble de SLS. *SrNP* se base sur une architecture client/serveur et les entités de négociation ne sont que des serveurs *SrNP* et des clients *SrNP*. Ces deux dernières disposent d'un certain nombre de requêtes qui permettent d'établir un accord, de modifier un accord déjà établi et

d'annuler un accord déjà négocié [30]. Ce protocole se focalise sur les actions de négociation (telles que *SessionInit, Proposal* et *AgreedProposal*) sans décrire le contenu des messages échangés. *SessionInit* sert à initialiser une session de négociation. *Proposal* permet de proposer une offre et *AgreedProposal* concerne l'acceptation d'une offre bien déterminée. Cependant, ceci n'aide pas à l'automatisation de la négociation d'une part et peut engendrer des incompréhensions et des mauvais choix d'autre part puisque les intervenants de la négociation peuvent ne pas partager le même langage et les mêmes connaissances.

2.2 LE PROTOCOLE COPS-SLS

COPS-SLS [31] est une extension du protocole de contrôle par politique COPS (*Common Open Policy Service*) [32]. *COPS-SLS* permet une spécification de niveau de service (*SLS*) (*Service Level Spécification* : partie technique du SLA) en se basant sur l'architecture de gestion par politiques. Dans l'architecture COPS-SLS, chaque domaine possède deux entités principales : le *SLS-PDP* (*Policy Decision Point*) et le SLS-PEP *(Policy Enforcement Point*). Le *SLS-PDP* fournit les paramètres à négocier au SLS-PEP qui se charge de les transmettre au *SLS-PDP* d'un autre domaine. La négociation se déroule en trois étapes : (1) l'envoi d'un message REQ (*Request*) de proche en proche, du domaine initiateur jusqu'au domaine destinataire, pour demander des ressources, (2) l'envoi de la réponse DEC (*Decision*) dans le sens inverse pour informer les différents domaines des décisions, enfin, (3) l'envoi du rapport RPT (*Report State*) pour informer sur le succès ou l'échec de la négociation. Cependant, ce protocole ne décrit pas le contenu de ces messages. Ces messages doivent être structurés pour pouvoir améliorer et automatiser la négociation entre le client et le fournisseur.

2.3 LE PROTOCOLE DSNP

Le protocole DSNP (*Dynamic Service Negotiation Protocol*) [33] est un protocole de négociation de SLS (Service Level Spécification) au niveau de la couche IP. La simplicité et la légèreté du protocole DSNP permet son utilisation dans les réseaux sans fil. De plus, il prend en charge des aspects de mobilité pour la négociation de la QdS. L'architecture *DSNP* repose aussi sur un client DSNP et un serveur DSNP qui s'échangent un certain nombre de requêtes (*SLS_List_Request/Response, SLS_Nego_Request/Response, SLS_Stat_Request/Response)* pour réaliser la négociation. Cependant, les messages échangés dans ce protocole se limitent à « Accepter », ou « Rejeter ». Ceci empêche le client d'exprimer librement ses exigences surtout s'il ne partage pas le même langage que le fournisseur.

2.4 LE PROTOCOLE NSLP

Le protocole NSLP [34] est issu des travaux du groupe de travail *NSIS (Next Steps In Signaling)* de l'IETF. L'objectif de ce groupe de travail est de standardiser un protocole pour la signalisation de la QdS. Les protocoles qui existent actuellement sont utilisés comme base de départ pour définir ce

nouveau standard. Les premiers résultats de ce groupe est un modèle structurel composé de deux couches [35] : une couche inférieure, appelée NTLP (*Signaling Transport Layer Protocol*) et une couche supérieure, appelée *NSLP* (Network Signaling Layer Protocol). *NTLP* est responsable de l'acheminement des messages de signalisation dans le réseau, tandis que *NSLP* contient des fonctionnalités spécifiques à l'application comme le type et les séquences des messages. *NSLP* est indépendant de l'architecture de QdS déployée sur le réseau. Comme les autres protocoles, *NSLP* ne tient pas compte du contenu des messages échangés lors de la négociation.

2.5 LE PROTOCOLE CONTRACT-NET

Le Protocole Contract-Net (CNP) a été défini par Smith [26]. C'est un mécanisme de négociation par appel d'offre (ou Contract) entre deux types d'agents : l'agent gestionnaire et les agents contractants. L'agent gestionnaire, souhaitant sous-traiter une tâche qu'il doit accomplir, est l'initiateur du contrat. Chaque agent contractant est un agent auquel on lui propose ce contrat. Ce protocole très souvent utilisé, a été normalisé par l'organisation FIPA [74]. Il se base sur cinq étapes principales :

- Le gestionnaire (considéré comme le client) envoie un appel de proposition pour un certain nombre de participants (considérés comme des fournisseurs). Cet appel est constitué de *n* contraintes demandées par le client.
- Chaque fournisseur vérifie l'appel de proposition reçu en analysant les *n* contraintes définies dans l'appel. Il peut envoyer trois types de messages pour chacune de ces contraintes : *positif* si le fournisseur peut satisfaire la contrainte, *négocier* s'il peut proposer une alternative et *négatif* s'il ne peut pas.
- Le client choisit la meilleure offre et lui envoie un message d'acceptation de proposition.
- Le client envoie aux autres fournisseurs un message de refus.
- Le fournisseur choisi, envoie le contrat final au client.

Malgré sa généricité, ce protocole se base sur un langage technique complexe et peu structuré (chaînes de caractères) pour la définition des contraintes des clients. Ce langage doit aussi être le même pour les fournisseurs sinon, la contrainte sera considérée comme insatisfaite.

3. Projets existants de négociation

Les protocoles présentés dans la section précédente sont utilisés dans plusieurs projets traitant le problème de la négociation de la QdS.

3.1 LE PROJET WS-AGREEMENT

[36] a proposé un composant logiciel (*Ws-agreement* Framework) pour la négociation des contrats implantés selon la structure *Ws-agreement* que nous avons présentée dans la section IV.1.4 de ce chapitre.

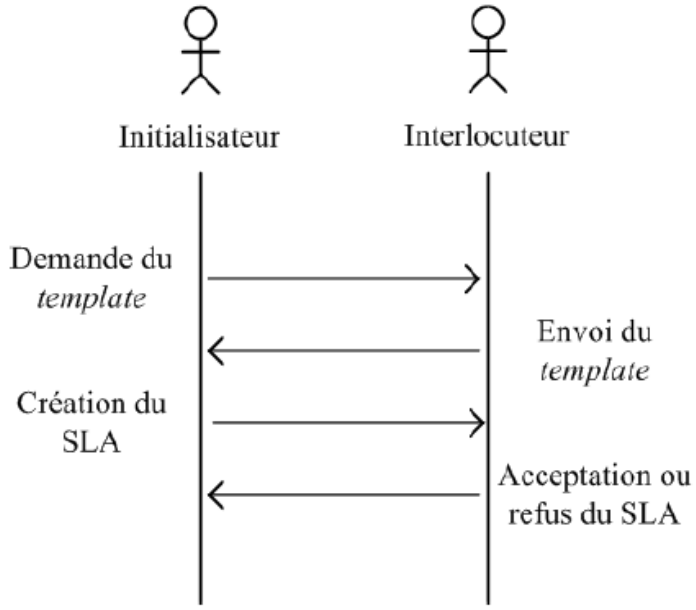

Figure 8 – Processus de négociation de WS-agreement

Ce composant assure la négociation entre deux participants (un initiateur et un interlocuteur) qui dialoguent pour aboutir à un accord. La Figure 8 illustre le processus de négociation réalisé par *Ws-agreement Framework*. Ce processus comprend les étapes principales suivantes : la demande d'un *template* de la part de l'initiateur, la réponse de son interlocuteur, une proposition de SLA sur la base du *template* recueilli de la part de l'initiateur et enfin l'acceptation ou le refus du SLA par son interlocuteur.

3.2 LE PROJET BEINGRID

Le projet *BEINGRID* [37] présente des mécanismes par lesquels les SLA sont diffusés par les fournisseurs et découverts par les clients. *BEinGRID* envisage une situation où les capacités des prestataires de services sont annoncées dans un marché virtuel de services. Les clients découvrent les fournisseurs en inspectant le marché ou en attribuant un courtier de ressources qui analyse les offres existantes.

Le succès d'un tel marché est tributaire de l'existence d'une représentation commune des SLAs afin que les clients puissent interagir avec le marché et surtout comprendre ce qui est fourni par les prestataires de services.

3.3 LE PROJET TRUSTCOM

Le projet *TrustCoM* [38] se focalise sur la publication et la découverte des offres des fournisseurs. Ces deux actions concernent les phases de « préparation » et d' « identification » des capacités de QdS. Dans la première phase, les fournisseurs préparent et enregistrent leurs capacités de QdS sous la forme

d’accords de QdS prédéfinis dans un registre de services.

Les actions de cette phase sont représentées par des événements étiquetés de « 1 » dans la Figure 9. Dans la phase d’identification, le client découvre les offres des fournisseurs à l’aide du registre. Cette découverte est effectuée en envoyant une requête au registre qui répond par la liste des accords de QdS qui correspondent à la requête du client. La phase d’identification est représentée par des événements étiquetés de « 2 » dans la Figure 9.

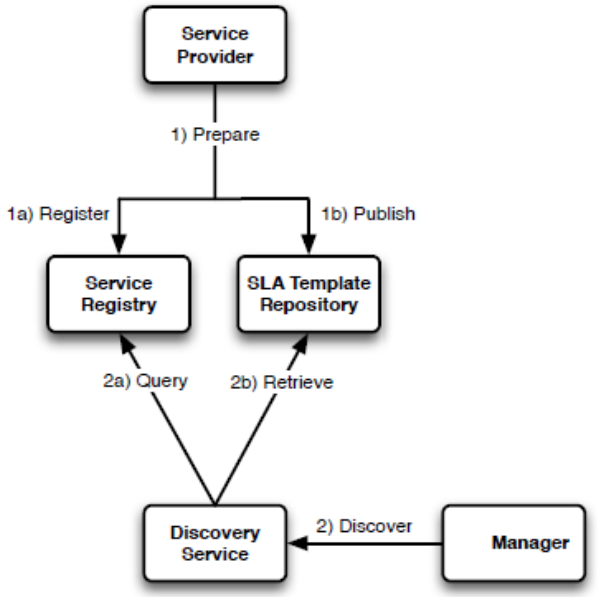

Figure 9 – L’architecture de négociation de TrustCoM

3.4 LE PROJET NEXTGRID

Ce projet [39] présente le mécanisme par lequelles les offres sont publiées et découvertes par les clients potentiels. Ce mécanisme se base sur le même principe de *TrustCoM*. En effet, les fournisseurs publient des SLAs prédéfinis dans un registre accessible par les clients. Ces derniers disposent d’une interface graphique qui permet d’accéder aux services de découverte pour rechercher des SLAs candidats pour la négociation.

L’interface graphique offerte aux clients interagit avec deux autres services : le registre de propositions de SLA par les fournisseurs et le service de découverte qui interroge d'abord le registre pour trouver les fournisseurs capables de répondre aux exigences, puis trouver les modèles de SLA appropriés pour le client.

4. Bilan des travaux existants sur la négociation des accords de QdS

Bien que les contributions que nous avons étudiées soient encore en évolution continue, elles présentent certaines limites. Les approches de négociation prédéfinies par le fournisseur ne décrivent

que de simples messages qui se limitent à *Accept*, ou *Reject*. Les solutions existantes ne donnent pas assez d'importance aux contenus de ces messages. Cependant, ces messages doivent être détaillés pour pouvoir améliorer et automatiser la négociation entre le client et le fournisseur. En effet, l'utilisation des informations contenues dans ces messages permettrait aux machines de prendre des décisions automatiques qui demandent de longues interventions manuelles coûteuses avec les approches de négociation existantes. De plus, les mécanismes qui permettraient d'élaborer de tels accords, y compris par des formes de négociation automatisées, restent à l'heure actuelle peu aboutis et consistent bien souvent à des échanges simples de gabarits de contrats dont la description nécessite encore à être développée. Les échanges entre fournisseurs et clients conduisent à effectuer des choix de services à des qualités données ou des choix de valeurs de certains paramètres dans des échelles prédéfinies par les fournisseurs [40]. Ainsi, il n'y a pas réellement de protocoles d'interaction et de négociation dynamique entre fournisseurs et clients pour l'établissement des contrats. En effet, les solutions existantes se basent sur des choix prédéfinis par le fournisseur. Cependant, le client peut ne pas comprendre ces choix surtout s'il n'est pas un expert.

Dans le cycle de vie de SLA, après avoir négocié et approuvé un contrat de QdS, ce dernier sera utilisé dans la phase de surveillance des accords de QdS afin de vérifier le respect des clauses du contrat par les parties impliquées. Dans la section suivante, nous présentons les principaux travaux relatifs à la surveillance de ces accords définis dans les contrats.

VI. SURVEILLANCE DES ACCORDS DE QUALITE DE SERVICE

Dans cette section, nous commençons par présenter une définition du terme surveillance de QdS. Par la suite, nous exposons les principales infrastructures existantes de surveillances des accords de QdS définis dans les contrats. Enfin, nous dressons un bilan où nous illustrons les insuffisances de ces infrastructures.

1. Surveillance : Définition et principe

Selon [12], la surveillance est « une étape primordiale qui consiste à observer le comportement du service et d'extraire des métriques nécessaires pour effectuer les mesures de sa qualité. La variété potentielle de paramètres de QdS rend cette tâche encore plus délicate ».

Cette définition insiste sur l'importance de la surveillance de la QdS dans les *SOAs*. Elle se focalise sur les tâches de mesure de cette qualité. Néanmoins, elle néglige une étape importante qui est la comparaison des qualités mesurées par rapport aux attentes du client. En conséquence, nous complétons cette définition par celle de [41] : « la surveillance joue un rôle important pour déterminer si un SLA a été violé ou non et quelle pénalité doit on appliquer en cas de violation».

2. Infrastructures de surveillance d'accords de QdS

Vu l'importance de l'étape de surveillance dans le cycle de vie des SLA, les fournisseurs et les clients ont besoin d'une infrastructure permettant de suivre les contrats électroniques. Dans la littérature, nous trouvons quelques infrastructures existantes. Ces dernières sont présentées dans les sous-sections suivantes :

2.1 WSLA

Le Web Service Level Agreement (*WSLA*) *framework* [15] est une infrastructure proposée par *IBM Research*. Dans ce *framework*, on propose à la fois un langage de spécification de SLA (comme présenté dans la section 1.1) ainsi qu'une infrastructure pour établir et surveiller les contrats lors de l'exécution des services. Le langage proposé [42] est basé sur *XML Schema*. Il décrit les parties signataires, la description des services (caractéristiques et paramètres observables) et les obligations. Dans cette infrastructure [43], la gestion des contrats est effectuée par le biais d'un ensemble de services qui permettent de surveiller, de mesurer les paramètres de QdS et de déclencher des activités données pour résoudre les problèmes. Ces services sont souvent assurés par des autorités de gestion indépendantes à la fois des clients et des fournisseurs. Cependant, la surveillance du contrat et ses violations éventuelles restent des tâches très peu explorées.

2.2 WSML

Hewlett-Packard a développé un langage basé sur XML pour spécifier les SLAs des services Web [44] [45]. Il s'agit du "*Web Service Management Language*" (*WSML*) qui se base sur *WSDL* [7] et *WSFL* [46]. Un document WSML contient la description d'une période de validité du SLA, des parties impliquées et un ensemble de *SLOs* (*Service Level Objectives*). Les *SLOs* sont des objectifs de qualité de service définis par le client ou par le fournisseur. Un SLO est considéré comme un moyen de mesurer la performance du fournisseur de service. Chaque SLO contient une description du jour et de l'heure durant lesquels le SLO est valide, ainsi qu'un ensemble de clauses. Une clause décrit un ou plusieurs objets (par exemple des opérations) pour lesquels une mesure est effectuée, des temps ou évènements qui déclenchent l'évaluation, des échantillons de mesure utilisés pour l'évaluation, une fonction d'évaluation booléenne et une action effectuée dans le cas où la fonction d'évaluation retourne un résultat négatif. La fonction d'évaluation capture la définition d'une métrique de QdS. Son objectif est de calculer cette métrique en fonction des échantillons de mesure et de vérifier une condition booléenne qui représente une clause du contrat.

2.3 CREMONA

Cremona [47] est une architecture dans laquelle les fournisseurs et les consommateurs de services établissent des accords selon le standard *Ws-Agreement*. L'architecture est constituée principalement

des entités de gestion des différentes phases définies dans *Ws-Agreement*. Elle est implantée en Java. Elle fournit des fonctionnalités de surveillance des accords. Elle définit en outre des descriptions abstraites de ces accords pour les fournisseurs de services qui peuvent être mises en œuvre dans un environnement spécifique au domaine. L'architecture Cremona contient les mêmes limites de *Ws-Agreement* (voir section III.1.4 de ce même chapitre) qui manque de structuration puisqu'elle se base sur des chaînes de caractères libres sans aucune contrainte. Ceci pourrait créer des incompatibilités lors des spécifications des accords de QdS définis dans les contrats.

2.4 AUTRES APPROCHES DE SURVEILLANCE DE SLA

Dans la littérature, il existe de nombreuses approches [48] pour la surveillance de SLA et la détection de violation (exemple [49], [50]). Lodi et al. [51] décrivent un middleware qui assure la classification des serveurs d'applications selon leur QdS. Dans leur approche, ils utilisent le langage XML pour la définition de leur SLA tout en s'inspirant de *SLAng* [20]. Leur architecture comprend trois types de services. Le service de configuration est responsable de la gestion des classes des serveurs d'applications selon leur QdS. Le service de surveillance observe le comportement de l'application pour détecter les violations de SLA. Le service d'ordonnancement intercepte les requêtes des clients et les oriente vers les différentes classes de serveurs.

Chau et al. [49] présente une approche pour la modélisation et la surveillance de SLA. Leur approche fait partie du projet *eQoSystem*. Leur modèle de SLA s'inspire de la spécification de *WSLA*. Il est basé sur un ensemble de *SLOs* et utilise une variété de métriques indiquant différents aspects de mesures. Leur approche est basée sur les événements. Ces événements doivent être envoyés par les processus métiers et doivent contenir un aperçu de l'état du processus courant. Ainsi, les processus métiers doivent mesurer les paramètres de QdS et les envoyer avec les réponses à leurs invocations. Ceci constitue un inconvénient majeur en imposant une contrainte forte sur le développement des différents services offerts aux utilisateurs. Pour des raisons de confiance et de performance, les mesures de QdS doivent être assurées par des entités logicielles externes indépendantes des processus métiers d'un point de vue fonctionnel.

3. Bilan sur la surveillance des accords de QdS

La phase de surveillance des accords de QdS est primordiale pour contrôler en permanence le respect des clauses négociées. Cette phase consiste essentiellement à effectuer le suivi d'entités bien ciblées pour pouvoir détecter les violations des contrats et éventuellement appliquer des pénalités adéquates. Malgré l'importance de cette phase, les architectures existantes proposées pour la gestion des accords de QdS n'explorent que superficiellement la surveillance des différents paramètres négociés. En effet, ces architectures ne permettent pas de surveiller des QdS composées (basées sur d'autres paramètres élémentaires). En plus, les évaluations des clauses du contrat se limitent à des tests booléens simples

qui ne permettent pas de détecter toutes les violations possibles surtout que ces clauses sont généralement exprimées en utilisant des chaînes de caractères très peu structurées. En outre, les solutions proposées se basent sur des entités ad-hoc pour réaliser la surveillance. En effet, elles présentent un manque d'évolutivité quand les paramètres à surveiller sont composés et ne sont pas directement calculables. Enfin, la majorité des approches existantes se contentent de décrire les aspects théoriques de la surveillance sans passage réel et complet à la réalisation. Il est donc nécessaire de penser à une approche complète de surveillance des accords de QdS. L'implantation de cette approche pourrait être gérée par une partie tierce de confiance qui sera solicitée par les clients et les fournisseurs pour surveiller les contrats de QdS qu'ils ont établis afin de protéger leurs droits.

VII. CONCLUSION

Dans ce chapitre, nous avons commencé par présenter la notion d'accords de QdS ainsi que son importance dans les architectures orientées services. Ensuite, nous avons donné un aperçu sur les travaux existants de modélisation de ces accords afin d'automatiser leur gestion. Nous nous sommes ensuite, intéressés aux travaux existants concernant la négociation et la surveillance de ces accords. L'étude de ces travaux nous a permis de constater que la majorité des approches de négociation existantes permettent au fournisseur d'exprimer ses offres pour le client qui se contente d'accepter ou de refuser ces offres. Dans le cas où le client n'est pas expert du domaine, il pourrait ne pas comprendre le contrat proposé par le fournisseur dû au langage technique utilisé par ce dernier dans ses offres. En outre, nous avons remarqué l'absence d'approches complètes et automatiques de surveillance des accords de QdS. Cette approche est essentielle pour le client pour lui garantir un contrôle de confiance par rapport au contrat qu'il a signé avec le fournisseur.

Ces insuffisances dans la phase de négociation et de surveillance d'accords de qualité de service sont dues principalement à l'absence de modèles qui peuvent être utilisés et « compris » par (i) l'être humain pour pouvoir négocier et exprimer ses attentes avec son propre langage d'une part et (ii) par la machine pour pouvoir automatiser et faciliter l'analyse de ces attentes, la génération d'un contrat de QdS en cas de compatibilité et la surveillance de ses différentes clauses d'autre part. Ces modèles doivent contenir la sémantique des attentes des deux parties (fournisseur et client) et la sémantique des accords conclus. Il est donc important d'explorer les techniques existantes pour pouvoir définir et utiliser des modèles sémantiques afin d'exprimer plus finement ces attentes et ces accords [52], [53]. Dans le chapitre suivant, nous détaillons d'avantages les techniques sémantiques utilisées pour réaliser ces modèles.

CHAPITRE II – ETAT DE L'ART : INTEGRATION DE LA SEMANTIQUE DANS LE CYCLE DE VIE DES ACCORDS DE QDS

SOMMAIRE DU CHAPITRE

I. INTRODUCTION

La modélisation et la représentation du monde réel constituent un sujet très riche à explorer. Le besoin de représenter les connaissances d'un domaine spécifique d'une manière à travers laquelle les machines puissent manipuler la sémantique des informations a donné naissance aux ontologies. Les ontologies apparaissent ainsi comme des composants logiciels s'insérant dans les systèmes d'information en leur apportant une dimension sémantique. Elles sont d'une grande utilité pour la représentation des accords de QdS. Dans ce chapitre, nous commençons par définir la notion d'ontologie. Ensuite, nous exposons différentes modélisations des accords de QdS en utilisant les ontologies. Par la suite, nous présentons quelques approches existantes d'alignement d'ontologies qui consistent à découvrir des correspondances possibles entre deux ontologies données. Finalement, nous terminons par une conclusion générale relative à ce chapitre.

II. LES ONTOLOGIES

Nous avons choisi les ontologies parce qu'elles offrent une modélisation expressive et formelle de l'information [56]. En effet, elles permettent de représenter la sémantique qui existe entre les différentes entités du domaine ainsi que les relations qui existent entre elles.

1. Notions d'ontologies et de Web sémantique

Gruber en 1993 proposait la définition la plus citée. Il définit l'ontologie comme étant une spécification explicite d'une conceptualisation [54]. La conceptualisation est le résultat d'une analyse du domaine étudié et l'abstraction du monde de ce domaine. Cette conceptualisation est représentée dans une forme concrète où les concepts, les relations ainsi que les contraintes sont explicitement définis dans un format et un langage formel. Par la suite, elle a été raffinée dans le domaine de l'informatique par Neches et al [55]. Ils définissent l'ontologie comme les termes et les relations de base comportant le vocabulaire d'un domaine aussi bien que les règles pour combiner les termes et les relations afin de définir des extensions du vocabulaire.

Selon *Wikipedia*, une ontologie [56] désigne un ensemble structuré de savoirs dans un domaine particulier de la connaissance ou un ensemble de concepts organisés en graphe dont les relations peuvent être des relations de nature sémantique, des relations de composition et des relations d'héritage. Une ontologie permet donc d'organiser des informations ou des concepts pour construire de la connaissance. Elle permet aussi d'ajouter des règles d'inférence et de faciliter la déduction des informations. En ce sens, elle a naturellement sa place dans le cadre des services Web sémantiques. Les services Web sémantiques [57], [58], [59] se situent à la convergence de deux domaines de recherche importants qui concernent les technologies de l'Internet : le Web sémantique et les services

Web. Le Web sémantique s'intéresse principalement aux informations statiques disponibles sur le Web et les moyens de les décrire de manière intelligible pour les machines. Les services Web, quant à eux, ont pour première préoccupation la garantie d'un moyen d'interopérabilité entre les applications via le Web. L'objectif visé par la notion de services Web sémantiques est de créer un Web sémantique de services dont les propriétés, les capacités, les interfaces et les effets sont décrits de manière non ambiguë et sont exploitables par des machines et ce en utilisant les couches techniques sans pour autant en être conceptuellement dépendants. La sémantique ainsi exprimée permettra l'automatisation des fonctionnalités suivantes qui sont nécessaires pour une collaboration inter-entreprises efficace : processus de description des services et négociation des contrats.

2. Structure générale d'une ontologie

La construction d'une ontologie consiste à identifier les concepts d'un domaine de connaissance [60] ainsi que les relations entre eux afin de représenter la sémantique de façon indépendante des diverses applications dans lesquelles pourra être utilisée l'ontologie. Les ontologies se basent généralement sur les composantes suivantes :

- Les concepts ou les classes

Un concept est un ensemble de ressources partageant les mêmes caractéristiques. Il peut représenter un objet matériel, une notion, une action, un processus de raisonnement ou une idée. Chaque concept est associé à un ensemble d'individus appelés des instances.

Pour assurer la sémantique et donner un sens aux représentations symboliques, les concepts doivent être reliés entre eux. Pour cela, chaque concept est défini par un ensemble de propriétés et de relations qui le lient avec les concepts voisins.

- Les propriétés :

Les propriétés aident à la caractérisation du concept. Certaines propriétés sont essentielles pour bien identifier le concept, d'autres sont facultatives et aident à mieux le caractériser. Il existe deux types de propriétés : propriétés d'objets *ObjectProperty* qui lient entre deux individus d'une même classe ou de deux classes différentes et les propriétés de type de données *DataTypeProperty* qui lient un individu avec une valeur de donnée. Chaque propriété admet un domaine (l'ensemble de ses concepts sources) et une cible *range* (l'ensemble de ces concepts destinations).

- Les relations :

Les relations d'une ontologie désignent les différentes interactions et corrélations entre les concepts. Ces relations englobent les associations suivantes : sous classe de (spécification ou généralisation), partie de (agrégation ou composition), associé à, instance de, est un …Les propriétés d'objets sont aussi considérées comme des relations dans la plupart des travaux de recherche.

- Les individus ou les instances

Les individus peuvent être définis comme des instances ou des exemples des classes. Ils héritent alors toutes les propriétés de cette classe.

3. Objectif général d'une ontologie

Nous pouvons distinguer deux principaux objectifs de l'ontologie selon [61] :

- Le premier est descriptif. Il consiste à former une base de connaissances en modélisant un ensemble de connaissances dans un domaine donné. Le langage le plus adopté pour la modélisation des ontologies est OWL.
- Le deuxième est de raisonnement. Il consiste à faire des inférences et à créer automatiquement de la connaissance à l'aide des règles exprimées dans les ontologies. L'inférence est une opération logique portant sur des propositions reconnues pour vraies pour en tirer une conclusion à partir de règles de base. Le langage SWRL permet de créer et de manipuler les règles d'inférence.

Nous allons utiliser ces deux langages (OWL et SWRL) de représentation des connaissances tout au long de notre travail.

i. Le langage OWL

OWL est un langage standardisé par l'organisme W3C [62] pour la construction des ontologies. Il se base sur de nombreux mécanismes de raisonnement de DAML + OIL [63]. Il peut être considéré comme une extension de Resource Description Framework (RDF) et RDF Schema (RDFS) [64]. Le but est d'ajouter plus de formalisation dans l'expression des informations afin d'assurer leur validité. Il permet de définir des restrictions de cardinalités telles que : «une personne a *exactement* deux parents » et «une personne reçoit l'enseignement d'*au moins* un professeur». Avec OWL, on peut aussi décrire des disjonctions de classes (par exemple mâle et femelle) ou des combinaisons booléennes entre les classes (comme par exemple la classe *personne* est l'union disjointe des classes mâle et femelle). Contrairement à RDF et RDFS, OWL peut aussi exprimer des caractéristiques particulières des propriétés comme *inverse de* (par exemple « mange » est l'inverse de « est mangé par ») et la *transitivité* (la propriété « plus grand que » est transitive).

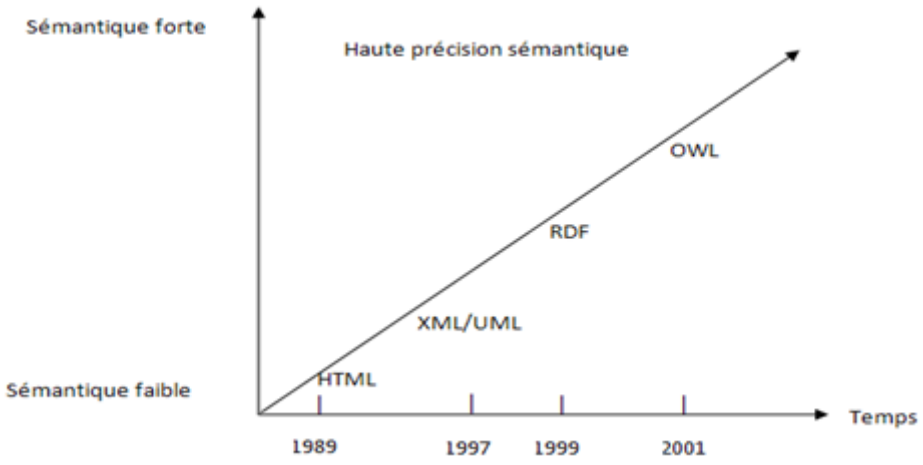

Figure 10 – Evolution du spectre sémantique

La Figure 10 montre l'évolution des langages de description sémantique. Malgré ses fonctionnalités,

OWL n'est pas un langage opérationnel en soi. En effet, il n'offre pas de mécanismes permettant de déduire les conséquences logiques d'une ontologie (c'est-à-dire les faits qui ne sont pas littéralement présents dans l'ontologie mais peuvent être déduits par la sémantique formelle). Pour remédier à cette insuffisance, on a recours au langage SWRL.

ii. Le langage SWRL

SWRL est un langage standardisé par W3C qui permet aux utilisateurs de rédiger des règles exprimées en termes de concepts OWL pour fournir des capacités de raisonnement déductif. SWRL définit plusieurs opérateurs mathématiques et assure des garanties formelles fortes lors de l'exécution de l'inférence [65]. Il est construit sur le même fondement logique de description OWL, il combine alors les sous langages OWL Lite, OWL DL et OWL Full de OWL avec certains sous langages de *RuleML*.

Ce langage enrichit la sémantique d'une ontologie définie en OWL en utilisant des mécanismes de raisonnement sur les connaissances exprimées en OWL. Ces mécanismes sont implantés dans des moteurs d'inférence qui assurent l'exécution des règles SWRL à travers des ponts connus par le nom *bridge* afin d'automatiser le processus de déduction sur les règles.

Un des éléments clés du langage SWRL est sa capacité d'être étendu par des méthodes définies par l'utilisateur qui peuvent être intégrées dans les règles. Ces méthodes sont appelées *built-in*s [66] qui acceptent un ou plusieurs arguments. Un certain nombre de *built-ins* de base sont définis dans la spécification de SWRL. Cet ensemble de base comprend des opérateurs mathématiques et des fonctions de manipulations de chaînes de caractères et de dates.

iii. Les règles SWRL

Les règles SWRL sont élaborées selon le schéma : antécédent (body) →conséquent (head).

La partie corps (swrl : body) spécifie les conditions qui doivent être vérifiées et la partie entête (swrl: head) spécifie les actions à prendre en cas de satisfaction des conditions de la règle.

Voici un exemple d'une règle SWRL qui utilise le *built-in* prédéfini *greaterThan* :

$Person(?p) \wedge hasAge(?p, ?age) \wedge swrlb:greaterThan(?age, 18) \rightarrow Adult(?p)$

Cette règle stipule que si l'âge d'une personne est supérieur à 18 ans, alors elle est adulte. Les points d'interrogation dans une règle SWRL permettent de déclarer des variables.

4. Exemple d'une ontologie

Dans cette section, nous présentons un exemple simple d'une ontologie écrite en langage OWL. Cette ontologie contient un seul concept *Person*.

Ce concept a trois propriétés d'objet nommées : *hasParent*, *hasBrother* et *hasUncle*. La première propriété exprime la relation parent/fils entre deux personnes. La deuxième propriété exprime la relation frère/frère et la troisième modélise le lien entre un neveu et son oncle. Toutes ces propriétés admettent comme domaine *domain* le concept *Person* et comme cible *range* le concept *Person*.

```
<rdf:RDF xml:base="http://www.owl-ontologies.com/Ontology1275896776.owl">
  <owl:Class rdf:ID="Person"/>
  <owl:ObjectProperty rdf:about="http://www.w3.org/2003/11/swrl#argument2"/>
  <owl:ObjectProperty rdf:ID="hasParent">
    <rdfs:range rdf:resource="#Person"/>
    <rdfs:domain rdf:resource="#Person"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hasBrother">
    <rdfs:range rdf:resource="#Person"/>
    <rdfs:domain rdf:resource="#Person"/>
  </owl:ObjectProperty>
  <owl:ObjectProperty rdf:ID="hasUncle">
    <rdfs:domain rdf:resource="#Person"/>
    <rdfs:range rdf:resource="#Person"/>
  </owl:ObjectProperty>
</rdf:RDF>
```

Figure 11 – Exemple d'un fichier OWL

Le fichier OWL implémentant cette ontologie est présenté dans la Figure 11.

Cette ontologie contient la règle SWRL suivante :

hasParent (?Child, ?Parent) ∧ hasBrother(?Parent, ?Person) → hasUncle(?Child, ?Person)

Cette règle indique que si *Parent* est le père de *Child* et *Person* est le frère de *Parent* alors *Person* est l'oncle de *Child*. Le code SWRL correspondant à cette règle est présenté dans la Figure 12.

```
            <swrl:AtomList>
              <rdf:first>
                <swrl:IndividualPropertyAtom>
                  <swrl:argument1 rdf:resource="#parent"/>
                  <swrl:propertyPredicate rdf:resource="#hasBrother"/>
                  <swrl:argument2 rdf:resource="#person"/>
                </swrl:IndividualPropertyAtom>
              </rdf:first>
              <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
            </swrl:AtomList>
          </rdf:rest>
        </swrl:AtomList>
      </swrl:body>
    </swrl:Imp>
</rdf:RDF>
```

```
<swrl:Variable rdf:ID="person"/>
<swrl:Imp rdf:ID="Rule-1">
  <swrl:head>
    <swrl:AtomList>
      <rdf:first>
        <swrl:IndividualPropertyAtom>
          <swrl:argument2 rdf:resource="#person"/>
          <swrl:argument1>
            <swrl:Variable rdf:ID="child"/>
          </swrl:argument1>
          <swrl:propertyPredicate rdf:resource="#hasUncle"/>
        </swrl:IndividualPropertyAtom>
      </rdf:first>
      <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
    </swrl:AtomList>
  </swrl:head>
  <swrl:body>
    <swrl:AtomList>
      <rdf:first>
        <swrl:IndividualPropertyAtom>
          <swrl:argument2>
            <swrl:Variable rdf:ID="parent"/>
          </swrl:argument2>
          <swrl:argument1 rdf:resource="#child"/>
          <swrl:propertyPredicate rdf:resource="#hasParent"/>
        </swrl:IndividualPropertyAtom>
      </rdf:first>
      <rdf:rest>
```

Figure 12 – Exemple d'une règle SWRL

Plusieurs travaux se sont intéressés à la modélisation des accords de QdS en utilisant les ontologies. Nous donnons un aperçu sur les contributions principales dans ce contexte.

III. MODELISATIONS DES ACCORDS DE QDS EN UTILISANT LES ONTOLOGIES

Nous avons exploré les modèles principaux existants relatifs aux *SLAs*. Nous avons constaté un grand intérêt à la modélisation de la qualité de service, qui est un élément principal dans la spécification des contrats. La majorité de ces modèles se basent sur les ontologies pour la description de leur qualité de service. Dans la suite de cette section, nous présentons les modèles principaux existants de spécification de la qualité de service.

1.1 OWL-QoS

OWL-QoS [67] est un modèle de description des QdS. Ce modèle est caractérisé par sa description formelle de la diffusion et de la consommation de la qualité de service. L'ontologie OWL-QoS peut être représentée en trois couches de la QdS : une couche du profil de QdS (*QoS Profile Layer*), une

couche de définition de propriétés de QdS (*QoS Property Definition Layer*) et une couche des métriques de QdS (*QoS Metrics Layer*). La première couche spécifie les profils des clients et des fournisseurs de services. La deuxième couche spécifie les contraintes sur les propriétés de QdS. La troisième couche contient la définition des métriques et des mesures. *OWL-QoS* présente l'avantage de la réutilisation de l'ontologie standard de la description de services OWL-S [68]. Cependant, Il a quelques insuffisances. Par exemple, les métriques sont spécifiées sans préciser à quoi elles servent. Il ne bénéficie pas aussi des relations d'inférence de l'ontologie puisqu'il utilise des chaînes de caractères non structurées pour la spécification de contraintes de QdS (par exemple : "temps de réponse <100ms").

1.2 QoSOnt

QoSOnt [69] est un autre modèle de QdS. Il est défini par le langage OWL [70]. Il peut être divisé en trois couches (voir Figure 13) : la couche de QdS de base (*Low level Concepts*), la couche des attributs de qualités (*Attribute layer*) et la couche spécifique au domaine (*Metric layer*). La première couche contient des concepts génériques relatifs à la QdS. La deuxième couche contient des ontologies définissant des attributs particuliers de QdS comme la performance par exemple. La troisième couche définit les métriques nécessaires pour le calcul des attributs de la deuxième couche.

QoSOnt est plus riche que l'ontologie *OWL-QoS* au niveau de la définition des métriques et de la correspondance des exigences de qualité de service avec les métriques. Il présente l'avantage d'être interopérable avec d'autres ontologies existantes tel qu'*OWL-S* [71]. Cependant, les types de données utilisés pour les métriques sont définis dans un langage XML spécifique. En conséquence, leur approche se limite à la description de leur qualité de service et n'utilise pas des techniques d'interprétation et d'inférence sémantiques pour faciliter la déduction des informations.

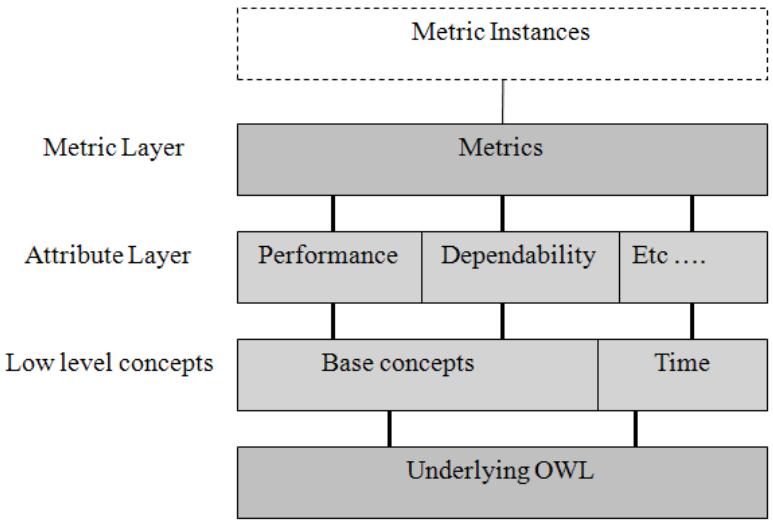

Figure 13 – La structure de *QosOnt* [69]

1.3 SL-ONTOLOGY

SL-Ontology [72] est une autre tentative de modélisation de QdS. La structure de *SL-Ontology* peut être représentée sous une forme pyramidale comme illustrée dans la Figure 14. Dans la partie haute, on trouve trois concepts de *SL-Ontology*. Le concept *Service-Level-Offer* représente les offres du fournisseur de service. Le concept *Service-Level-Request* représente les demandes du client et finalement le concept *Service-Level-Agreement* spécifie les accords des deux parties.

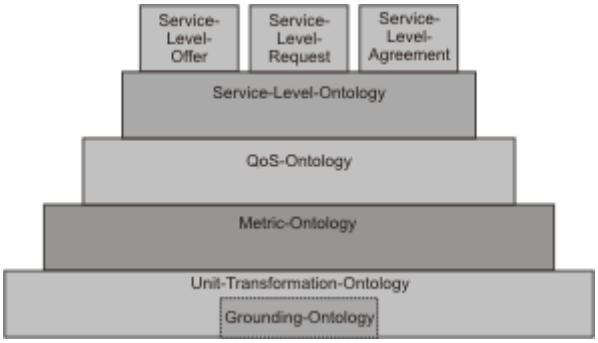

Figure 14 – La structure de SL-Ontology [72]

Au dessous de ces trois concepts, on trouve l'ontologie de QdS nommée *QoS-Ontology*, l'ontologie des métriques nommée *Metric-Ontology* et l'ontologie de transformation des unités nommée *Unit-Transformation-Ontology*. L'ontologie de QdS sert à décrire les paramètres de qualité de service. L'ontologie des métriques sert à décrire les métriques qui permettent de mesurer les paramètres de qualité de service. L'ontologie de transformation des unités sert à résoudre les disparités de langages entre les clients et les fournisseurs. Cette résolution n'est spécifiée qu'au niveau des unités dans ce modèle. Aussi, ce modèle est commun pour les offres du fournisseur ainsi que les demandes des clients ce qui limite l'expressivité du client.

1.4 WS-QOS

WS-QoS [73] est un *framework* qui utilise un modèle de QdS à base d'ontologies pour la sélection dynamique de services Web selon les exigences de performances d'exécution et du réseau. L'ontologie WS-QoS est composée de trois éléments principaux (voir Figure 15) : *QoSInfo*, *WSQoSOntology* et *QoSOfferDefinition*. L'élément *QoSInfo* contient des informations sur les performances du serveur et des protocoles de QdS. L'élément *WSQoSOntology* comprend les définitions des paramètres de QdS et les références des protocoles. Le *QoSOfferDefinition* peut contenir un ou plusieurs éléments *QoSInfo*.

Ce modèle est caractérisé par l'utilisation de métriques spécifiques qui doivent être connues à l'avance par tous les services. Il utilise aussi un langage XML spécifique pour la description des métriques d'où la perte des possibilités de raisonnement et des inférences sémantiques offertes par OWL.

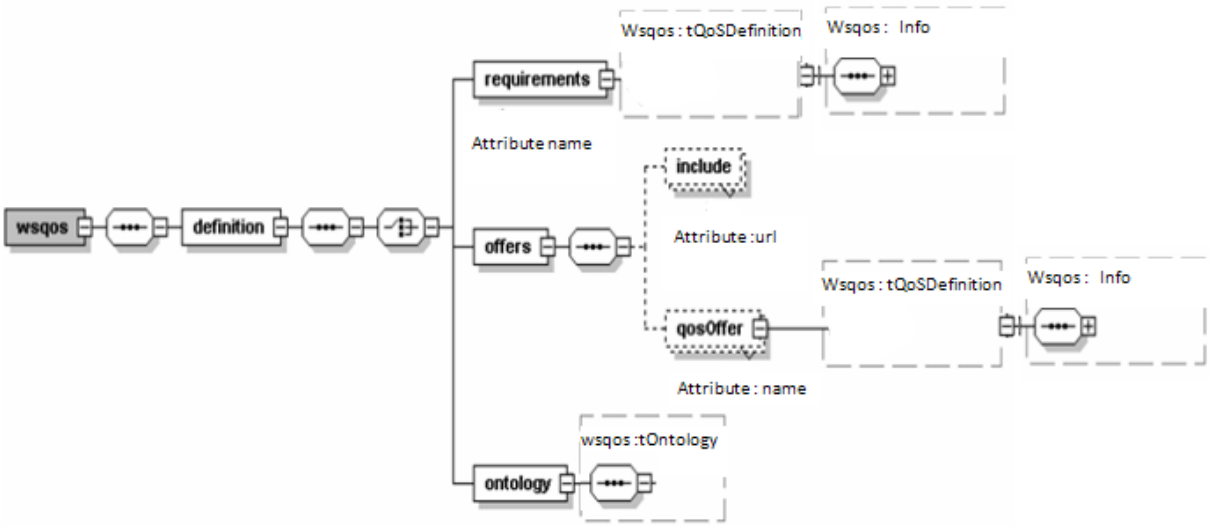

Figure 15 – La structure des éléments de WS-QoS [73]

1.5 FIPA QoS

FIPA QoS [*74*] est un autre modèle de représentation de QdS sous la forme d'une ontologie. Cette ontologie est conçue pour la communauté des agents pour représenter et échanger des informations portant sur la qualité de service. Elle fournit un vocabulaire de QdS et supporte deux méthodes pour récupérer les informations : à travers une simple requête ou bien par abonnement. Par exemple, un agent peut interroger les valeurs actuelles de qualité de service d'un autre agent en utilisant le protocole d'interaction *FIPA-query* [75] ou bien il peut s'abonner aux notifications en cas de mise à jour au niveau de la qualité de service en utilisant le protocole *Fipa-subscribe interaction protocol* [76]. Ce modèle est complet mais il reste trop spécifique aux couches basses du modèle OSI [77].

1.6 MOQ

MOQ (*Mid-Level Ontologies for Quality*) [78] est une autre tentative de modélisation de QdS qui définit des exigences de QdS composites. Par exemple, une exigence de QdS peut avoir des sous exigences (sub-requirements). Une exigence est considérée primitive si elle n'a pas des sous exigences. Dans ce cas, elle est directement mesurable. Si la mesure est dans un interval acceptable, alors cette exigence est considérée satisfaite. Ainsi, d'une façon plus générale, une exigence est satisfaite si toutes ses exigences sont satisfaites. Cependant, Kim HM et al. [78] n'apportent aucune précision sur la forme d'expression des exigences. Elles sont considérées comme des instances abstraites d'une classe *requirement*. En outre, la relation de composition des exigences n'est pas explicitée formellement et se limite à dire qu'une exigence Q peut avoir un ensemble de sous exigences Q_i. Ce modèle se base sur quatre ontologies : *ontologie de besoins*, *ontologie de mesure*,

ontologie de traçabilité et ontologie *de gestion de la qualité* sans préciser les relations qui existent entre elles ce qui peut engendrer des incohérences entre leurs contenues.

1.7 BILAN SUR LA MODELISATION DES ACCORDS DE QDS

Tous les modèles que nous venons de citer présentent des avantages et des lacunes. Certaines contributions (comme *WS-QoS*) utilisent des formats XML spécifiques pour l'implantation complète ou partielle de leurs modèles. Ceci restreint considérablement l'intérêt de l'utilisation d'une ontologie qui offre des possibilités d'inférence et d'interprétation sémantique en se basant sur le langage OWL. En outre, certains modèles sont soit spécifiques à un domaine particulier (comme le cas de *FIPA-QoS* qui est spécifique aux couches basses du modèle OSI) où présentent des incomplétudes diverses (comme l'absence de spécifications de contraintes logiques composées sur les QdS pour le modèle MOQ). Enfin, tous ces travaux se focalisent sur la modélisation de qualité de service sans détailler les obligations et les accords entre les acteurs impliqués. Cette dernière constatation nous a encouragés à élaborer un modèle d'accords de qualité de service en s'inspirant des modèles que nous venons d'explorer pour faciliter la gestion des *SLAs*.

Outre la modélisation des accords de QdS, les ontologies peuvent être intéressantes pour la représentation des besoins du client et des offres du fournisseur. D'une part, ceci permettrait de décrire ces besoins et ces offres avec des structures et des termes différents selon les connaissances propres des clients et des fournisseurs. D'autre part, l'utilisation des ontologies permet de faciliter l'analyse automatique de la compatibilité sémantique entre les besoins du client et les offres du fournisseur. Pour pouvoir représenter ces besoins et ces offres, il est intéressant de capturer les intentions [79] des clients et des fournisseurs. Dans la section suivante, nous présentons le principe de la notion d'intention.

IV. NOTION D'INTENTION

1. Définition

Le concept d'intention est omniprésent dans toute action humaine et est particulièrement important dans la communication (orale ou écrite) [80]. Plusieurs travaux essayent d'expliquer les relations entre une action entreprise par un être humain et l'état mental qui guide cette action.

2. Formalisation d'une intention

Les résultats principaux dans cette recherche nécessitent l'explication de la notion de contexte. Les études sur la modélisation de l'intention dérivent des théories causales de l'action [80]. Décrire une intention, c'est trouver une explication rationnelle à l'action qui a été causée par cette intention. Cette explication relève plus de la pragmatique situationnelle, c'est à- dire qu'elle dépend du contexte dans

lequel l'action peut se dérouler. Pour cette raison, il est évident que la modélisation des intentions en tant qu'états mentaux ne peut se faire que par rapport aux contextes dans lesquels les actions ciblées par ces intentions peuvent se dérouler. Pour identifier les constituants de l'intention en essayant de répondre aux questions de type : quoi, pourquoi, comment et pour quelle raison. Ces questions doivent permettre de détecter les actes décrivant les intentions.

Dans le but de comprendre la notion d'intention dans les documents, [81] propose de formaliser cette notion comme suit :

I (A, B*, M*, R*) où

- I représente l'intention sous jacente à l'action A ;
- A représente l'acte qui exprime l'action de l'intention ;
- B exprime ce que veut faire l'auteur en faisant l'action (But) ;
- M exprime comment l'action est accomplie ;
- R explique pourquoi et pour quelles raisons l'auteur fait cette action ;

* indique que le nombre des actes composant l'intention peut être multiple (0 à N).

La Figure 16 est un schéma graphique de l'intention, elle montre le triplet « But, moyen et raison » d'une action intentionnelle. Ce schéma décrit l'action A, le but de cette action B, un acte qui lie l'acte de réaliser une action pour trouver un but fixé avec le moyen utilisé M, et un acte qui exprime la justification ou la raison R de réaliser cette action.

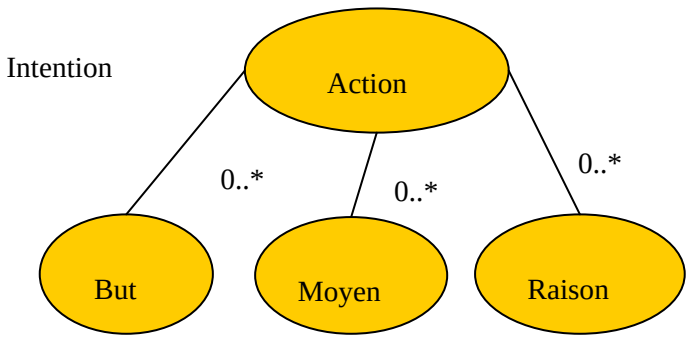

Figure 16 – Représentation schématique d'une intention

Une intention peut avoir :

Une seule action ;

Plusieurs Buts, Moyens et Raisons ;

Ni But, ni Moyens ni Raisons ;

De fait, dans le cas où il n'y a pas de but, l'action est égale au but. Dans une autre situation les composants du triplet d'une action peuvent devenir une action d'une nouvelle intention.

3. Exemple de représentation d'une intention

La Figure 17, montre une représentation schématique d'un exemple d'une intention. Cette intention est déduite de la phrase suivante : "Un client veut télécharger périodiquement des films récents après avoir établi un contrat avec une librairie numérique de vidéos dans le but de les visualiser hors ligne afin de se distraire ".

Dans cet exemple :

L'action A est "télécharger des films",

Le but B est "Visualiser des films",

Le moyen M est "Etablir un contrat",

La raison R est "Se distraire".

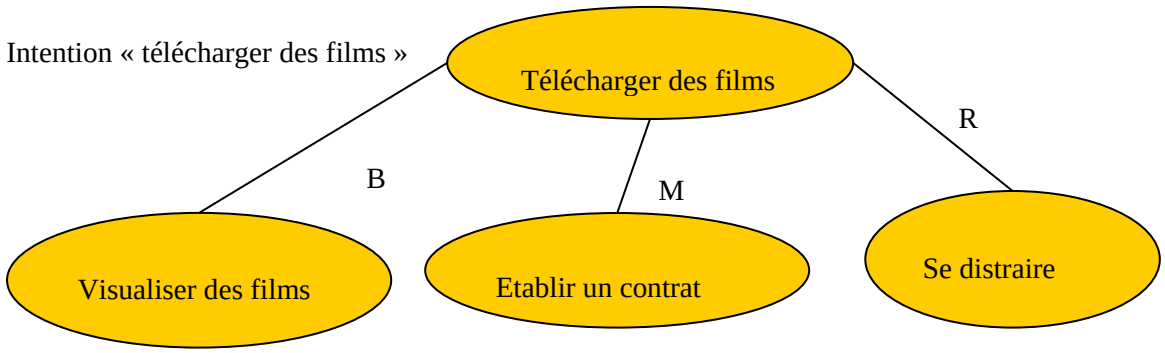

Figure 17 – Exemple de représentation des intentions

Partant du fait que les besoins du client et les offres du fournisseur peuvent être représentés sous la forme d'intentions, il est nécessaire de disposer d'outils permettant de faire le lien entre les connaissances exprimées dans chacune de ses sources d'informations. Nous présentons quelques approches qui permettent de fusionner et réconcilier, d'un point de vue sémantique, les opinions de plusieurs parties [82]. Dans la section suivante, nous exposons quelques techniques d'alignement sémantiques et leurs différentes caractéristiques.

V. ALIGNEMENT DES ONTOLOGIES

Bien que les ontologies soient un élément essentiel des systèmes d'information car elles permettent à la fois de décrire le contenu des sources à intégrer et d'expliciter le vocabulaire utilisable dans les requêtes des utilisateurs, la tâche d'alignement est particulièrement importante dans ces systèmes d'information puisqu'elle autorise la prise en compte conjointe de ressources décrites par des ontologies différentes.

1. Alignement : définition et principe

L'alignement d'ontologies est un processus de découverte des correspondances entre deux ontologies sources. Il est généralement décrit comme une application de l'opérateur *MATCH* [83], dont l'entrée est constituée d'un ensemble d'ontologies et la sortie, formée des correspondances entre ces ontologies [84] (voir Figure 18). L'alignement est employé quand des sources doivent être rendues conformes et cohérentes entre elles mais elles sont toujours gardées séparément [86]. Cela implique de mettre deux (ou plus) ontologies en accord mutuel et de les rendre conformes et cohérentes [87], [88]. Plusieurs techniques d'alignements ont été créées pour définir collectivement les relations entre les ontologies originales.

Ainsi, l'alignement des ontologies permet d'établir des liens sémantiques entre les concepts et les relations inter-ontologies, comme l'illustre la définition de Namyoun [85] : L'alignement d'ontologies est un processus de création de liens de deux ontologies. Cet alignement est intéressant quand les sources produisent un résultat consistant en les associant tout en les laissant séparées. Dans la majorité des cas, l'alignement produit un résultat significatif quand les sources sont complémentaires.

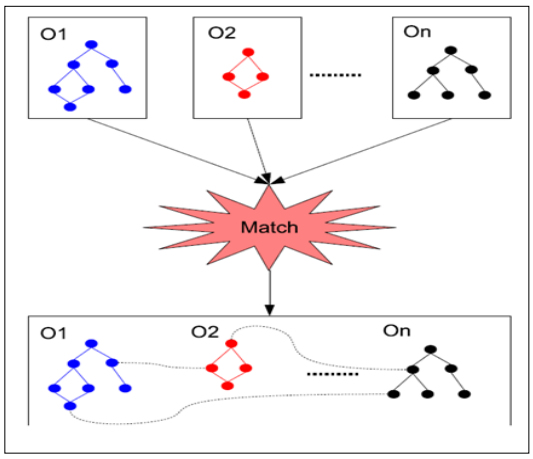

Figure 18 – L'opérateur Match

2. Techniques d'alignement

On trouve dans la littérature plusieurs algorithmes qui implantent l'opérateur Match qu'on vient d'introduire dans la section 1 de ce chapitre. Ces algorithmes sont regroupés principalement en quatre classes :

i. Basée-linguistique : les termes utilisés dans les ontologies ont les rôles les plus importants. Cet alignement est basé sur la similitude (synonymie, hyponymie,…) des noms de concepts et des relations.

ii. Basée-heuristique : Un ensemble de mesures similaires heuristiques sont adoptés pour l'analyse de la structure des concepts en termes de propriétés et de relations sémantiques. Par exemple, des concepts sont considérés similaires s'ils ont des propriétés similaires ou des relations similaires avec d'autres concepts.

iii. Basée-probabilité : les concepts similaires sont calculés avec des raisonnements probabilistes ou avec des techniques d'apprentissage. L'alignement est réalisé si deux concepts ont une grande probabilité de représenter le même objet réel ou ont les mêmes instances.

iv. Basée-raisonnement : Cette approche est basée sur des techniques de déduction selon des règles logiques prédéfinies. L'alignement final propose des relations sémantiques découvertes entre les concepts des deux ontologies.

3. Approches existantes d'alignements

Les techniques d'alignement jouent un rôle crucial dans la construction d'un lien sémantique entre les ontologies d'un même domaine. Quelques approches d'alignement [89], [90], [91], [92] considèrent que l'utilisation d'une connaissance sur le domaine est une manière d'assurer la correspondance sémantique entre la dissimilarité syntaxique des ontologies. L'obtention de la bonne connaissance sur le domaine est primordiale. D'autres approches n'exploitent pas une connaissance sur le domaine et ne réalisent pas un modèle sémantique formel pour l'alignement des structures produites.

Parmi les approches d'alignement d'ontologies, on trouve *Anchor-PROMPT* [93], *S-Match* [94], *MAFRA* [95], *GLUE* [96], *QOM* [97] et *ASCO* [98]. Le Tableau 1 [99] montre un aperçu des approches existantes et les techniques d'alignement adoptées [100], [101].

Approches	Technique d'alignement	Organisation
Anchor-PROMPT (2000)	Basé-linguistique	Stanford University (USA)
S-Match (2003)	Basé-linguistique Basé-raisonnement	U.trento (Italie)
MAFRA (2002)	Basé-linguistique Basé-heuristique	U.Karlsruhe (Allemagne)
GLUE (2002)	Basé-probabilité	U. Washington (USA)
QOM (2005)	Basé-heuristique	University of Karlsruhe (Allemagne)
ASCO (2004)	Basé-heuristique	INRIA Sophia-Antipolis (France)

Tableau 1 – Exemples d'outils d'alignement (OWL) [100]

3.1 ANCHOR-PROMPT

Noy et Musen [93] ont développé un algorithme nommé *Anchor-PROMPT* afin de découvrir automatiquement les termes sémantiquement similaires. *Anchor-PROMPT* considére une ontologie comme un graphe, les nœuds de ce graphe représentant les concepts et les arcs représentant les propriétés. L'algorithme utilise deux paires de termes relatifs comme entrée. Il analyse les chemins dans le sous-graphe délimité par des ancres et détermine quelles sont les concepts qui apparaissent fréquemment dans des positions similaires sur des chemins similaires. L'algorithme cherche alors des termes en parcourant des chemins qui pourraient être similaires aux termes d'autres chemins. Les termes qui sont fortement semblables sont présentés à l'utilisateur pour améliorer l'ensemble des suggestions possibles. Cependant, cette approche d'alignement ne tient pas compte de la compatibilité des objectifs de QdS.

3.2 S-MATCH

S-Match est une approche pour l'alignement des hiérarchies de classifications. Les auteurs mettent en œuvre un opérateur de correspondance qui admet en entrée deux structures sous forme de graphes (par exemple, schémas de base de données ou ontologies) et produit un alignement entre les éléments qui sont en correspondance sémantique. L'alignement sémantique calcule une relation, de nature ensembliste, entre les nœuds en tenant compte de la signification de chaque nœud. La sémantique d'un nœud est déterminée par son label et celle de tous les nœuds qui sont plus haut dans la hiérarchie. Les relations possibles retournées par l'algorithme d'alignement sémantique sont l'égalité, l'intersection, la disparité, la généralité ou la spécificité. Cependant, le processus d'alignement utilisé se base uniquement sur les liens hiérarchiques de subsumption pour détecter ces relations. Dans une ontologie, on trouve d'autres relations sémantiques entre les concepts qui sont utiles pour le processus d'alignement (voir *ObjectProperty* dans la section II.2 de ce présent chapitre).

3.3 MAFRA

MAFRA (*MApping FRAmework*) est une plateforme pour la correspondance d'ontologies. Son approche repose sur la notion fondamentale de Passerelle qui permet de formaliser les correspondances en établissant des liens entre les entités d'une ontologie source et celles d'une ontologie cible. L'architecture conceptuelle de *MAFRA* distingue la dimension horizontale et la dimension verticale. La dimension horizontale est caractérisée par cinq modules. Le module de *normalisation* où les ontologies sont représentées de manière uniforme. Le module de *similarités* établit les ressemblances entre l'ontologie source et l'ontologie cible. Le module du *bridging* sémantique, basé sur les similarités, vise à établir les correspondances entre les entités appartenant aux ontologies sources et cibles en utilisant des heuristiques. Ce module spécifie des passerelles entre entités pour lesquelles chaque instance représentée dans l'ontologie source est transformée en

l'instance la plus similaire dans l'ontologie cible. Le module *execution* transforme les instances à partir de l'ontologie source vers l'ontologie cible en évaluant les passerelles sémantiques. Enfin, le module *Post-Processing* récupère le résultat d'exécution.

Concernant la dimension verticale, elle est caractérisée par quatre modules. Le module *d'Evolution* sert à encapsuler les passerelles sémantiques générées dans le module *bridging sémantique*. Le module *Connaissance du domaine et contraintes* est utilisé pour améliorer les passerelles sémantiques. Le module de *Construction d'un consensus coopératif* consiste à établir un consensus sur les passerelles sémantiques entre deux communautés participant au processus de correspondances. Enfin, le module *Interface graphique pour l'utilisateur (GUI)* offre une interface graphique pour utiliser la plateforme.

Il est à noter que cette plateforme de correspondances d'ontologies se base sur des interventions manuelles d'un expert pour choisir les passerelles sémantiques valides parmi celles qui sont détectées par la plateforme. En plus, l'approche adoptée ne tient pas compte de la compatibilité des correspondances de QdS.

3.4 GLUE

GLUE est un système qui emploie une approche de type *machine learning* pour créer des correspondances semi-automatiques entre ontologies. Il se focalise sur la détermination des correspondances de type 1-à-1. La similarité de deux concepts A et E dans deux taxinomies O1 et O2 utilise l'ensemble des instances des deux concepts qui convergent. Pour déterminer si une instance du concept E est également une instance du concept A, un classifieur est d'abord construit en utilisant les instances de A comme ensemble d'apprentissage. Ce classifieur est ensuite utilisé à son tour pour traiter les instances de E. Le classifieur décide alors pour chaque instance de E, s'il est également une instance de A ou non. Avec ces classifications, quatre probabilités sont calculées : P(A,E), P($\bar{A}$,E), P(A, $\bar{E}$) et P($\bar{A}$, $\bar{E}$). La probabilité P (A, $\bar{E}$) par exemple, s'interprète par l'appartenance de l'instance du domaine à A et la non appartenance à E. Cependant, cette approche ne gére pas la compatibilité des correspondances de qualité de services.

3.5 QOM

QOM (*Quick Ontology Mapping*) est une approche qui a été conçue pour fournir un outil d'alignement pour la création à la volé des correspondances entre les ontologies. Afin d'accélérer l'identification des similarités entre deux ontologies, *QOM* ne compare pas celles de la première ontologie avec toutes les entités de la seconde ontologie, mais emploie des heuristiques (par exemple, labels semblables) pour abaisser le nombre de correspondances candidats. Le calcul réel de similarité est effectué en utilisant une large gamme de fonctions de similarité, telles que la similarité des chaînes de caractères. Cette approche, est comme les précédentes, ne détecte pas les correspondances de QdS entre deux ontologies.

3.6 ASCO

ASCO est un algorithme qui permet de comparer deux ontologies. Il trouve des correspondances en suivant un processus à deux phases :

• La phase linguistique dans laquelle la valeur de similarité entre deux entités, telles que des concepts ou des relations provenant de deux ontologies différentes, est calculée à partir de différentes informations disponibles telles que leurs noms, leurs labels et leurs descriptions. Le calcul de la valeur de similarité linguistique est basé sur les relations de synonymie ou hyperonymie entre termes de *WordNet*.

• La phase structurelle exploite les informations dans les structures des ontologies. Elle utilise des heuristiques et les connaissances du domaine pour calculer les valeurs de similarité structurelle entre entités des deux ontologies.

Les valeurs de similarité dans les deux phases sont combinées pour obtenir les valeurs de similarité finales entre les entités. Les alignements sont déduits de ces valeurs. Cependant, cet alignement n'est pas basé sur le langage OWL ce qui limite l'intérêt de cette approche.

4. Bilan sur l'alignement des ontologies

La plupart de ces méthodes d'alignement exploitent des ontologies décrites soit avec le langage RDF (*ANCHOR-PROMPT, QOM* et *ASCO*) soit avec le langage *OWL-Lite* (*OLA*). Étant donné que le langage OWL est un standard pour les ontologies, toute méthode d'alignement n'exploitant pas ce format présente un inconvénient.

Par ailleurs, ces approches n'exploitent pas toutes les relations sémantiques qui existent entre les concepts d'une ontologie. Ces relations se limitent à des liens hiérarchiques de subsumption uniquement. En plus, ces approches ne prennent pas en considération la compatibilité des contraintes de qualité de service lors du processus d'alignement. En effet, elles se basent uniquement sur une vérification d'équivalence des termes utilisés dans deux ontologies et ne traitent pas la vérification des contraintes de QdS définies par le client.

VI. CONCLUSION

Dans ce chapitre, nous avons exploré les modèles principaux existants relatifs aux *SLAs*. Nous avons remarqué que les meilleurs modèles se basent sur le formalisme des ontologies présentant une richesse sémantique pouvant améliorer et automatiser plusieurs tâches dans le cycle de vie d'un SLA. Cependant, malgré cette avancée, ces modèles se focalisent sur la qualité de service sans détailler les obligations, les pénalités et les accords entre les acteurs impliqués.

Pour automatiser la négociation de ces accords de QdS tout en laissant la liberté aux clients et aux fournisseurs d'exprimer leurs demandes et leurs offres avec leurs propres langages et connaissances,

nous avons exploré la notion d'intention de ces acteurs. En utilisant cette notion, le processus de négociation de QdS devient donc une mise en correspondance des intentions des différents acteurs de la négociation. Cette mise en correspondances est connue sous le nom d'alignement sémantique. Après avoir exploré les approches d'alignements existantes, nous avons analysé certaines limites. En effet, ces approches s'appuient sur des similarités syntaxiques entre les modèles. Cette similarité n'est pas suffisante puisque le sens des termes utilisés est toujours plus important que la syntaxe d'une part, et puisque ces correspondances ne permettent pas de comparer les contraintes de QdS d'autre part. Nous rappelons que ces contraintes forment la base d'un accord de qualité de service.

Pour remédir aux insuffisances que nous avons décelées après notre analyse de l'existant, nous nous sommes fixés les objectifs suivants : (1) faciliter la compréhension entre le client et le fournisseur en intégrant la notion d'intention dans le processus de négociation de QdS, (2) aider les experts commerciaux des fournisseurs à analyser les demandes des clients d'une façon automatique (3) générer automatiquement un contrat de QdS en cas de compatibilité entre les demandes et les offres (4) assurer la surveillance automatique des contrats générés en appliquant directement les pénalités nécessaires en cas de violation.

Suite à ces objectifs, nous présentons dans le chapitre suivant nos contributions dans le cycle de vie des accords de qualité de service par le biais d'une approche d'alignement nommée ***ODACE SLA*** (***O****ntology* ***D****riven* ***A****pproach for automati****C*** ***E****stablishment of* ***S****ervice* ***L****evel* ***A****greements*) et une approche de surveillance nommée ***SLAOnt-Monitoring*** **(*SLA Ont****ology*** ***Monitoring***).

CHAPITRE III – CONTRIBUTIONS : APPROCHE SEMANTIQUE POUR LA NEGOCIATION ET LA SURVEILLANCE DES ACCORDS DE QDS

SOMMAIRE DU CHAPITRE

I. INTRODUCTION

Compte tenu de la dynamicité des systèmes dans les architectures orientées services (services de nature hétérogène, relations fournisseurs-clients à la demande, externalisation de systèmes, modifications transparentes des implantations, etc.), il est nécessaire de spécifier les relations qui existent entre les fournisseurs et les clients. Ces relations sont traduites en une négociation entre le client et le fournisseur. La négociation est une étape indispensable pour aboutir à un contrat cohérent et stable. Les approches de négociation existantes consistent à choisir un sous-ensemble de clauses et de valeurs parmi des choix prédéfinis par le fournisseur. Généralement, ces choix se basent sur des termes techniques que le client pourrait ne pas comprendre s'il n'est pas un expert du domaine. Ceci rend l'élaboration des contrats déséquilibrée et asymétrique vu que le fournisseur impose ses choix d'une part. D'autre part, la surveillance des contrats reste encore une tâche difficile à réaliser vu la complexité des mesures et le suivi des différents paramètres de qualité de service (*QdS*) spécifiés dans ces contrats. Ainsi, un défi majeur des *SLAs* dans les architectures orientées services réside en la capacité d'automatiser le processus complet de création des contrats, dans des formats traitables en machine, jusqu'à leur surveillance et leur gestion au cours d'exécution des services. Dans la section suivante, nous commençons par présenter les modèles sémantiques que nous avons élaborés pour faciliter le processus d'automatisation du cycle de vie des *SLA*.

II. PRE-REQUIS DE NOTRE APPROCHE DE NEGOCIATION D'ACCORDS DE QUALITE DE SERVICE

Dans cette section, nous nous focalisons sur l'élaboration des modèles sémantiques nécessaires pour faciliter l'établissement des contrats de QdS et la surveillance des obligations spécifiées dans ces contrats. Nous décrivons ces modèles par des ontologies. Chaque modèle $\boldsymbol{O}$ est caractérisé par un ensemble de concepts $\boldsymbol{C}$ et de relations $\boldsymbol{R}$ tels que $\boldsymbol{O = \{C, R\}}$ avec :

– $\boldsymbol{C = \{C_i\}}$: est l'ensemble de tous les concepts $\boldsymbol{C_i}$ de l'ontologie $\boldsymbol{O}$ et $\boldsymbol{C_i = \{C_{i,k}\}}$: est l'ensemble de toutes les instances de $\boldsymbol{C_i}$

– $\boldsymbol{R = \{R_n\}}$: est l'ensemble de toutes les relations $\boldsymbol{R_n = (C_i, C_j)}$ entre les concepts de l'ontologie $\boldsymbol{O}$

et $\boldsymbol{R_{n,k} = (C_{i,k}, C_{j,m})}$: est l'ensemble des instances des relations $\boldsymbol{R_n}$

Dans ce qui suit, nous présentons notre modélisation des intentions du client et des offres du fournisseur. Nous avons choisi le diagramme de classes UML pour la modélisation de nos ontologies vu son pouvoir d'expression générique qui permet de modéliser les concepts des ontologies en tant que des classes et les relations entre ces concepts en tant que associations de classes.

1. Proposition d'une modélisation des intentions du client : « ClientOnto »

Dans cette section, nous présentons l'ontologie *ClientOnto* que nous avons proposée pour modéliser les intentions des clients. Une intention correspond à un état d'esprit de tout acteur qui effectue une action. Pour utiliser la notion d'intention, il est indispensable d'utiliser un langage formel. Nous avons adapté et enrichi le modèle intentionnel de [102] (H. Kanso et al. 2007) en ajoutant des éléments spécifiques relatifs à la négociation d'accords de qualité de service. En effet, ce travail se contente de présenter une structure abstraite et générale de la notion d'intention sans exprimer comment définir les contraintes de QdS. La Figure 19 présente le diagramme de classe illustrant notre modèle intentionnel étendu. Ce modèle exprime les exigences du client. Dans son intention, ce dernier souhaite effectuer une action que nous représentons par le concept *Action* dans notre modèle. Cette action peut être sous la forme d'un verbe qui décrit l'intention du client (par exemple : *download*). Pour chaque action, le client peut définir son sujet *Subject* (par exemple : *Film*) et ses intentions sous la forme de contraintes *Constraint*. Par exemple, le client peut formuler une contrainte sur le temps de téléchargement des films qui l'intéressent (*Download Time* < 10mn). Dans ce cas, la propriété est le temps de téléchargement *Download Time*, le seuil est la valeur maximale fixée par le client (10mn) et l'opérateur est «inférieur». Chaque seuil *Threshold* doit avoir une valeur *Value* qui désigne la valeur de la propriété. Le concept valeur peut avoir une instance, un type et une unité de mesure *Unit* (par exemple : minutes). Les propriétés indiquées par le client peuvent être de type qualité *Quality*, prix *Price* ou *SubjectAttribute* (les propriétés du sujet de l'action).

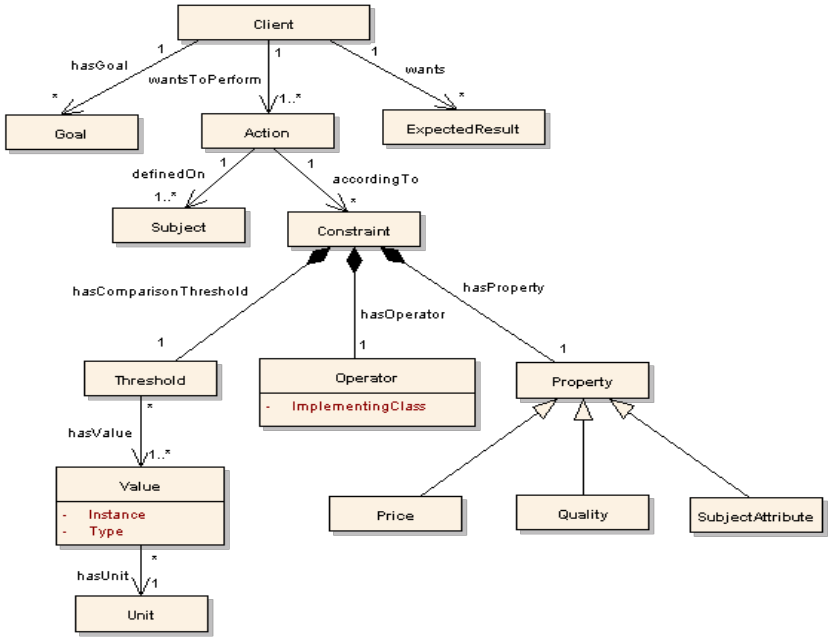

Figure 19 – Structure de l'ontologie *ClientOnto*

Le client peut aussi définir les résultats qu'il attend de la part du fournisseur *ExtendedResult* (par exemple : établir un contrat) et le but *Goal* qu'il l'a amené à exprimer ses intentions (par exemple : divertissement). Notre modèle *ClientOnto* se distingue par rapport aux modèles des demandes des clients par la simplicité de ses termes pour exprimer les besoins du client. Les termes de ce modèle sont indépendants des termes du modèle du fournisseur ce qui donne une opportunité au client de s'exprimer librement avec ses propres connaissances. Dans la section suivante, nous présentons le modèle des offres du fournisseur.

2. Proposition d'une modélisation des offres des fournisseurs : « ProviderOnto »

WSOL *(Web Services Offerings Language*) [19] (V. Tosic et al. 2004) est le langage le plus connu pour la description des offres des fournisseurs.

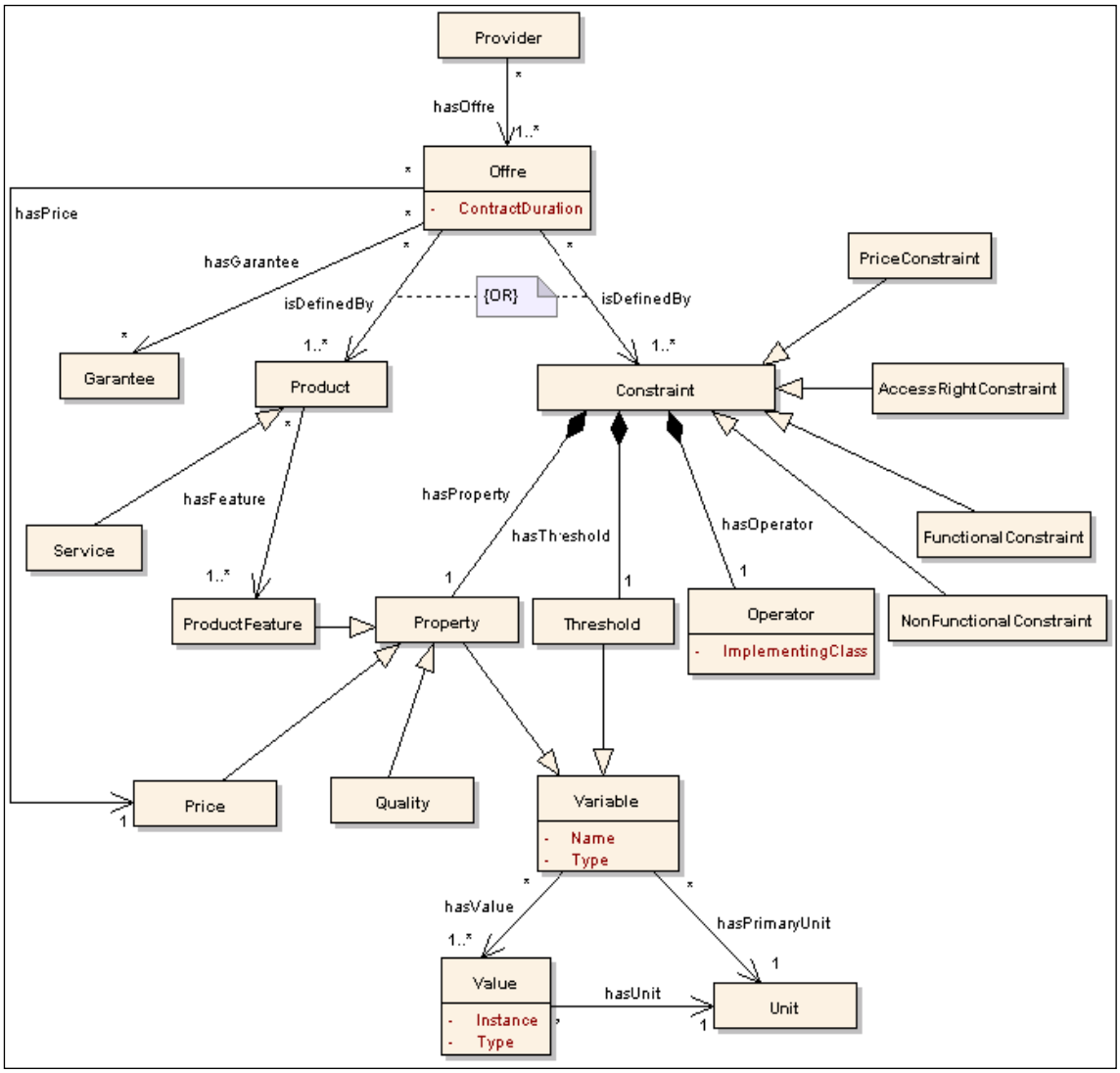

Figure 20 – Structure de l'ontologie *ProviderOnto*

Il permet de spécifier les différents niveaux de qualité de service en se basant sur les descriptions du service WSDL. WSOL est adapté pour la définition des métriques et des contraintes. Toutefois, l'extraction de l'information à partir de ce langage est difficile à utiliser en raison de sa granularité trop fine. En plus, les contraintes sont exprimées sous la forme de chaînes de caractères. Par conséquence, WSOL n'aide pas à la correspondance des offres du fournisseur avec les exigences du client. De ce fait, nous proposons un modèle des offres du fournisseur plus complet et plus riche. La Figure 20 présente le modèle des offres du fournisseur que nous proposons. Le concept racine de cette ontologie est le concept *Provider*. Ce dernier permet d'identifier le fournisseur. Initialement, le fournisseur peut mentionner toutes ses offres *Offer*. Chaque offre est définie par un ensemble de produits *Product* et de contraintes *Constraint*, un prix *Price*, une garantie *Garantee* et une durée de validité. Chaque contrainte est définie par une propriété *Proprety* qui peut conserver une qualité *Quality*, une caractéristique du produit *ProductFeature* ou un prix *Price*, un opérateur *Operator* et un seuil *Threshold*. Les propriétés et les seuils sont des variables. Chaque variable admet un nom, un type, une unité de mesure *Unit* et une ou plusieurs valeurs *Value*. Les contraintes définies par le fournisseur peuvent être fonctionnelles *FunctionalConstraint*, non fonctionnelles *NonFunctionalConstraint*, des contraintes de prix *PriceConstraint* ou de droit d'accès *AccessRightConstraint*. Les contraintes fonctionnelles concernent les propriétés directes du service ou du produit recherché par le client tel que le type des films recherchés ou la date de leur parution. Les contraintes non fonctionnelles concernent des qualités de service telles que le temps de réponse de téléchargement des films. Finalement, les contraintes de droits d'accès concernent les droits accordés par le fournisseur aux différentes catégories de clients. Par exemple, ne fournir le service de visualisation en ligne que pour une catégorie spéciale de clients.

Notre modèle présente aussi l'avantage de la réutilisation de l'ontologie standard de la description de services OWL-S (*Web Ontology Language for Services*). Cette dernière est une ontologie en OWL décrivant les propriétés et les capacités des services Web. Ainsi, nous modélisons chaque service par le concept *Service* définissant les services offerts par le fournisseur comme présenté dans la Figure 21. Chaque service est caractérisé par un ensemble de définitions de services *ServiceDefinition* et au moins une opération *Operation*. Chaque opération admet des paramètres d'entrée *InputParameters* et des paramètres de sortie *OutputParameters*. Chacun d'eux est représenté par un ensemble de paramètres *Parameter* qui doivent avoir des rangs et des valeurs. Les valeurs des paramètres sont des variables *Variable*.

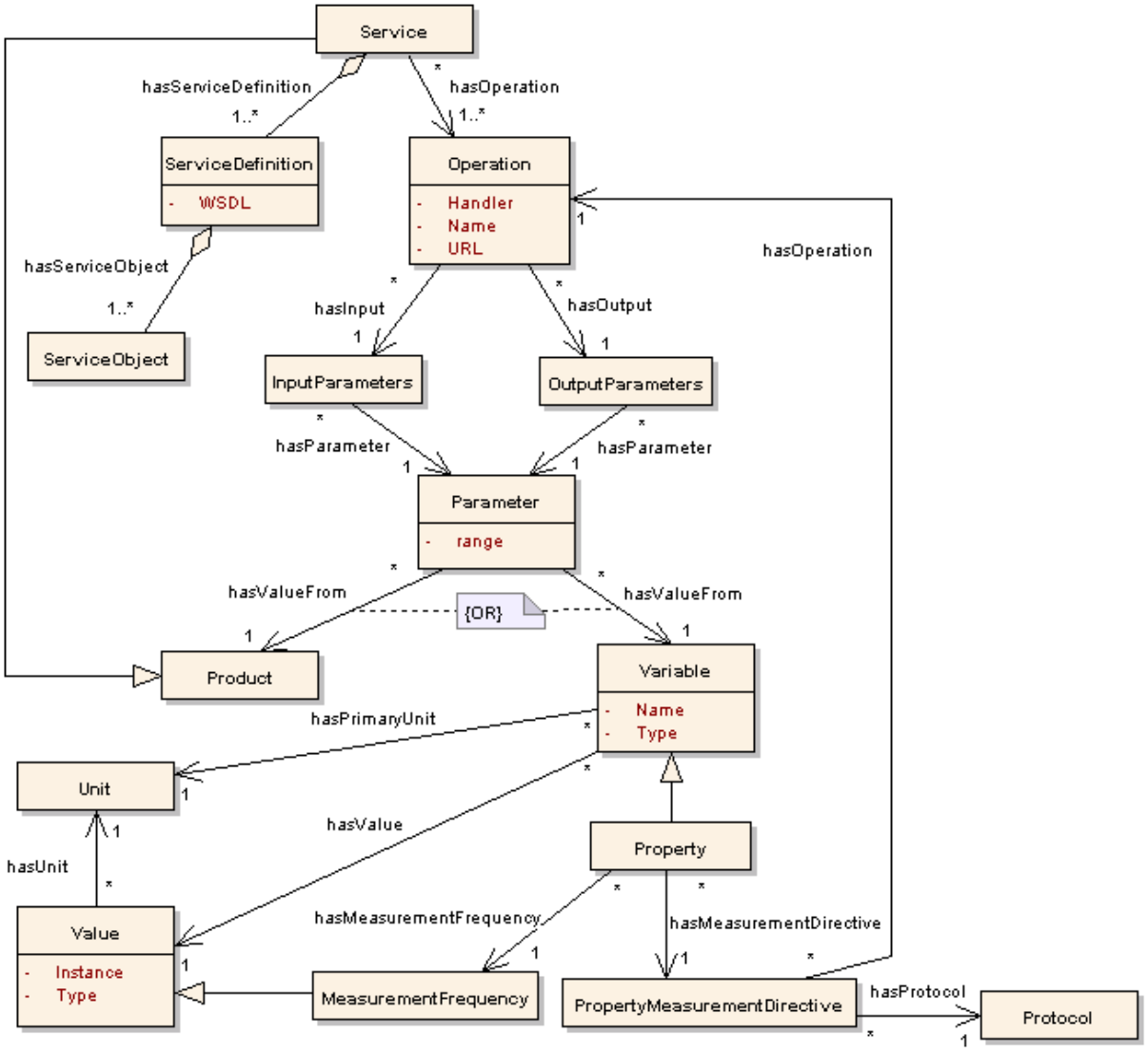

Figure 21 – Structure du concept Service de l'ontologie *ProviderOnto*

Pour assurer la qualité de service, le suivi du contrat généré est indispensable. A cet effet, il faut appliquer des mesures à des intervalles réguliers pour s'assurer de la continuité de l'adéquation des services du fournisseur par rapport aux besoins du client. Pour cela, chaque variable admet une directive de mesure *PropertyMeasurementDirective* et une fréquence de mesure *MeasurementFrequency*. Les fréquences de mesures de ce concept peuvent être périodiques définies par des valeurs numériques comme 30 millisecondes ou une heure. Elles peuvent être aussi événementielles définies par des notifications avant *OnCall* ou après *AfterCall* l'invocation du service associé au contrat. Toutes les données de ce concept vont servir dans l'étape de génération du contrat et son suivi.

3. Proposition d'un modèle de conversion des unités « UnitsOnto »

Afin d'assurer la fonctionnalité de la conversion des différentes unités de mesures dans les modèles de notre approche, nous avons développé une ontologie nommée *UnitsOnto*. La Figure 22 présente une structure simplifiée de cette ontologie. Le concept principal de cette ontologie est le concept *conversionFactor* qui admet une source et une destination présentant respectivement la première unité à convertir et la deuxième unité qui est le résultat de la conversion.

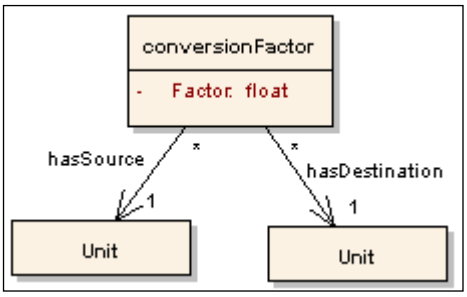

Figure 22 – Structure simplifiée de l'ontologie *UnitsOnto*

Ce concept admet aussi un facteur de conversion qui permet le passage de la première unité à la deuxième unité. Toutes les ontologies utilisées doivent alors importer cette ontologie pour exprimer les différentes unités utilisées.

4. Proposition d'un modèle de SLA : « SLAOnt »

Après avoir présenté les modèles d'entrées de notre approche (modèle des intentions des clients et des offres des fournisseurs), nous présentons dans cette section notre modèle de sortie. Ce dernier est caractérisé par le développement d'un modèle d'accords de qualité de service que nous avons nommé *SLAOnt*. Ce modèle (voir Figure 23) constitue une extension sémantique de la spécification *WSLA* que nous avons présentée dans la section IV.1.1 du chapitre 1 de ce mémoire. SLAOnt définit une ontologie décrivant les différents concepts et propriétés nécessaires constituant un contrat de qualité de service. La racine de ce modèle est le concept *SLA*. Il représente le concept des contrats que nous pouvons instancier à partir de notre modèle. Ce concept est composé des concepts suivants : *Parties*, *Obligations* et *ServiceDefinition*. Le premier concept définit les parties impliquées dans le contrat qui sont les parties signataires et les parties tierces. Les parties signataires sont généralement les fournisseurs des services et leurs clients. Les parties tierces fournissent les entités nécessaires pour la mesure de la qualité de service requise définie dans le contrat.

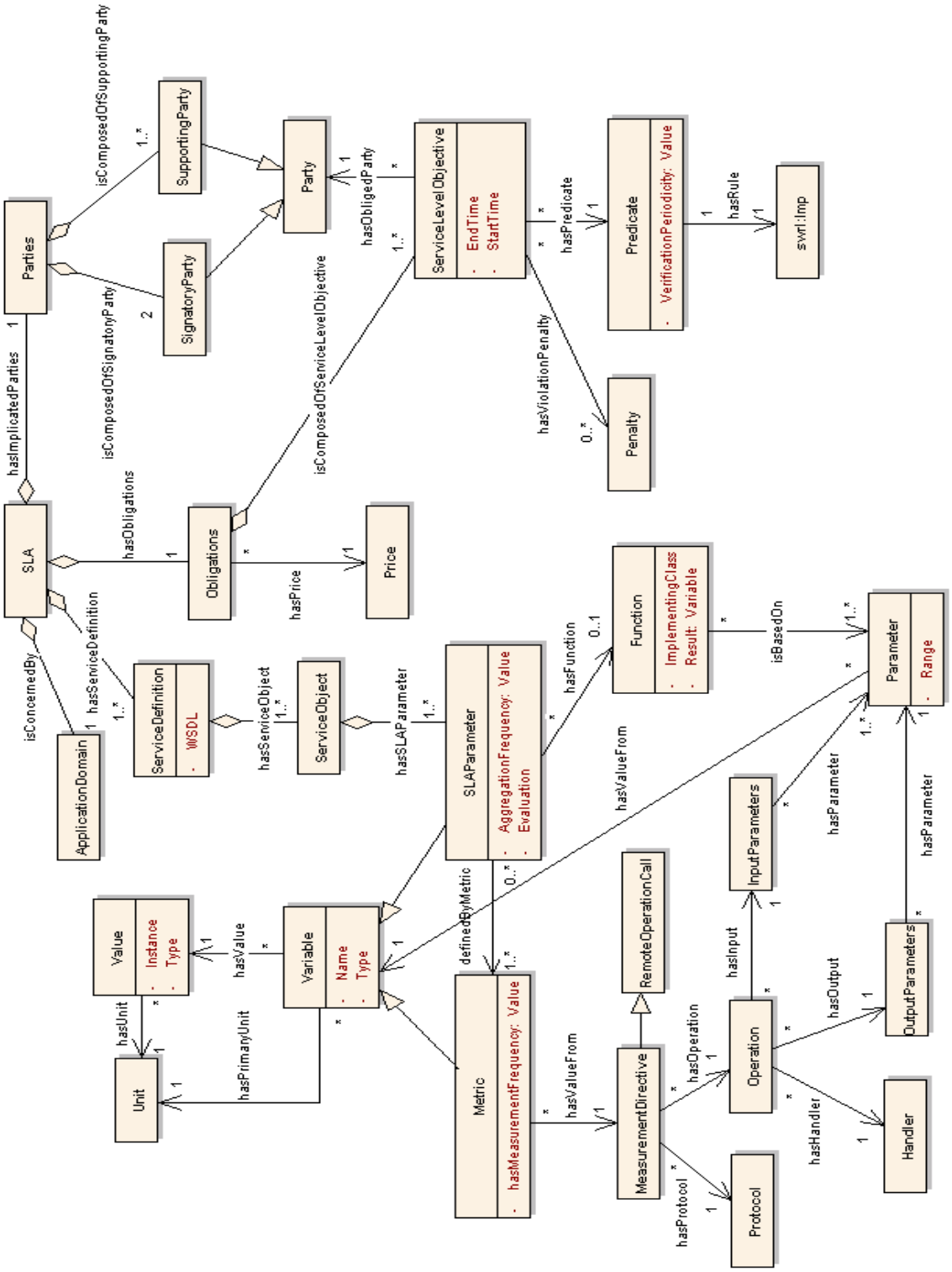

Figure 23 – *Structure de l'ontologie SLAOnt*

Le deuxième concept définit les obligations de qualité de service que les parties s'engagent à respecter. Le troisième concept décrit les services fournis concernés par ces obligations. Le contrat est défini dans un domaine d'application bien déterminé. Ainsi, nous avons défini un concept domaine d'application nommé *ApplicationDomain* pour décrire le contexte du contrat. Ce concept peut être un

lien vers une ontologie définissant les terminologies utilisées dans ce contexte. Les variables nécessaires pour la définition de la qualité de service évaluée par le contrat sont modélisées dans *SLAOnt* par le concept *SLAParameter*. Un *SLAParameter* est associé à une ou plusieurs métriques (ou des agrégations de métriques) qui définissent la qualité de service à mesurer dans le contrat. Ces métriques, modélisées par le concept *Metric*, peuvent être agrégées par une fonction mathématique ou algorithmique *Function*. Par exemple, cette fonction peut être une moyenne ou un pourcentage calculé à partir d'un ensemble de mesures. Un paramètre de QdS *SLAparameter* représente une variable qui a une unité de mesure *Unit* et qui peut être des secondes, des minutes, des pourcentages, etc. Chaque fonction est définie par un opérateur (comme *avg* pour calculer une moyenne) et un opérande qui représente une ou plusieurs mesures d'une métrique de QdS. Ces mesures sont obtenues à l'aide d'une directive de mesure *MeasurementDirective*. Ces directives sont généralement des appels d'opérations distantes qui renvoient les valeurs des métriques. Nous avons modélisé ces appels par le concept *RemoteOperationCall*. Ce dernier décrit l'opération à appeler avec son protocole d'invocation. Chaque opération est associée à un *Handler* qui représente l'objet ou la classe distante qui l'héberge. En outre, nous décrivons les paramètres d'entrées de l'opération à l'aide du concept *Parameter*. Ce dernier encapsule les valeurs à passer à l'opération distante pour son invocation. Le concept Predicate définit les obligations de qualité de service qui doivent être respectées dans le contrat. Ces obligations sont composées par des garanties *ActionGarantee* qui définissent les actions à déclencher en cas de violation. Chaque prédicat est exprimé par une règle SWRL définie dans *SLAOnt*. Le contrat a une durée bien déterminée ce qui est définie par les attributs *StartTime* et *EndTime* du concept *ServiceLevelObjective.*

La règle *PredicateEvaluationRule* présentée dans la Figure 24 montre un exemple concret d'un prédicat dans *SLAOnt*. Ce prédicat indique que si le temps de téléchargement est supérieur à 10 minutes alors une violation est déclenchée.

slaont:hasEvaluation(downloadTime, ?x) ∧ swrlb:greaterThanOrEqual(?x, 10)
→ slaActions:disseminateViolation("dowloadTimeConstraint", "false", downloadTime, ?x)

Figure 24 – *La règle «PredicateEvaluationRule»*

Dans la section suivante, nous présentons comment nous avons utilisé ces modèles pour développer notre approche d'alignement des intentions des clients et des offres des fournisseurs.

III. APPROCHE D'ALIGNEMENT DES INTENTIONS DU CLIENT AVEC LES OFFRES DU FOURNISSEUR

Après avoir élaboré le modèle intentionnel du client et le modèle des offres du fournisseur, nous présentons dans cette section les différentes étapes nécessaires de notre processus d'alignement entre

ces deux modèles. Notre approche d'alignement nommée *ODACE-SLA* (***O**ntology **D**riven **A**pproach for automati**C** **E**stablishment of **S**ervice **L**evel **A**greements*) se compose de 4 principales étapes comme présenté dans la Figure 25. La première étape (nommée étape de génération de correspondances) consiste à chercher toutes les correspondances possibles entre les termes de l'ontologie des intentions du client *ClientOnto* et ceux de l'ontologie des offres du fournisseur *ProviderOnto*. La deuxième étape (nommée étape de raffinement des correspondances), consiste à corriger les correspondances erronées générées. La troisième étape est une étape de vérification et d'évaluation des correspondances générées. Cette étape se compose de deux sous étapes : (1) l'évaluation structurelle pour tester s'il y a assez de correspondances entre les termes du client et les termes du fournisseur et (2) l'évaluation de contraintes pour tester si les contraintes du client peuvent être satisfaites par les offres du fournisseur. Finalement, nous terminons par générer automatiquement une version préliminaire d'un contrat de QdS en cas de compatibilité totale entre les termes du client et les termes du fournisseur.

Toutes ces étapes seront orchestrées par une ontologie globale que nous avons nommée *MatchingOnto*. Cette dernière importe les ontologies contenant l'intention du client et les offres du fournisseur. Elle est utilisée tout au long du processus d'alignement pour enregistrer toutes les nouvelles connaissances générées par inférence dans chaque étape de notre processus d'alignement.

Nous présentons plus de détails sur les étapes de ce processus dans les sections suivantes.

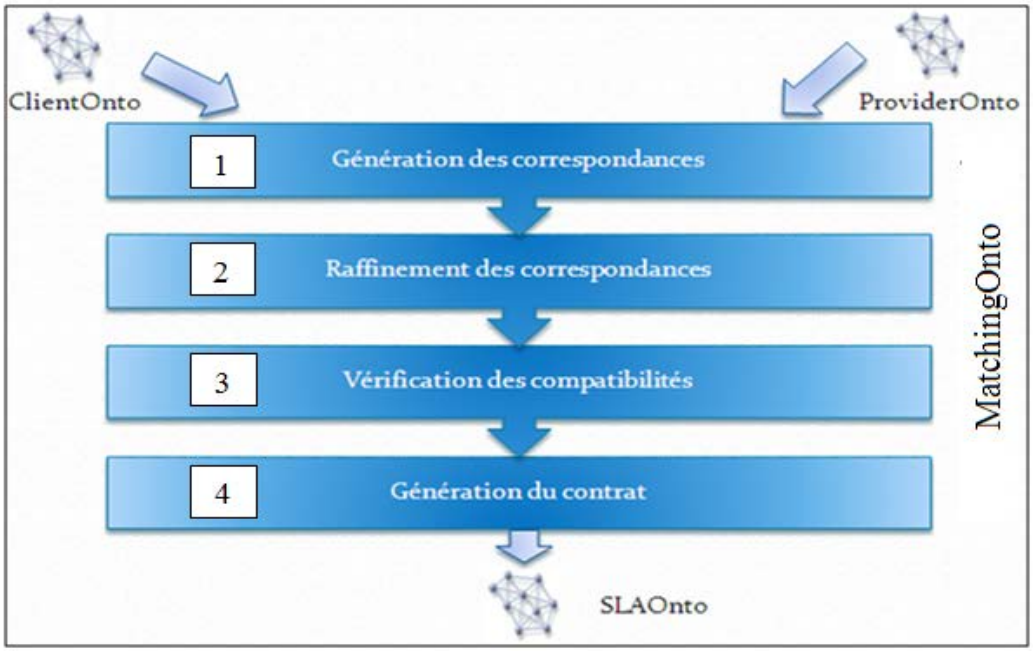

Figure 25 – Architecture simplifiée de l'application

1. Génération des correspondances

Pour pouvoir étudier la compatibilité entre les besoins du client et les offres du fournisseur, il est indispensable d'étudier dans une première étape le degré de similarité sémantique entre leurs

ontologies pour savoir s'ils se situent dans le même domaine de connaissances ou non. Dans notre cas, nous nous intéressons principalement aux correspondances entre les instances des concepts de ces ontologies.

1.1 DEFINITION D'UNE CORRESPONDANCE

Nous appelons une correspondance tout lien sémantique existant entre deux instances de concepts appartenant à deux ontologies différentes O_1 et O_2. Chaque correspondance est composée d'un couple d'instances $(C_{i,k}, C_{j,m})$ avec $C_{i,k}$ une instance du concept C_i appartenant à l'ontologie O_1 et $C_{j,m}$ une instance du concept C_j appartenant à l'ontologie O_2. Dans la section suivante, nous présentons comment nous modélisons la notion de correspondance.

1.2 MODELISATION D'UNE CORRESPONDANCE

Nous modélisons une correspondance par le concept *Correspondence*. Ce concept est défini par deux relations comme illustré dans la Figure 26. La première relation *hasSourceTerm* fait référence sur une instance de la première ontologie *Root Concept1* et la deuxième relation *hasDestinationTerm* concerne une autre instance de la deuxième ontologie *Root Concept2*.

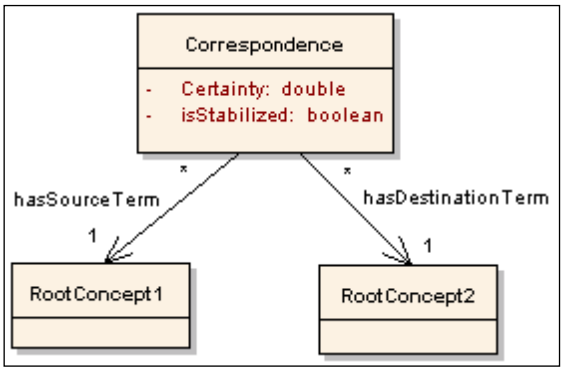

Figure 26 – *Modélisation d'une correspondance*

Chaque correspondance est définie aussi par un attribut nommé *Certainty* qui est un réel entre 0 et 1 désignant la probabilité de son existence. La valeur de cette probabilité représente la mesure de similarité sémantique entre l'instance source et l'instance destination de la correspondance considérée. Ainsi, si l'instance source et l'instance destination sont sémantiquement proches, la correspondance aura une grande probabilité d'existence. Sinon, cette probabilité sera faible. Dans la section suivante, nous expliquons davantage cette notion de probabilité d'existence de correspondance.

1.3 PROBABILITE D'EXISTENCE D'UNE CORRESPONDANCE

Nous désignons par probabilité d'existence d'une correspondance la similarité sémantique entre son

instance source et son instance destination. La similarité sémantique représente l'équivalence de sens entre ces deux instances appartenant à deux ontologies différentes. Chaque instance $C_{i,k}$ peut être définie par un seul terme W par exemple *Film* ou par plusieurs termes constituant une expression T par exemple *Film Download Time*. Pour mesurer la similarité entre ces deux instances, nous nous sommes basés sur la formule **(F1)** de [103].

$$\text{Sim}(T1, T2) = 1/2\left(\frac{\sum_{w\in\{T1\}}(\text{maxSim}(w,T2)*\text{Importance}(w))}{\sum w\in\{T1\}\text{Importance}(w)} + \frac{\sum_{w\in\{T2\}}(\text{maxSim}(w,T1)*\text{importance}(w))}{\sum w\in\{T2\}\text{Importance}(w)}\right) \quad \textbf{(F1)}$$

Avec :

> ***T_1*** et ***T_2*** deux instances de concepts des deux ontologies ***O_1*** et ***O_2***. Ces instances sont généralement des expressions de termes ***W_i***.
>
> ***W^1_i*** est le i ème terme dans l'expression ***T_1***
>
> ***W^2_j*** est le j ème terme dans l'expression ***T_2***
>
> ***maxSim (w, T)*** est le maximum des mesures de similarité sémantique entre le terme ***W*** et tous les termes de l'expression ***T***. Ces mesures de similarités représentent la probabilité d'existence d'une correspondance entre le terme ***W*** et chaque terme de l'expression ***T***.
>
> ***Importance (w)*** est une pondération pour le terme W selon son importance dans son expression ***T***.

Cette formule (**F1**) est une moyenne pondérée des maxima des similarités sémantiques individuelles entre chaque terme ***W^1_i*** de l'expression ***T1*** d'une part et chaque terme de ***W^2_j*** de l'expression ***T2*** d'autre part. La valeur de la similarité individuelle entre deux termes ***W^1_i*** et ***W^2_j*** peut être déduite à partir de techniques existantes de mesures sémantiques [103].

Les pondérations ***Importance (w)*** correspondent à l'importance de chaque terme ***W*** dans son expression ***T***. Si nous considérons le cas de la langue anglaise, les termes ont une importance croissante selon leur ordre d'apparition dans les expressions nominales. Par exemple, dans l'expression *Film Download Time*, le terme *Time* est plus important que *Download* qui est lui aussi plus important que *Film*. Dans le cas des expressions verbales, l'importance des termes est décroissante selon leur ordre d'apparition dans ces expressions. Par exemple, dans l'expression *Download Film*, le terme *Download* est plus important que *Film*.

Après avoir présenté le processus de mesure de probabilité d'existence des correspondances, nous

illustrons dans la section suivante comment nous l'avons utilisé pour assurer la génération de correspondances directe et indirecte dans notre approche.

1.4 GENERATION DES CORRESPONDANCES DIRECTES

Le principe de l'étape de génération de correspondances directes consiste à rechercher tous les liens sémantiques directs entre l'ontologie des intentions du client *ClientOnto* et l'ontologie des offres du fournisseur *ProviderOnto* (voir Figure 27).

Ces liens directs correspondent à une mesure de similarité sémantique entre les termes de ces deux ontologies. Les valeurs de ces mesures peuvent être calculées selon le principe de la section 1.3 de ce même chapitre.

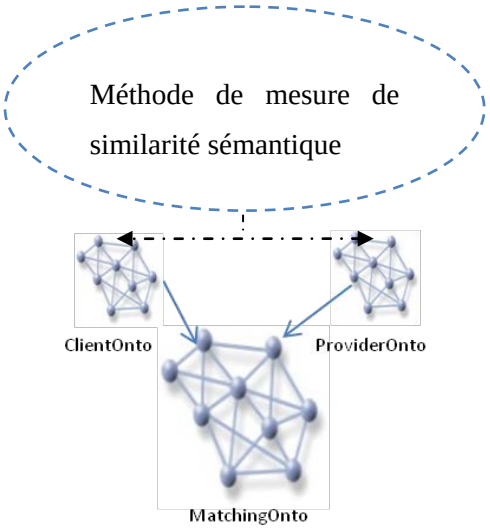

Figure 27 – Schéma générique de la correspondance directe entre l'intention du client et les offres du fournisseur

Pour générer ces correspondances directes, nous avons développé un algorithme présenté dans la Figure 28. Cet algorithme consiste à parcourir toutes les instances $C_{i,k}$ des deux ontologies pour récupérer toutes les paires formées d'une instance de la première ontologie et d'une instance de la deuxième ontologie. Puis, il calcule la valeur de similarité sémantique entre ces deux instances.

```
1.  generateCorrespondences(rootTerm1, rootTerm2)
2.    For each Ci,k instanceOf rootTerm1 do
3.      For each Cj,m instanceOf rootTerm2 do
4.        semanticSimilarity := getSemanticSimilarity(Ci,k, Cj,m);
5.        if semanticSimilarity > MIN_SIMILARITY
6.          createCorrespondence(Ci,k, Cj,m, semanticSimilarity);
```

Figure 28 – Algorithme de correspondances entre les termes du client et les termes du fournisseur

Si cette valeur est supérieure à un seuil prédéfini par l'expert commercial du fournisseur (Ligne 5 de la Figure 28), alors la correspondance ainsi retrouvée sera créée dans une nouvelle ontologie d'alignement nommée *MatchingOnto*. Cette correspondance a comme source le premier terme de la paire, comme destination le deuxième terme de la paire et comme mesure la valeur de similarité sémantique ainsi calculée (Ligne 6 de la Figure 28).

1.5 GENERATION DES CORRESPONDANCES INDIRECTES

Après avoir défini l'étape de génération de correspondances directes présentée dans la section précédente, nous avons constaté qu'elle ne peut pas couvrir tous les types de correspondances surtout lorsqu'il existe des dépendances mathématiques entre les instances des deux ontologies. Par exemple, l'instance *temps de téléchargement* de l'ontologie des intentions du client dépend fortement de l'instance *débit* de l'ontologie des offres du fournisseur à travers une relation mathématique qui est « temps de téléchargement= taille / débit». Pour assurer ce type de lien, nous avons remarqué la nécessité d'un modèle intermédiaire entre l'ontologie du client et l'ontologie du fournisseur. Ce modèle a pour rôle de gérer les dépendances entre les qualités de service des deux ontologies afin de pouvoir calculer, déduire ou comparer des valeurs demandées par le client. Dans la littérature, nous trouvons quelques modèles de QdS tels qu'OWL-QoS [67], QoSOnt [69], SL-Ontology [72], WS-QoS [73], FIPA QoS [74] et MOQ [78]. Cependant, ces modèles n'expriment pas la dépendance sémantique entre les métriques lorsqu'elles sont composées. En conséquence, nous ne pouvons pas déterminer la valeur d'une métrique de QdS lorsqu'elle est composée. Pour remédier à ces insuffisances, nous avons développé notre propre ontologie de QdS nommée *QoSOnto* en s'inspirant de l'ontologie OWL-QoS. Notre ontologie (présentée dans la Figure 29) décrit un ensemble de métriques modélisées par le concept *QoSMetric*.

Chaque métrique est définie sous la forme d'une variable *Variable* qui admet un nom, un type et une unité de mesure *Unit*. Par exemple, la métrique « temps de réponse » admet comme type « réel » et comme unité « milliseconde ». Une variable peut aussi avoir une valeur qui admet à son tour un type, une instance et une unité de mesure. La métrique admet aussi un attribut nommé *BetterQualityDirection* qui représente le meilleur sens (décroissant ou croissant) de la métrique considérée. Par exemple, cet attribut a la valeur « décroissant » pour la métrique « temps de réponse » puisque ce dernier est meilleur s'il est plus petit. Les métriques peuvent être composées en se basant sur des fonctions *Function* mathématiques ou algorithmiques. Chaque fonction admet un ensemble de paramètres d'entrée qui peuvent être issus d'autres métriques de QdS. Chaque paramètre a un *rang* qui désigne sa position dans la fonction. Il peut avoir une valeur représentée par une variable.

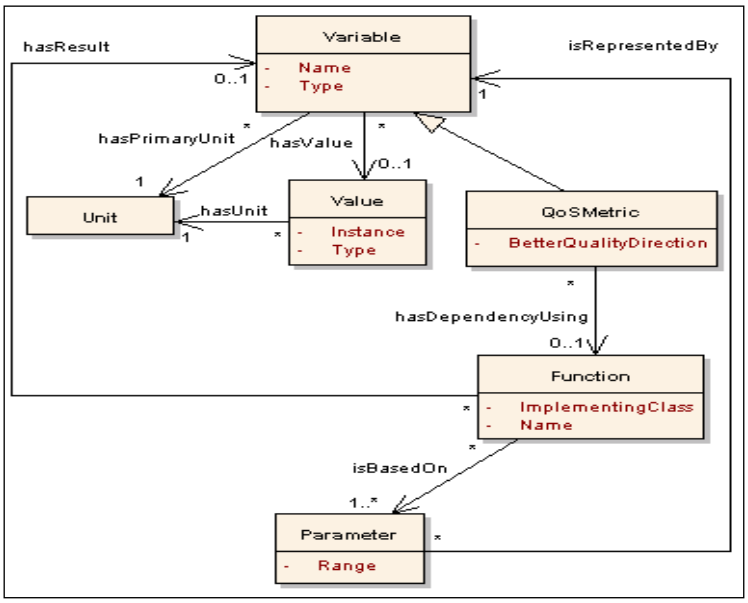

Figure 29 – Structure de l'ontologie *QosOnto*

Pour générer les correspondances indirectes, nous avons repris le principe de l'algorithme de génération des correspondances directes (voir Figure 28) présenté dans la section précédente. Notre algorithme prend, cette fois ci, trois ontologies en entrée (voir Figure 30).

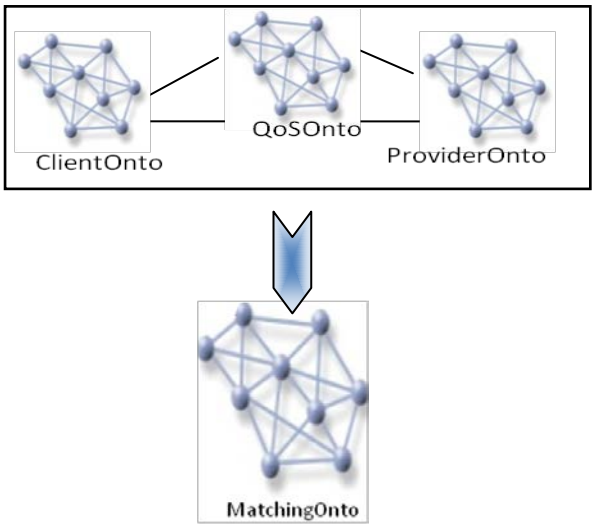

Figure 30 – Schéma générique de la correspondance indirecte entre l'intention du client et les offres du fournisseur

L'accès à ces ontologies se déroule de la manière suivante : En un premier temps, nous cherchons les similarités sémantiques entre les instances de l'ontologie du client *ClientOnto* et les instances de l'ontologie de qualité de service *QosOnto*. Puis, en un deuxième temps, nous cherchons les similarités

entre les instances de l'ontologie de qualité de service *QosOnto* et les instances de l'ontologie du fournisseur *ProviderOnto*. Les valeurs de ces mesures de similarité peuvent être calculées aussi selon le même principe que la section 1.3.

La Figure 31 présente un exemple de correspondance indirecte. Le terme *Film Download Time* de l'ontologie du client *ClientOnto* possède une correspondance indirecte avec l'ontologie du fournisseur à travers le terme *Transfer Time* de l'ontologie de QdS *QosOnto*. La valeur correspondante de ce terme dépend des termes taille du film *Film Size* et débit de transfert du film *Throughput* de l'ontologie du fournisseur *ProviderOnto*.

Finalement, les correspondances retrouvées vont être créées dans l'ontologie d'alignement *MatchingOnto* pour rejoindre les correspondances directes déjà créées dans la section précédente. Les méthodes de mesures de similarités sémantiques existantes ont quelques imprécisions dues principalement à différentes interprétations possibles d'un même terme.

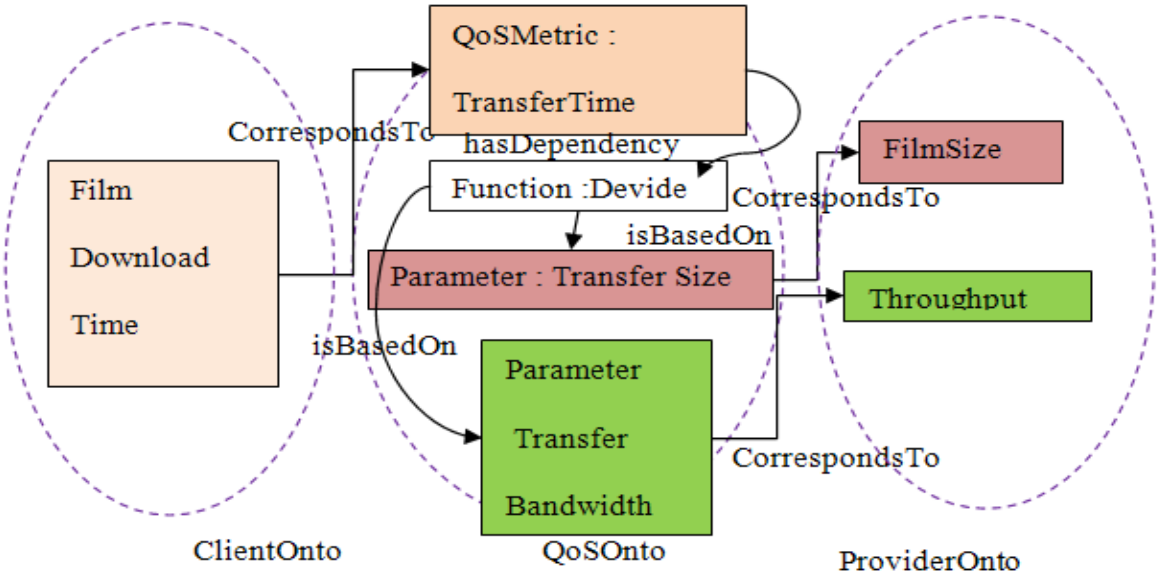

Figure 31 – Exemple d'une correspondance indirecte avec l'ontologie de QdS

Ceci induit la production de quelques valeurs erronées isolées. Ces valeurs peuvent ne pas correspondre avec le résultat souhaité par l'être humain. Pour rapprocher ces valeurs à des résultats plus correctes, nous procédons à l'étape de raffinement des correspondances.

2. Raffinement des correspondances

L'objectif principal de l'étape de raffinement des correspondances consiste à rectifier les erreurs isolées des correspondances générées dans la section 1 de ce chapitre. Elle se compose de deux sous étapes : la génération des adjacences et la stabilisation des correspondances. La génération des adjacences consiste à chercher des liens linguistiques entre correspondances en se basant sur les

relations existantes entre leurs termes sources et/ou leurs termes destinations. L'étape de stabilisation des correspondances consiste à équilibrer les probabilités d'existences de ces correspondances en utilisant une méthode de propagation de mesures de similarité sémantique. Cette propagation se base sur la dépendance des correspondances les unes par rapport aux autres en utilisant les adjacences créées lors de l'étape précédente.

2.1 GENERATION DES ADJACENCES

2.1.1 Définition et principe d'une adjacence

Une adjacence est un lien linguistique entre les termes sources ou les termes destinations de deux correspondances différentes. Ce lien linguistique est proportionnel à la distance de ces termes dans les corpus linguistiques. Si cette distance est grande, l'adjacence sera faible. Sinon, elle sera importante. La notion d'adjacence représente la dépendance d'une correspondance sur une autre dont les termes sources ou destinations sont proches dans les corpus linguistiques.

2.1.2 Modélisation d'une adjacence

Une adjacence, comme présentée dans la Figure 32, est un concept lié par deux relations nommées *hasSourceCorrespondence* et *hasDestinationCorrespondence*. Chacune d'elles fait référence à une correspondance.

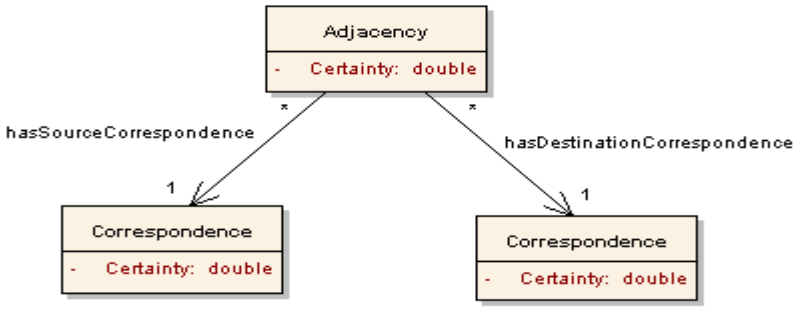

Figure 32 – Modélisation d'une adjacence

Une adjacence est définie aussi par une probabilité *certainty* qui est une pondération de la dépendance des deux correspondances l'une sur l'autre. Cette pondération a une valeur entre 0 et 1.

Deux correspondances sont dites adjacentes si leurs sources ou leurs destinations ont une relation linguistiques entre elles (comme présenté dans la Figure 33). Dans ce cas, nous procédons à la création d'une nouvelle instance du concept adjacence dans l'ontologie d'alignement *MatchingOnto* en lui attribuant comme source la première correspondance, comme destination la deuxième correspondance et comme pondération la moyenne de la proximité linguistique entre les deux termes sources et les

deux termes destinations de ces correspondances.

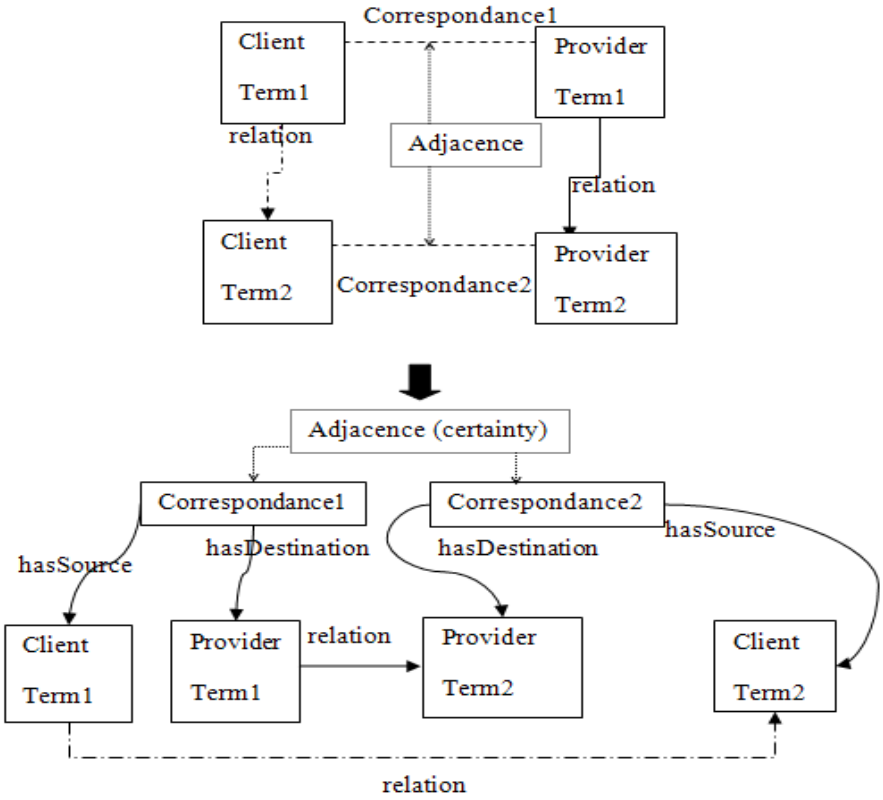

Figure 33 – Modèle de génération d'adjacence

2.1.3 Algorithme de génération des adjacences

Le principe de notre algorithme de génération des adjacences (présenté dans la Figure 34) se base sur le parcours de toutes les instances des correspondances créées dans l'étape précédente.

Pour chaque couple de correspondances *(correspondance 1, correspondance 2)*, nous récupérons les termes sources correspondants (*source 1*, *source 2*) (Ligne 3 de la Figure 34) et les termes destinations (*destination1, destination 2*) (Ligne 4 de la Figure 34). S'il existe une relation entre le terme *source 1* et le terme *source 2* ou bien entre le terme *destination 1* et le terme *destination 2*, nous calculons la valeur de proximité linguistique entre les termes en relation (Ligne 10 à 15 de la Figure 34). Suite à ce calcul, nous procédons à la création d'une nouvelle instance du concept *adjacence* dans l'ontologie d'alignement (*MatchingOnto.owl*). Cette instance aura comme extrémités les deux correspondances *correspondance1* et *correspondance2* (ligne 16 et 17 de la Figure 34). La moyenne des proximités linguistiques entre les termes *source1* et *source2* d'une part et *destination 1* et *destination2* d'autre part sera affectée à la pondération de cette nouvelle adjacence. Cette pondération constituera le poids de dépendance de la correspondance1 sur la probabilité d'existence de la correspondance2 (et vice-versa) dans le processus de stabilisation des correspondances que nous détaillons dans la section suivante.

```
1.  createCorrespondenceAdjacencies()
2.    For each correspondence1 instanceOf Correspondence Do
3.      sourceTerm1:= getSourceTerm(correspondence1);
4.      destinationTerm1:= get DestinationTerm(correspondence1);
5.      For each correspondence2≠correspondence1 instanceOf Correspondence Do
6.        sourceTerm2:= getSourceTerm(correspondence2);
7.        destinationTerm2:= getDestinationTerm(correspondence2);
8.        linguisticRelatedness:=0;
9.        adjacencyNumber:=0;
10.       If ∃ R ∈ Rclient OR R ∈ Rqos | R=(sourceTerm1, sourceTerm2)
11.         linguisticRelatedness:= wordnet::relatedness(sourceTerm1, sourceTerm2);
12.         adjacencyNumber++;
13.       If ∃ R ∈ Rqos OR R ∈ Rprovider | R=( destinationTerm1, destinationTerm2)
14.         linguisticRelatedness+= wordnet::relatedness(destinationTerm1, destinationTerm2);
15.         adjacencyNumber++;
16.       If adjacencyNumber≠0 then
17.         linguisticRelatedness:= linguisticRelatedness/adjacencyNumber;
18.         createAdjacency(correspondence1, correspondence2, linguisticRelatedness);
```

Figure 34 – Algorithme de génération des adjacences entre les correspondances

2.2 STABILISATION DES CORRESPONDANCES

2.2.1 Définition et objectif de la stabilisation

Nous appelons étape de stabilisation des correspondances le processus qui engendre un équilibre sémantique entre les différentes correspondances constituant l'ontologie d'alignement *MatchingOnto*. Le principe de stabilisation consiste à corriger les valeurs des similarités sémantiques des correspondances obtenues en se basant sur les adjacences ainsi créées. Chaque correspondance dépend de probabilités d'existences des correspondances voisines. Cette dépendance peut agir d'une manière croissante ou décroissante sur la correspondance concernée.

2.2.2 Algorithme de stabilisation

Pour mettre en œuvre le principe de stabilisation, nous avons développé un algorithme itératif de propagation des probabilités d'existences des correspondances. Initialement, nous considérons que les correspondances qui ont des valeurs de probabilités d'existences égales à 1 sont stables et elles ne seront pas incluses dans l'étape de stabilisation (Ligne 3 et 4 de la Figure 35). Si la valeur de probabilité d'existence est différente de 1 (c.-à-d. $\in [0,1[$), alors nous marquons cette correspondance comme étant instable (Ligne 5 de la Figure 35). L'algorithme procédera ainsi à sa stabilisation pour rectifier sa valeur par rapport à ses nœuds voisins dépendants.

```
1.  initializeCorrespondencesForStabilization()
2.    For each correspondence instanceOf Correspondence Do
3.      If correspondence.hasCertainty ==1
4.        mark(correspondence, "stabilized");
5.      else mark(correspondence,"unStabilized");
```

Figure 35 – Algorithme d'initialisation de stabilisation des adjacences entre les correspondances

En effet, pour chaque correspondance *Ci* (voir Figure 36) qui a une probabilité *pi*, nous récupérons toutes les adjacences *Wij* ayant comme pondération *pij* dont la correspondance *Ci* est l'une de ses extrémités (source ou destination). Pour chaque adjacence trouvée, nous récupérons la probabilité *pj* de sa deuxième extrémité qui est la correspondance *Cj*. La fonction de moyenne pondérée (F2) est ainsi appliquée pour calculer la nouvelle probabilité *pi'* de la correspondance *Ci* où n est le nombre d'adjacences de *Cj*. Ce calcul sera répété tant qu'il existe des nœuds qui n'ont pas atteint l'état de stabilité.

$$pi' = \frac{pi + \sum_{j=1}^{n}(\frac{Wij}{n} * pj)}{1 + \sum_{j=1}^{n}(\frac{Wij}{n})}$$

(F2)

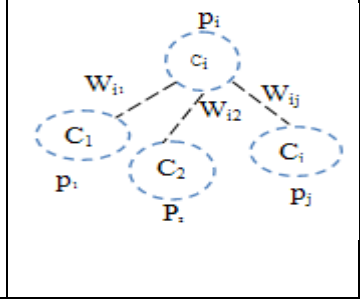

Figure 36 – Formule de propagation des probabilités des correspondances

Cette stabilité est définie lorsque la différence entre l'ancienne valeur de similarité sémantique et la nouvelle valeur obtenue devienne faible (par exemple 0,01). Ce seuil de stabilisation est fixé par l'expert commercial du fournisseur.

```
1.  correspondenceStabilization()
2.    For each correspondence1 instanceOf Correspondence | correspondence1.isMarked("unStabilized") Do
3.      oldCertainty:= correspondence1.hasCertainty;
4.      ExtraCertainties:=0;
5.      AdjacenciesRelatedness:=0;
6.      For each adjacency instanceOf Adjacency | adjacency.hasExtremity(correspondence1) Do
7.        correspondence2 := adjacency.hasExtremity | correspondence2≠correspondence1;
8.        AdjacenciesRelatedness +=adjacency.hasRelatedness;
9.          ExtraCertainties +=adjacency.hasRelatedness*correspondence2.hasCertainty;
10.     newCertainty:=(oldCertainty+ ExtraCertainties)/(1+ AdjacenciesRelatedness);
11.     correspondence1.hasCertainty:= newCertainty;
12.     If newCertainty-oldCertainty<CERTAINTY_PRECISION  mark(correspondence1,"stabilized");
13.  correspondenceStabilization();
```

Figure 37 – Algorithme de stabilisation des adjacences entre les correspondances

3. Vérification de compatibilité

3.1 DEFINITION ET OBJECTIFS

Le processus de vérification de compatibilité se base sur les résultats des étapes précédentes d'alignement. Il comporte deux sous étapes : l'évaluation structurelle et l'évaluation de contraintes.

L'évaluation structurelle consiste à tester l'existence d'un nombre suffisant de correspondances entre les intentions du client et les offres du fournisseur. L'évaluation de contraintes consiste à déterminer la satisfaction des contraintes du client avec les contraintes du fournisseur. Dans ce qui suit, nous présentons les algorithmes que nous avons développés pour cette étape.

3.2 ALGORITHME DE VERIFICATION DE COMPATIBILITE STRUCTURELLE

Le principe de l'algorithme de vérification de compatibilité structurelle consiste à calculer la moyenne globale de toutes les mesures de similarités sémantiques des correspondances qui ont comme source un terme de l'ontologie du client et comme destination un terme de l'ontologie du fournisseur (Ligne 4 à 11 de la Figure 38).

Pour pouvoir décider de la satisfaction de la compatibilité structurelle entre l'ontologie des intentions du client et l'ontologie des offres du fournisseur, nous avons choisi un seuil égal à 0.6. Ce seuil désigne la moyenne globale des mesures de similarités de correspondances de l'ontologie d'alignement *MatchingOnto*. Il est fixé par l'expert commercial du fournisseur. Si la moyenne globale est inférieure à 0.6 par exemple, alors nous affirmons l'absence de compatibilité structurelle entre les intentions du client et les offres du fournisseur (Ligne 12 et 13 de la Figure 38). Si cette compatibilité est vérifiée, alors nous abordons la deuxième sous étape qui est la vérification de compatibilités de contraintes entre le client et le fournisseur.

```
1.  checkGlobalSemanticCompatibility()
2.    optimisticCertainties:=0;
3.    clientTermsNumber:=0;
4.      For each clientTerm instanceOf ClientTerm Do
5.      optimisticClientTermCertainty:=0;
6.      clientTermsNumber++;
7.      For each correspondence instanceOf Correspondence | correspondence.hasSourceTerm==clientTerm Do
8.        If correspondence.hasCertainty> optimisticClientTermCertainty
9.          optimisticClientTermCertainty:= correspondence.hasCertainty;
10.     optimisticCertainties+= optimisticClientTermCertainty;
11.   optimisticCertainty:= optimisticCertainties/ clientTermsNumber;
12.   if optimisticCertainty<MIN_CERTAINTY
13.     compatibiliy:=false;
```

Figure 38– Algorithme de vérification de la compatibilité sémantique globale

3.3 ALGORITHME DE VERIFICATION DE COMPATIBILITE DE CONTRAINTES

Le processus de vérification de compatibilité de contraintes consiste à vérifier si les contraintes demandées par le client sont satisfaites ou non avec les offres du fournisseur. Nous pouvons distinguer trois types de compatibilités selon le degré de satisfaction des contraintes entre le client et le fournisseur : compatibilité totale, compatibilité partielle ou incompatibilité. La compatibilité totale signifie que toutes les contraintes du client peuvent être satisfaites par les offres du fournisseur.

Concernant la compatibilité partielle, elle implique la satisfaction d'au moins une seule contrainte.

L'incompatibilité est décidée en absence totale de contraintes satisfaites avec les offres du fournisseur. Dans cette étape, nous nous intéressons principalement au concept *Constraint* de l'ontologie des intentions du client. La Figure 39 présente le modèle fonctionnel de contraintes que nous avons utilisé dans nos ontologies.

Comme décrit dans *ClientOnto* présenté dans la Figure 19, une contrainte est composée d'une « opération », une « propriété » et un seuil *threshold*. Chaque propriété admet un prix, une qualité et une caractéristique du produit *product Feature*. L'algorithme d'évaluation de compatibilité de toutes les contraintes demandées par le client avec les offres du fournisseur est présenté dans la Figure 40. Le principe de cet algorithme consiste à récupérer les propriétés de toutes les contraintes du client ainsi que leurs seuils et leurs opérateurs (Ligne 3 à 11 de la Figure 40).

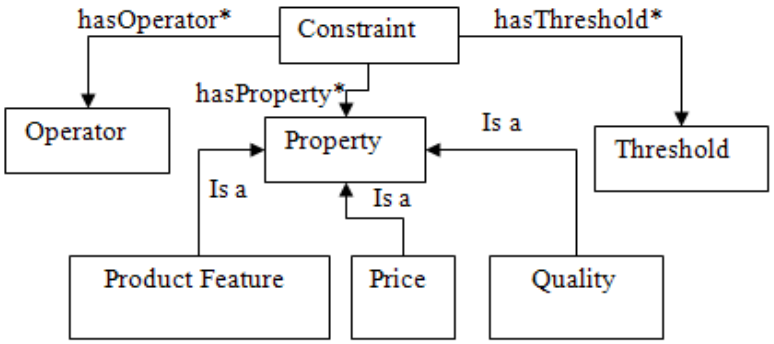

Figure 39 – Le modèle de contrainte

Par la suite, nous passons à la vérification de la compatibilité de toutes les propriétés du client avec celles du fournisseur. En effet, pour chaque propriété, nous récupérons toutes les correspondances dont leur source est cette propriété (ligne 6 de la Figure 40). Pour chaque correspondance trouvée, si sa destination est un terme du fournisseur alors ce terme sera ajouté comme valeur possible à la propriété du client (Ligne 7 et 8 de la Figure 40).

Si la destination de la correspondance est un terme de l'ontologie de QdS, nous récupérons la fonction qui permet de la calculer toutes les correspondances qui lient les opérandes de cette fonction avec le terme du fournisseur. Nous calculons ainsi les valeurs possibles de la propriété du client en utilisant les correspondances indirectes à travers l'ontologie de QdS. Ce calcul est fait grâce à une règle d'inférence intitulée *CalculateFunctionRule* (Ligne 12 à 18 de la Figure 40). Enfin, toutes les contraintes sont évaluées en même temps en utilisant les valeurs correspondantes possibles à partir des correspondances directes et indirectes. Cette évaluation est assurée par la règle d'inférence *CheckConstraintRule*. En cas d'incompatibilité totale ou partielle, une notification s'affiche au client pour l'informer de la ou des contraintes non satisfaites. Dans ce cas, il n'y aura pas de passage à l'étape de génération de contrat. En cas de compatibilité totale, l'étape suivante sera la génération

d'une version préliminaire d'un contrat de qualité de service entre le client et le fournisseur.

```
1.  checkConstraintsCompatibility()
2.    checkConstraintsRule:="";
3.    For each constraint instanceOf Constraint Do
4.      createConcept(constraint.hasProperty+"ProviderValues");
5.      checkConstraintsRule+= constraint.hasProperty+"ProviderValues(?value)^checkConstraint(?value,"+
    constraint.hasOperator +","+ constraint.hasThreshold +")";
6.      For each corresp instanceOf Correspondence | corresp.hasSourceTerm==constraint.hasProperty Do
7.        If corresp.hasDestination instanceOf providerTerm
8.          addInstance(constraint.hasProperty +"ProviderValues", corresp.hasDestinationTerm)
9.        If corresp.hasDestination instanceOf QdSTerm
10.         function:=corresp.hasDestinationTerm.hasdependencyUsing
11.         operands:=corresp.hasDestinationTerm.hasdependencyUsing.isBasedOn
12.         calculateFunctionRule:="calculateFunction("+function+","
13.         For each operand in Operands do
14.           calculateFunctionRule:=operand+"Values(?"+operand+"value)^"+calculateFunctionRule
15.           For each correspOp instanceOf Correspondence | correspOp.hasSourceTerm==operand Do
16.             addInstance(operand+"Values", correspOp.hasDestinationTerm)
17.         calculateFunctionRule+="?result)→"+constraint.hasProperty+"ProviderValues(?result)";
18.         runRule(calculateFunctionRule);
19.     if (Not isLast(client:property, client:propertyVector) )then
20.       checkConstraintsRule+="^"
21.     else  checkConstraintsRule+="→ Compatibility"
22.   Compatibility:= SWRLEngine.infer(checkConstraintsRule)
```

Figure 40 – Algorithme de vérification de la compatibilité des contraintes

4. Génération de contrats

4.1 DEFINITION ET PRINCIPE

Nous appelons phase de génération de contrat l'étape de production d'une version ontologique d'un contrat de qualité de service entre un client et un fournisseur.

Cette étape se base essentiellement sur la structure de l'ontologie *SLAOnt* que nous avons présentée dans la section 4. Elle se fonde aussi sur les informations disponibles dans l'ontologie d'alignement *MatchingOnto* (voir Figure 41). Lors de la génération du contrat, nous privilégions les terminologies du client tout en gardant un lien avec les concepts du fournisseur. Dans la section suivante, nous présentons le processus de génération de contrat de QdS que nous avons élaboré.

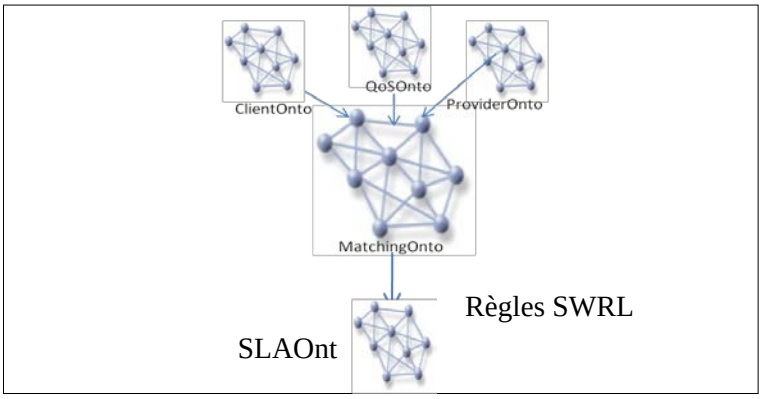

Figure 41 – Processus général de génération de contrat

4.2 PROCESSUS DE GENERATION DE CONTRAT

Après avoir validé les trois premières étapes de notre processus d'alignement, nous abordons la dernière étape de génération de contrat de QdS. Nous rappelons que cette étape se base sur la structure du modèle de SLA *SLAOnt*. Comme nous avons présenté dans la section 4, les concepts de base de ce modèle sont les directives de mesure, les métriques, les paramètres de QdS et les prédicats. Pour assurer cette génération, nous avons défini un algorithme présenté dans la Figure 42. Pour chaque contrainte du client, nous récupérons sa propriété, son opérateur et son seuil (voir Lignes 1 et 2 de la Figure 42). Nous retenons la correspondance qui a la plus haute probabilité d'existence ayant comme source la propriété de la contrainte (Ligne 5 de la Figure 42). Si la destination de cette correspondance est un terme du fournisseur, dans ce cas, la correspondance est directe. Nous créons ainsi une métrique basée sur le terme destination du fournisseur avec la directive de mesure qui permet de l'évaluer. Ensuite, nous générons un paramètre de QdS simple qui se base sur la métrique créée. Ce paramètre aurait le même nom que la propriété du client. Si la destination de la correspondance retenue est un terme de l'ontologie de QdS, nous sommes dans le cas d'une correspondance indirecte (le cas de la Figure 31 par exemple). Nous commençons alors par récupérer la fonction qui permet de calculer la valeur de ce terme de QdS ainsi que ses opérandes (Lignes 11 et 12 de la Figure 42).

```
1.  generatePredicates()
2.    For each constraint instanceOf clientOnto:Constraint Do
3.      property:= constraint.hasProperty; operator :=constraint.hasOperator ; threshold := constraint.hasThreshold ;
4.      For each correspondence instanceOf Correspondence | correspondence.hasSourceTerm=property Do
5.        maxPropertyCorresp := getMaxCorrespCertainty(correspondance) ;
6.      if maxPropertyCorresp .hasDestination instanceOf providerTerm
7.        directive :=createMeasurementDirective(maxPropertyCorresp .hasDestinationTerm) ;
8.        metric :=createMetric(maxPropertyCorresp .hasDestinationTerm, directive) ;
9.        parameter :=createSimpleSLAParameter (property, metric) ;
10.     else if maxPropertyCorresp .hasDestinationTerm instanceOf QosTerm
11.       function:= maxPropertyCorresp .hasDestinationTerm.hasdependencyUsing ;
12.       operands:= maxPropertyCorresp .hasDestinationTerm.hasdependencyUsing.isBasedOn;
13.       metrics : = ∅ ;
14.       For each operand in Operands do
15.         For each correspondence instanceOf Correspondence | correspondence.hasSourceTerm=operand Do
16.           maxOperandCorresp := getMaxCorrespCertainty(correspondance) ;
17.         directive :=createMeasurementDirective(maxOperandCorresp.hasDestinationTerm) ;
18.         metric :=createMetric(maxOperandCorresp.hasDestinationTerm, directive) ;
19.         metrics : = metrics ∪ {metric} ;
20.       parameter :=createComposedSLAParameter(property, metrics, function) ;
21.     createViolationPredicate( parameter, operator.hasInverse, threshold) ;
```

Figure 42 – Partie Principale de l'algorithme de génération de contrat

Pour chaque opérande, nous identifions les correspondances qui ont comme source cette opérande et comme destination un terme du fournisseur (Lignes 14 et 15 de la Figure 42). Nous retenons la correspondance qui a la probabilité maximale pour chaque opérande (ligne 16 de la Figure 42). La destination de cette correspondance constitue ainsi une métrique dans le contrat généré (Lignes 17 et 18 de la Figure 42). Ensuite, nous créons un paramètre de QdS composé basé sur la fonction identifiée

et les métriques créées (Ligne 20 de la Figure 42). Enfin, pour chaque contrainte, nous générons un prédicat qui permet de vérifier si le paramètre de QdS généré respecte l'exigence du client. Ceci correspond à la fonction *createViolationPredicate* (voir ligne 21 de la Figure 42).

5. Bilan des contributions sur la négociation des accords de QdS

Dans les sections précédentes, nous avons présenté notre approche automatique d'alignement sémantique entre les intentions des clients et les offres des fournisseurs de services. Dans cette approche, nous avons commencé par définir les modèles des intentions du client et des offres du fournisseur en utilisant les ontologies. Ensuite, nous avons établi une approche complète d'alignement sémantique entre les instances de ces modèles. Notre approche se base sur quatre étapes principales : l'étape de génération de correspondances, l'étape de leur raffinement, l'étape de leur évaluation et l'étape de génération de contrat de QdS en cas de compatibilité. Nous avons développé les algorithmes nécessaires pour réaliser ces étapes d'alignement. Tous ces algorithmes sont orchestrés par l'algorithme principal présenté dans la Figure 43.

Cet algorithme consiste à générer en premier lieu les correspondances (1) entre les termes du client et les termes du fournisseur, puis, (2) entre les termes du client et les termes de l'ontologie de QdS et enfin (3) entre les termes de cette ontologie avec ceux du fournisseur (Ligne de 1 à 3 dans la Figure 43). Ensuite, cet algorithme procède à la création d'adjacences entre les correspondances afin de raffiner les erreurs isolées (Ligne 4 dans la Figure 43). Ce raffinement, est assuré par le processus de stabilisation des correspondances décrit aux lignes 5 et 6 de la Figure 43. Par la suite, notre algorithme principal passe à la vérification de la compatibilité globale entre l'intention du client et les offres du fournisseur. Cette vérification consiste à effectuer (i) une vérification structurelle en calculant la moyenne globale des correspondances et (ii) une vérification de la compatibilité des contraintes des clients avec les offres des fournisseurs (Ligne 7 et 8 de la Figure 43). En cas de compatibilité, notre algorithme génère automatiquement une version complète d'un contrat de qualité de service (Ligne 9 de la Figure 43) entre le client et le fournisseur.

Par rapport aux solutions existantes, notre approche se distingue par l'intégration de la notion d'intention dans la modélisation des besoins du client. Cette notion permet au client de s'exprimer librement sans être limité à utiliser les termes techniques du langage du fournisseur. Notre apport se manifeste aussi par l'intégration de la sémantique dans les modèles que nous avons définis en utilisant des règles d'inférence qui facilitent l'automatisation dans différentes étapes du processus d'alignement. Nous croyons que notre approche est bénéfique, d'une part, au fournisseur puisqu'elle assure l'automatisation de l'analyse des exigences des clients. D'autre part, elle peut être bénéfique au client en l'aidant à sélectionner et à comparer des services qui lui conviennent pour une même intention et de générer automatiquement une version préliminaire d'un SLA.

```
1. generateCorrespondences("ClientTerm","ProviderTerm");
2. generateCorrespondences("ClientTerm","QdSTerm");
3. generateCorrespondences("QdSTerm","ProviderTerm");
4. createCorrespondenceAdjacencies();
5. initializeCorrespondencesForStabilization();
6. correspondenceStabilization();
7. checkGlobalSemanticCompatibility();
8. checkConstraintsCompatibility();
9. generateContract();
```

Figure 43 – Algorithme principal d'alignement pour la génération de contrat

Il est à noter que les étapes de notre approche d'alignement se basent sur un ensemble de seuils qui doivent être fixés par l'expert commercial du fournisseur. Nous présentons des moyens techniques qui permettent d'aider cet expert à fixer ou même à automatiser les valeurs de ces seuils dans le chapitre 4 de ce mémoire.

Après avoir établi notre approche d'alignement sémantique entre les intentions des clients et les offres des fournisseurs qui se termine par une génération d'un contrat complet de QdS entre les parties impliquées, il est nécessaire d'assurer la surveillance des accords générés. En effet, ceci constituera une solution de valeur pour s'assurer que les parties impliquées respectent parfaitement ce qu'ils ont signé. Ainsi, quand il y a une violation ou même une dégradation des qualités requises, toutes les parties seront informées et des pénalités seront automatiquement appliquées. Nous présentons nos contributions dans la phase de surveillance des contrats de QdS dans la section suivante.

IV. APPROCHE SEMANTIQUE DE SURVEILLANCE DES ACCORDS DE QDS

Dans cette section, nous présentons nos contributions dans la phase de surveillance des accords de QdS. Notre approche de surveillance nommée *SLAOnt-Monitoring* (*SLA Ontology Monitoring*) se base sur le modèle sémantique de contrat de qualité de service *SLAOnt* que nous avons présenté dans la section 4. Ce modèle s'appuie sur l'utilisation des règles d'inférence afin de faciliter l'automatisation du processus de surveillance. Dans ce qui suit, nous présentons l'architecture que nous avons élaborée pour cette phase de surveillance.

1. Architecture de surveillance des obligations de *SLAOnt*

La Figure 44 illustre les entités principales de notre architecture de surveillance des obligations des accords de qualité de service. Cette architecture prend en entrée une instance d'un contrat de QdS, selon la structure *SLAOnt*, pour générer des services de surveillance des obligations spécifiées dans ce contrat. Les instances principales qui concernent le processus de surveillance de ces contrats sont : les métriques, les paramètres de QdS et les obligations.

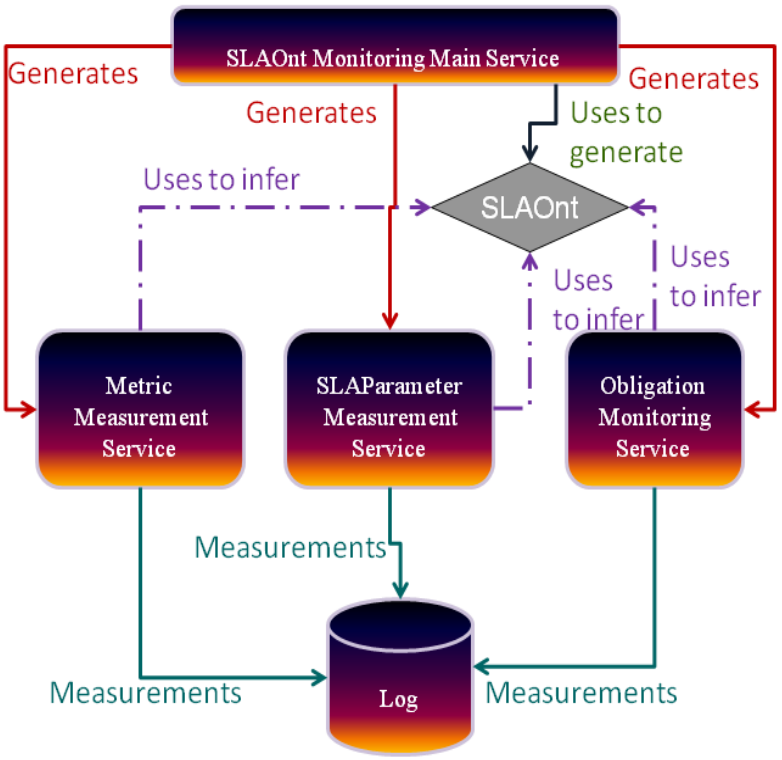

Figure 44 – Architecture de surveillance des obligations de SLAOnt

En effet, nous avons défini un service principal nommé *SLAOnt Monitoring Main Service* pour pouvoir déployer les instances de contrats de qualité de service à surveiller. Pour chaque métrique, paramètre de QdS et obligation définis dans *SLAOnt*, *SLAOnt Monitoring Main Service* instancie respectivement un service de mesure de métrique, un service de mesure du paramètre de QdS et un service de mesure des obligations et leur attribue un ordre pour leur exécution. Ces services vont utiliser les instances de *SLAOnt* pour récupérer les valeurs nécessaires à leurs exécutions. Les valeurs surveillées seront stockées dans un historique de mesures pour pouvoir effectuer des analyses statistiques ou sémantiques au cours de la phase de surveillance. En cas de violation, un message de notification est envoyé aux parties impliquées et la pénalité définie dans le contrat est automatiquement déclenchée. Dans la section suivante, nous présentons le déroulement des différentes étapes de surveillance des obligations relatives à notre architecture.

2. Etapes de surveillance des accords de qualité de service

Notre approche de surveillance est caractérisée par l'utilisation des inférences pour faciliter l'extraction des instances de l'ontologie *SLAOnt* et la génération automatique des différents services de surveillance des obligations de QdS définies dans ces instances. Ces inférences sont assurées par l'utilisation des règles SWRL tout au long de ce processus de surveillance. Ce dernier se déroule sur quatre étapes principales. Dans la première étape, nous générons automatiquement des règles SWRL

qui permettent de récupérer les mesures des différentes métriques définies dans le contrat (voir Lignes 1, 2 et 3 de la Figure 45). Un service est automatiquement généré pour chacune de ces métriques pour exécuter la règle SWRL associée et stocker les mesures obtenues dans l'historique de notre architecture de surveillance. Ces mesures sont effectuées selon les fréquences spécifiées dans le contrat. Nous rappelons que ces fréquences peuvent être des valeurs numériques comme 30 millisecondes ou une heure. Elles peuvent être aussi des notifications avant *OnCall* ou après *AfterCall* l'invocation du service associé au contrat. Dans la seconde étape, pour chaque paramètre de QdS *SLAParameter*, nous générons une règle SWRL qui appelle la fonction appropriée pour calculer la valeur du paramètre de QdS *SLAParameter*. Ce calcul se base sur les valeurs des métriques mesurées par le service généré à l'étape précédente (voir Lignes 4, 5 et 6 de la Figure 45). Un nouveau service est aussi automatiquement généré pour chaque paramètre de qualité de service. Ce service effectue le calcul du paramètre selon la fréquence spécifiée dans le contrat. Dans la troisième étape, pour chaque prédicat de SLA *SLAPredicate* défini dans les obligations du contrat, nous générons un service qui évalue le prédicat selon sa fréquence de vérification (voir Lignes 7, 8 et 9 de la Figure 45). Nous rappelons que tous les prédicats définis dans le contrat sont exprimés sous la forme de règles SWRL. La quatrième et dernière étape du processus de surveillance consiste à démarrer l'exécution de tous les services générés au cours des étapes précédentes (voir Lignes 10 et 11 de la Figure 45). Tous ces services sont enregistrés dans un annuaire spécifique selon les noms des métriques, des paramètres de QdS et des obligations correspondantes.

Chaque service de surveillance doit attendre les services de dépendances correspondants avant son exécution. Par exemple, le service de calcul du paramètre de QdS doit attendre la fin de mesure de toutes les métriques qui lui correspondent. Pour cela, nous avons défini une synchronisation entre les services de surveillance. Nous appelons synchronisation, la coordination entre plusieurs services afin de préserver leurs ordres d'exécution et éviter leurs chevauchements. Cette synchronisation est assurée au cours de fonctionnement des services de surveillance. Dans la section suivante, nous détaillons davantage le processus de fonctionnement de ces services.

```
1.  metrics := runRule("Metric( ?metric)→ query :select( ?metric)") ;
2.    For each metric in metrics do
3.      metricMeasurementService := generateMetricMeasurementService(metric) ;
4.      MeasurementServicesPool.put(metric, metricMeasurementService) ;
5.    parameters := runRule("SLAParameter( ?slaParameter)→ query :select( ?slaParameter)") ;
6.    For each slaParameter in parameters do
7.      slaParameterMeasurementService := generateSlaParameterMeasurementService(slaParameter) ;
8.      MeasurementServicesPool.put(metric, metricMeasurementService) ;
9.    obligations := runRule("Obligation( ?obligation)→ query :select( ?obligation)") ;
10.   For each obligation in obligations do
11.     obligationEvaluationService := generateObligationService(obligation) ;
12.     MeasurementServicesPool.put(obligation, obligationEvaluationService) ;
13.   For each service in MeasurementServicesPool
14.     Service.start() ;
```

Figure 45 – Algorithme principal de génération de services de surveillance

2.1 PRINCIPE DE FONCTIONNEMENT DES SERVICES DE MESURES DES METRIQUES

2.1.1 Définition et principe des services de mesure des métriques

Nous appelons service de mesure d'une métrique, l'entité logicielle qui collecte l'ensemble de mesures d'une métrique pendant une période de temps bien déterminée. Ces mesures sont assurées par l'interrogation d'une directive de mesure spécifiée dans le contrat.

2.1.2 Algorithme de mesure des métriques

Pour assurer la mesure des métriques, nous avons défini un algorithme qui constitue la première étape du processus de surveillance des obligations des accords de QdS. Le principe de base de cet algorithme consiste à instancier automatiquement un service de mesure des métriques pour chaque métrique définie dans le contrat. Cette instanciation est orchestrée par le service principal *SLAOnt Monitoring Main Service*.

Initialement, notre algorithme rassemble les valeurs des instances correspondantes à la métrique, sa directive de mesure et sa fréquence de mesure (lignes 2 et 3 de la Figure 46). Ensuite, il invoque la directive de mesure de la métrique associée selon la fréquence de mesure récupérée comme présenté dans les lignes 4 et 5 de la Figure 46. Après, il stocke les valeurs retrouvées dans un historique dénommé « Log » (Ligne 6 de la Figure 46). Ces valeurs seront utilisées dans l'étape suivante par les services de mesure du paramètre de QdS *SLA parameter*.

```
1.  MetricMeasurement(Metric metric)
2.    getMetricDetailsRule:="hasName("+ metric + ", ?metricName) ^ hasValueFrom("+metric+",?measurementDirective)^
    hasMeasurementFrequency("+ metric +",?frequency) → query:select(?metricName, ? measurementDirective, ? frequency)";
3.    (metricName, measurementDirective, metricMeasurementNotification):= runRule(getMetricDetailsRule);
4.    Every metricMeasurementNotification do
5.      measurement := invokeMeasurementDirective(measurementDirective);
6.        Log.store(metricName,measurement);
```

Figure 46 – Algorithme de mesure des métriques

2.2 PRINCIPE DE FONCTIONNEMENT DES SERVICES DE MESURES DES PARAMETRES DE QDS

2.2.1 Définition et principe des services de mesures des paramètres de QdS

Nous appelons services de mesures des paramètres de QdS *SLA parameter*, les entités logicielles qui appliquent des fonctions d'agrégations sur les valeurs générées à partir des services de mesure des

métriques. Ces fonctions sont récupérées à partir du modèle de contrat de QdS.

2.2.2 Algorithme de mesure des paramètres de QdS

Nous avons développé un algorithme de mesure des paramètres de QdS présenté dans la Figure 47. Ce dernier est automatiquement instancié par le service principal de surveillance *SLAOnt Monitoring Main Service*. Pour assurer la synchronisation des services de surveillance, le service de mesure d'un paramètre de QdS doit attendre la fin des mesures de toutes les métriques nécessaires pour son calcul (Ligne 10 de la Figure 47).

```
1.  SLAParameterMeasurement(SLAParameter parameter)
2.    getDependencyVariables := "hasFunction("+parameter+",?function) ^ hasOperand(?function, ?variable)
    →query:select(?variable);
3.    dependencyVariables := runRule(getDependencyVariables);
4.    getSLAParameterDetailsRule:= "hasFunction("+parameter+",?function) ^ hasAggregationFrequency("+ parameter + ", ?
    parameterMeasurementNotification) →query:select(?function, ? metricMeasurementNotification)";
5.     (function, parameterMeasurementNotification):=runRule(getSLAParameterDetailsRule);
6.    Every parameterMeasurementNotification do
7.       TemporaryMeasurements:= ∅;
8.       For each variable in dependencyVariables do
9.          metricMeasurementService := MeasurementServicesPool.get(variable) ;
10.         metricMeasurementService.waitFor() ;
11.         TemporaryMeasurements.put(variable, Log.getLastValues(variable));
12.         functionClass := loadFunctionClass(function) ;
13.         slaParameterValue := functionClass.call(TemporaryMeasurements);
14.      SLAOnt.setLastParameterValue(parameter, slaParameterValue);
15.         Log.store(parameter, slaParameterValue);
```

Figure 47 – Algorithme de mesure des SLA parameter

Ces métriques de dépendances sont détectées par une règle d'inférence présentée dans la ligne 2 de la Figure 47. Cette règle est exécutée à la ligne 3 de notre algorithme.

Ensuite, cet algorithme récupère la fréquence de calcul et la fonction d'agrégation du chaque paramètre de QdS. A chaque période de cette fréquence, il assure (1) la collection des mesures récupérées des métriques associées au paramètre de QdS (Lignes 8 et 9 de la Figure 47), (2) le calcul des fonctions d'agrégations de ce paramètre (Ligne 12 de la Figure 47) et (3) l'ajout des résultats dans un historique comme présenté à la ligne 15 dans la Figure 47. Ces résultats seront utilisés par la suite par les services de surveillance.

2.3 PRINCIPE DE FONCTIONNEMENT DES SERVICES DE SURVEILLANCE DES OBLIGATIONS DE QDS

2.3.1 Définition et principe des services de surveillance des obligations

Nous appelons services de surveillance des obligations du contrat de QdS, les entités logicielles qui

ont un rôle de vérification du déroulement de chaque obligation définie dans un contrat de QdS.

2.3.2 Algorithme de surveillance des obligations de QdS

Nous avons défini un algorithme (présenté dans la Figure 48) pour assurer la surveillance des obligations de QdS. Nous rappelons que ces services de surveillance sont instanciés par le service principal *SLAOnt Monitoring Main Service.*

Le service de surveillance des obligations commence par récupérer le prédicat défini dans l'obligation considérée (lignes 2 et 3 de la Figure 48). Ensuite, il obtient la règle associée au prédicat ainsi que sa période d'évaluation (lignes 4 et 5 de la Figure 48). Pour assurer la synchronisation des services de mesures, le service d'évaluation des obligations doit attendre la fin des mesures de tous les paramètres nécessaires à l'exécution de sa règle associée.

```
1.  checkObligation(Obligation obligation)
2.    getPredicateRule:="isComposedOfSLO("+obligation+",?slo) ^ hasPredicate(?slo, ?predicate) → query:select(?predicate)";
3.    predicate: = runRule(getPredicateRule) ;
4.    getPredicateRule := "hasRule("+predicate+", ?rule) ^ hasVerificationPeriodicity("+predicate+",? obligationEvalNotification) →
      query:select(?rule,? obligationEvalNotification)";
5.    (rule,obligationEvalNotification) := runRule(getPredicateRule) ;
6.    getDependencyParameters := "hasSLAParameter("+predicate+",?parameter) →query:select(?parameter)";
7.      dependencyParameters:= runRule(getDependencyParameters);
8.    Loop on obligationEvalNotification
9.      For each parameter in dependencyParameters do
10.       slaParameterMeasurementService := MeasurementServicesPool.get(parameter) ;
11.       slaParameterMeasurementService .waitFor() ;
12.     runRule(rule);
```

Figure 48 – Algorithme de surveillance des obligations

Ces paramètres sont détectés par une autre règle d'inférence présentée à la ligne 6 de la Figure 48.

A chaque notification d'évaluation du prédicat (ligne 8 de la Figure 48), le service de surveillance des obligations attend la fin des mesures des paramètres de QdS dont il dépend (lignes 9 à 11 dans Figure 48) avant l'exécution de la règle SWRL associée à ce prédicat et déclenche l'action à prendre en cas de violation. Par exemple, dans la règle SWRL de la Figure 49, nous indiquons que si le temps de réponse est supérieur à 100 millisecondes, l'action *disseminateViolation* est déclenchée automatiquement. Cette action transmet toutes les violations détectées et leurs causes aux parties impliquées dans le SLA. Les pénalités définies dans le contrat seront aussi automatiquement déclenchées.

```
hasEvaluation(average_response_time, ?x) ^  swrlb: greaterThanOrEqual(?x, 100.0) →
slaActions:disseminateViolation(AverageLessThan100ms, «false», average_ response_time,
?x)^slaActions:applyPenalty(TenPerCentDiscount)
```

Figure 49 – Règle d'évaluation de prédicat

3. Bilan des contributions sur la surveillance des accords de QdS

Dans les sections précédentes, nous avons présenté nos contributions dans le domaine de surveillance des contrats de qualité de service. Ce domaine constitue une autre étape du cycle de vie de ces contrats. En raison de l'absence d'une approche complète, automatique et sémantique de surveillance de SLA, nous avons proposé une solution orientée services pour remédier à cette insuffisance. Notre approche de surveillance se base principalement sur le modèle sémantique de contrat de qualité de service *SLAOnt* que nous avons présenté dans la section 4. Dans cette approche, nous commençons par extraire les instances principales de ces contrats. Par la suite, nous procédons à générer les trois services principaux : services de mesure des métriques, services de mesure des paramètres de QdS et services de mesure des obligations pour réaliser le processus de surveillance. Le rôle de ces services est de vérifier le respect des obligations d'une manière continue pendant toute la période de validité du contrat.

V. CONCLUSION

Dans ce chapitre, nous avons commencé par présenter notre modèle d'accord de qualité de service *SLAOnt* qui se distingue par sa complétude et sa richesse sémantique par rapport aux modèles existants notamment au niveau de la composition des paramètres de QdS. Puis, nous avons détaillé nos approches de négociation et de surveillance des accords de qualité de service. L'approche de négociation se compose de quatre étapes principales. La première étape consiste à générer les correspondances entre les termes du client et ceux du fournisseur. Le but de cette étape est de trouver les similarités possibles entre les termes en leur affectant des degrés de similarités. La deuxième étape consiste à raffiner et stabiliser ces valeurs dans le but de réduire les erreurs de mesures de similarité. La troisième étape est une étape d'évaluation d'alignement dans laquelle nous commençons par vérifier que les deux dernières phases ont produit un nombre suffisant de correspondances puis nous vérifions si les contraintes des clients sont satisfaites par les offres du fournisseur en se basant sur les correspondances générées. La quatrième étape consiste à générer automatiquement un contrat de qualité de service entre le client et le fournisseur en cas de compatibilité. Le contrat généré est une instance de la structure sémantique *SLAOnt* que nous avons élaborée dans ce travail de thèse.

Dans une deuxième phase de cette thèse, nous avons proposé une approche sémantique et automatique de surveillance des obligations de contrats de qualité de service générés par la phase de négociation. Cette approche prend en entrée l'instance de *SLAOnt* représentant le contrat à surveiller. Ensuite, elle génère les services nécessaires de mesure des métriques élémentaires de QdS, mesure des paramètres de haut niveau de QdS et évaluation des obligations définies dans le contrat. Le rôle de ces services est de détecter les dégradations afin de notifier les parties concernées et d'appliquer automatiquement les

pénalités en cas de violation.

Nos approches de négociation et de surveillance des contrats de QdS exploitent la puissance d'inférence logiques liées aux structures des ontologies ce qui nous a permis d'obtenir une haute précision et un haut degré d'automatisation de ces deux phases.

Nos contributions bénéficient aussi des avantages des approches existantes de négociation et de surveillance des accords de QdS. En effet, elles réutilisent le principe des approches (1) basées-linguistique puisqu'elles utilisent une ontologie lexicale *WordNet* [104], (2) basées-raisonnement puisque nous avons eu toujours recours à utiliser des règles d'inférence, (3) basées-heuristique puisque nous utilisons une formule heuristique pour raffiner les correspondances trouvées entre les termes du client et ceux du fournisseur et finalement (4) basées-probabilité étant donné que nous utilisons des méthodes de mesure de similarité qui représentent des probabilités d'existence de ces correspondances. Notre approche est aussi basée-sémantique puisque nous utilisons des modèles sémantiques (ontologies) pendant les phases de négociation et de surveillance des accords de QdS. Dans le chapitre suivant, nous détaillons la mise en œuvre de ces contributions.

CHAPITRE IV – CONTRIBUTIONS : IMPLANTATION ET EVALUATION DE NOS APPROCHES DE NEGOCIATION ET DE SURVEILLANCE DES ACCORDS DE QDS

SOMMAIRE DU CHAPITRE

I. INTRODUCTION

Dans ce chapitre, nous présentons les outils que nous avons développés pour la validation de nos approches de négociation et de surveillance de contrats de QdS. Nous commençons par énumérer les outils techniques que nous avons utilisés pour l'implantation de nos contributions théoriques. Nous passons ensuite aux architectures techniques que nous avons choisies pour la mise en œuvre de nos outils. Nous dédions la section IV de ce chapitre aux expérimentations que nous avons effectuées avec les outils réalisées.

II. CHOIX D'IMPLANTATION

1. Langage de programmation

Pour accéder aux différentes ontologies de notre travail, nous avons opté pour Java comme langage de programmation. En effet, Java est un langage à usage général, évolué et orienté objet. C'est un langage développé par Sun. Java possède plusieurs points forts tel que : la simplicité et la sûreté, l'utilisation de la notion de threads qui le rend multitâche et l'indépendance de toute plateforme. Nous avons également opté pour ce Langage vu l'ensemble de ses bibliothèques disponibles pour manipuler les ontologies.

2. Langages d'implantation des ontologies

Pour implanter nos ontologies représentant les intentions des clients, les offres des fournisseurs et les contrats de qualité de service, nous avons opté pour le langage OWL [70]. Un des éléments clés d'OWL est sa capacité de décrire les classes d'une façon plus intéressante et plus complète que le RDF et le RDFS. En plus, grâce à son mécanisme d'importation, il permet d'incorporer plusieurs ontologies dans une nouvelle ontologie accélérant ainsi le processus de développement des ontologies. En conséquence, il est impératif d'utiliser le langage OWL pour pouvoir bénéficier de tous les avantages et de toute la puissance sémantique des ontologies. Cependant, pour pouvoir faire des inférences sur les connaissances décrites en OWL, ce langage doit être étendu par un autre langage dérivé qui est SWRL. Ce dernier permet de définir des règles d'inférence pour interroger les connaissances exprimées en OWL dans l'ontologie. Il permet aussi de produire de nouvelles connaissances inférées à partir de celles qui existent déjà dans l'ontologie. L'une des plus grandes motivations qui nous a poussées à utiliser le couple OWL/SWRL pour l'implantation des ontologies est l'extensibilité du langage SWRL par des fonctions personnalisées externes à l'ontologie. En effet, une règle SWRL peut appeler une fonction externe définie en tant que méthode Java dans une classe dédiée. Ces fonctions sont connues sous le nom *built-ins*. Nous donnons une idée plus détaillée sur ces *built-ins* dans la section suivante.

3. Bibliothèques Java de gestion des ontologies

Protégé est un éditeur d'ontologies *open source* distribué par l'université d'informatique médicale de Stanford. Il offre tous les outils essentiels pour construire, manipuler, visualiser et sauvegarder une ontologie dans différents formats (*RDF*, *RDFS*, *OWL*…). Dans le contexte du Web sémantique, des *plugins* ont été développés pour ajouter des fonctions dédiées à Protégé. Les principaux *plugins* que nous avons utilisés sont *OntoViz* pour la visualisation des ontologies et *SwrlTab* pour l'édition des règles SWRL.

Protégé est également une librairie Java qui peut être étendue pour créer des applications indépendantes qui accèdent en lecture et en écriture aux ontologies créées. Il s'appuie sur la notion de pont *Bridge* qui permet d'échanger les informations stockées dans l'ontologie avec des moteurs d'inférence externes pour effectuer des raisonnements et pour déduire de nouvelles connaissances en exécutant les règles SWRL liées aux instances de l'ontologie. Parmi les moteurs existants, nous avons choisi JESS [105] vu sa facilité d'intégration dans l'éditeur Protégé et dans des applications Java indépendantes. Ce moteur d'inférence efficace est entièrement écrit en Java avec une documentation et des forums de discussions en ligne pour faciliter son utilisation. Il permet aussi d'invoquer directement à partir des règles SWRL, les *built-ins* sous la forme de méthodes Java définies dans une classe spéciale nommée *SwrlBuiltinLibrary*. Il est à noter que le langage SWRL offre déjà quelques *built-ins* prédéfinies pour la comparaison de valeurs numériques et pour la manipulation des chaînes de caractères. Ces *built-ins* peuvent être étendues par d'autres fonctions personnalisées en les déclarants dans une ontologie externe qu'il faut importer avant de les utiliser. Il faut aussi déclarer la classe qui implante ces fonctions dans le *classpath* pour que le moteur d'inférence puisse les trouver.

4. Mesures de similarités sémantiques et de proximité linguistiques

4.1 WORDNET::SIMILARITY

Pour pouvoir évaluer les correspondances entre les termes de l'intention du client et ceux des offres du fournisseur, nous nous sommes basés sur des mesures de similarité sémantique. Ces similarités sémantiques représentent la probabilité d'existence des correspondances détectées par notre approche d'alignement. Pour avoir ces mesures, nous avons utilisé *WordNet::Similarity* [103]. *WordNet::Similarity* est un logiciel gratuit qui permet de mesurer la similitude sémantique et la proximité linguistique entre deux termes en se basant sur leur sens ou leur hiérarchie dans l'ontologie *WordNet* [104]. Il est l'outil actuel le plus complet fournissant six mesures de similarité sémantique et trois mesures de proximité linguistique qui sont toutes basées sur les données lexicales de *WordNet*. Ces mesures sont mises en œuvre avec le langage *Perl* [106]. Les modules *Perl* prennent en entrée deux termes et retournent comme sortie une valeur numérique représentant le degré de similarité

sémantique ou de proximité linguistique entre ces deux termes. Nous rappelons que nous avons utilisé les mesures de similarité sémantique pour la création des correspondances entre les termes de l'intention du client et ceux du fournisseur. En plus, nous avons utilisé les mesures de proximité linguistique pour affecter des pondérations aux adjacences utilisées dans la phase de stabilisation des similarités sémantiques des correspondances.

4.2 CHOIX DE LA MESURE DE PROXIMITE LINGUISTIQUE

Les mesures de proximité linguistique de *WordNet::Similarity* sont basées sur l'analyse de la hiérarchie *WordNet* « is a » (« est un »). Ces mesures sont divisées en deux catégories :

- Similarités basées sur la longueur du chemin entre une paire de concepts telle que : WuP [107], Lch [108] et Path [109].
- Similarités basées sur le contenu de l'information [110] telles que : JCN [111], Lin [112] et Res [113].

Nous avons opté pour la mesure *Path* puisqu'elle permet d'évaluer la proximité linguistique des termes en se basant sur leurs distances par rapport à un nœud représentant la racine commune de ces termes dans la hiérarchie *WordNet*.

4.3 CHOIX DE LA MESURE DE SIMILARITE SEMANTIQUE

Les mesures de similarité sémantique de *WordNet::Similarity* sont plus raffinées que les mesures de proximité linguistique. Ils ne sont pas basées uniquement sur la relation *is a* (« est un ») de *WordNet*, mais ils utilisent d'autres relations non hiérarchiques tel que : *has-part, made-of* … Les trois mesures de similarité sémantique qu'offre *WordNet::Similarity* sont : *hso* [114], *Lesk* [115] et *Vector* [116]. La mesure *Vector* crée une matrice de cooccurrence de chaque mot utilisé dans les sens des mots *gloss* de *WordNet*. Cette matrice est réutilisée pour représenter chaque terme donné par un vecteur qui est la moyenne de ces occurrences dans la matrice. La similarité sémantique entre deux mots est calculée par le cosinus de leurs deux vecteurs. Nous avons opté pour cette mesure car elle offre la meilleure précision par rapport aux deux autres comme prouvé par Steffen & al [117]. Elle se distingue par l'utilisation des sens des mots *gloss* pour évaluer l'équivalence sémantique des termes.

4.4 WORDNET

WordNet est un réseau sémantique de bases de données lexicales disponible gratuitement et développé en langue anglaise à l'Université de Princeton. La caractéristique la plus importante de *WordNet* est sa capacité de répertorier, classifier et mettre en relation de diverses manières le contenu sémantique et lexical de la langue anglaise en se basant sur les sens des informations. Chaque nœud de la base lexicale de *WordNet* présente un concept du monde réel et son lexique est séparé en quatre grandes catégories : les noms, les verbes, les adjectifs et les adverbes. *WordNet* est construit sous la forme

d'une hiérarchie de concepts appelés *synsets* (ensemble de synonymes désignant un même sens ou un usage particulier) qui sont liés entre eux par différents types de relations sémantiques (hypéronymie, hyponymie, antonymie…).

5. Couches logicielles utilisées

Nos outils de négociation et de surveillance d'accords de QdS se basent sur un modèle à cinq couches (voir Figure 50) qui réutilisent les différents outils techniques que nous avons présentés dans les sections précédentes de ce chapitre. Chaque couche est nécessaire pour le fonctionnement des couches qui lui sont supérieures. Dans la couche 1, nous retrouvons les différentes ontologies que nous avons utilisées dans notre approche. Pour pouvoir accéder aux connaissances définies dans ces ontologies, nous avons utilisé l'API Xerces XML dans la couche 2. Cependant, l'API Xerces XML permet seulement de lire des balises XML.

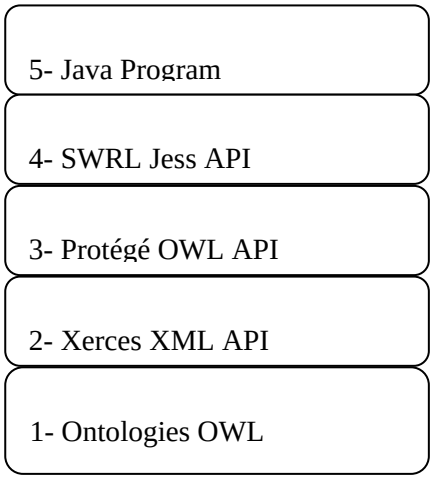

Figure 50 – Couches logicielles utilisées pour l'implantation technique de nos contributions

Pour pouvoir transformer ces balises en composants OWL (Concepts, relations, instances, etc), nous avons utilisé l'API Protégé OWL (couche 3). Cette API est utilisée par la couche 4 contenant le moteur SWRL Jess qui permet de réaliser des requêtes et des inférences sur les connaissances présentes dans les ontologies. Enfin la couche 5 contient les classes Java que nous avons développées pour réaliser nos outils de négociation et de surveillance de contrats de QdS. Nous donnons une idée plus précise sur le fonctionnement de ces classes dans la section suivante.

III. CONCEPTION DE NOS OUTILS DE NEGOCIATION ET DE SURVEILLANCE D'ACCORDS DE QDS

1. Conception d'outil de négociation d'accords de QdS

1.1 FONCTIONS OFFERTES AUX UTILISATEURS

Dans cette section, nous commençons par une identification des différents acteurs de l'outil d'alignement des intentions des clients avec les offres des fournisseurs. Ensuite, nous associons les différents cas d'utilisation avec chacun de ces acteurs. Nous finissons cette section par le diagramme de cas d'utilisation de notre outil.

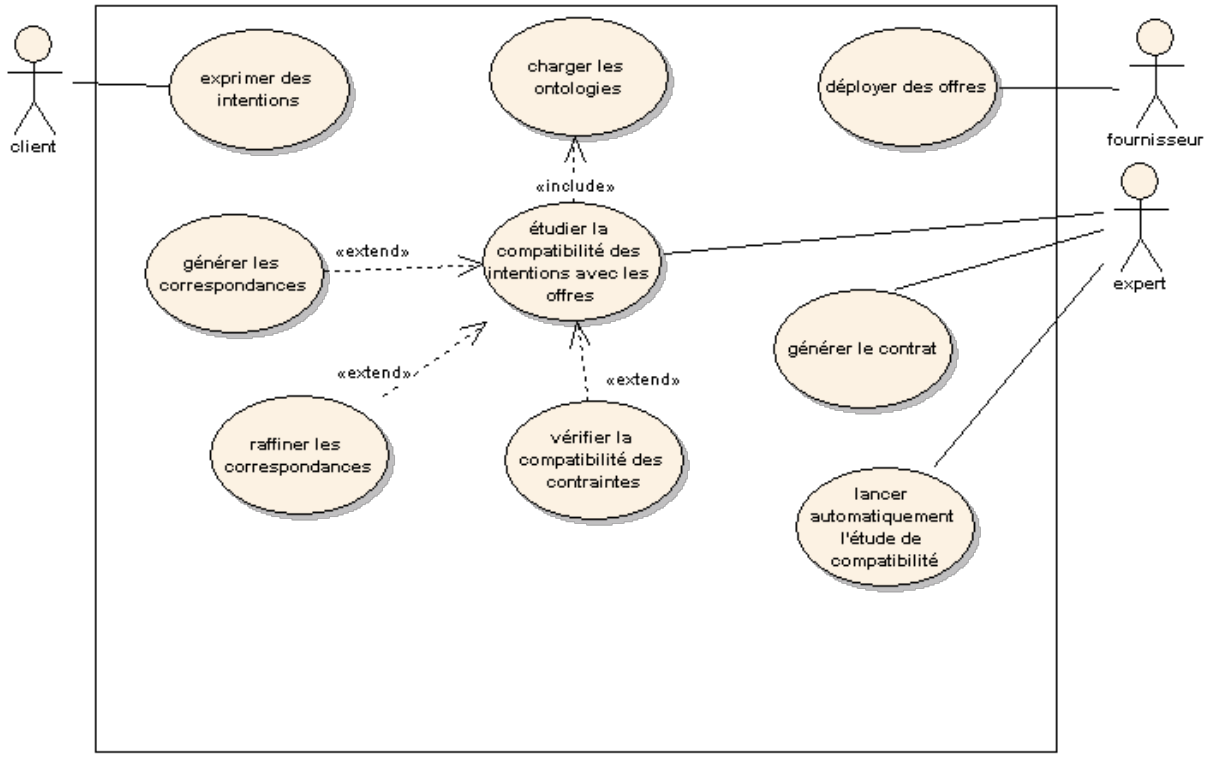

Figure 51 – Diagramme des cas d'utilisation de notre outil d'alignement

Dans notre outil, nous distinguons trois acteurs principaux : le client, le fournisseur et l'expert commercial du fournisseur. Avec notre outil, le client pourrait exprimer ses intentions, le fournisseur pourrait déployer ses offres et enfin l'expert commercial pourrait analyser les correspondances des intentions des clients avec les offres du fournisseur. De ce cas d'utilisation dérivent plusieurs cas d'utilisation qui représentent les étapes d'analyse de l'intention du client par rapport aux offres disponibles. Ces cas d'utilisation sont : « générer les correspondances », « raffiner les correspondances », « évaluer les correspondances » et « générer le contrat » en cas de compatibilité. Il est à noter aussi que l'expert commercial pourrait avoir le choix d'effectuer ces étapes d'un seul coup

ou de les exécuter une à une. La Figure 51 présente le diagramme des cas d'utilisation offerts par notre outil à ses utilisateurs.

Dans ce qui suit, nous présentons les diagrammes de séquences relatifs à ces cas d'utilisation.

1.2 GENERATION DE CORRESPONDANCES SEMANTIQUES

1.2.1 Génération des correspondances directes

Pour garantir la généricité de notre travail, nous avons défini un concept principal pour chaque ontologie nommé *RootConcept* comme présenté dans la section III.1.2 du chapitre 3 de ce mémoire. Tous les concepts de l'ontologie étendent ce super concept. Pour générer les correspondances entre les termes du client et les termes du fournisseur, nous avons défini une méthode nommée *generateCorrespondances(RootConcept1,RootConcept2)*. Cette méthode permet de créer une règle SWRL nommée *Concept1Concept2CorrespondanceRule*. *RootConcept1* et *RootConcept2* désignent respectivement le super concept de la première ontologie et celui de la deuxième ontologie. Dans notre cas, la première ontologie est celle du client *ClientOnto* et la deuxième est celle du fournisseur *ProviderOnto*. La règle créée par la méthode *generateCorrespondences* est exécutée par le moteur d'inférence grâce à sa méthode *infer()*. L'exécution de cette règle consiste à parcourir toutes les paires possibles composées d'un terme de l'ontologie du client *term1* et d'un terme de l'ontologie du fournisseur *term2*.

Pour chaque paire :

- Le moteur d'inférence invoque une méthode *similarity* définie dans une classe *SWRLBuiltInLibraryImpl*. Cette méthode interagit avec l'outil *WordNet::Similarity* pour obtenir la similarité sémantique entre *term1* et *term2*.
- La méthode *createCorrespondence(?term1, ?term2, ?correspondence)* assure la création d'une nouvelle instance du concept *correspondence* ayant comme source le premier terme du couple *term1* et comme destination le deuxième terme du couple *term2*.
- La méthode *hasCertainty(?correspondence,?correspondenceCertainty)* affecte la valeur de similarité sémantique entre les deux termes.

La Figure 52 présente le diagramme de séquence de l'étape de génération des correspondances.

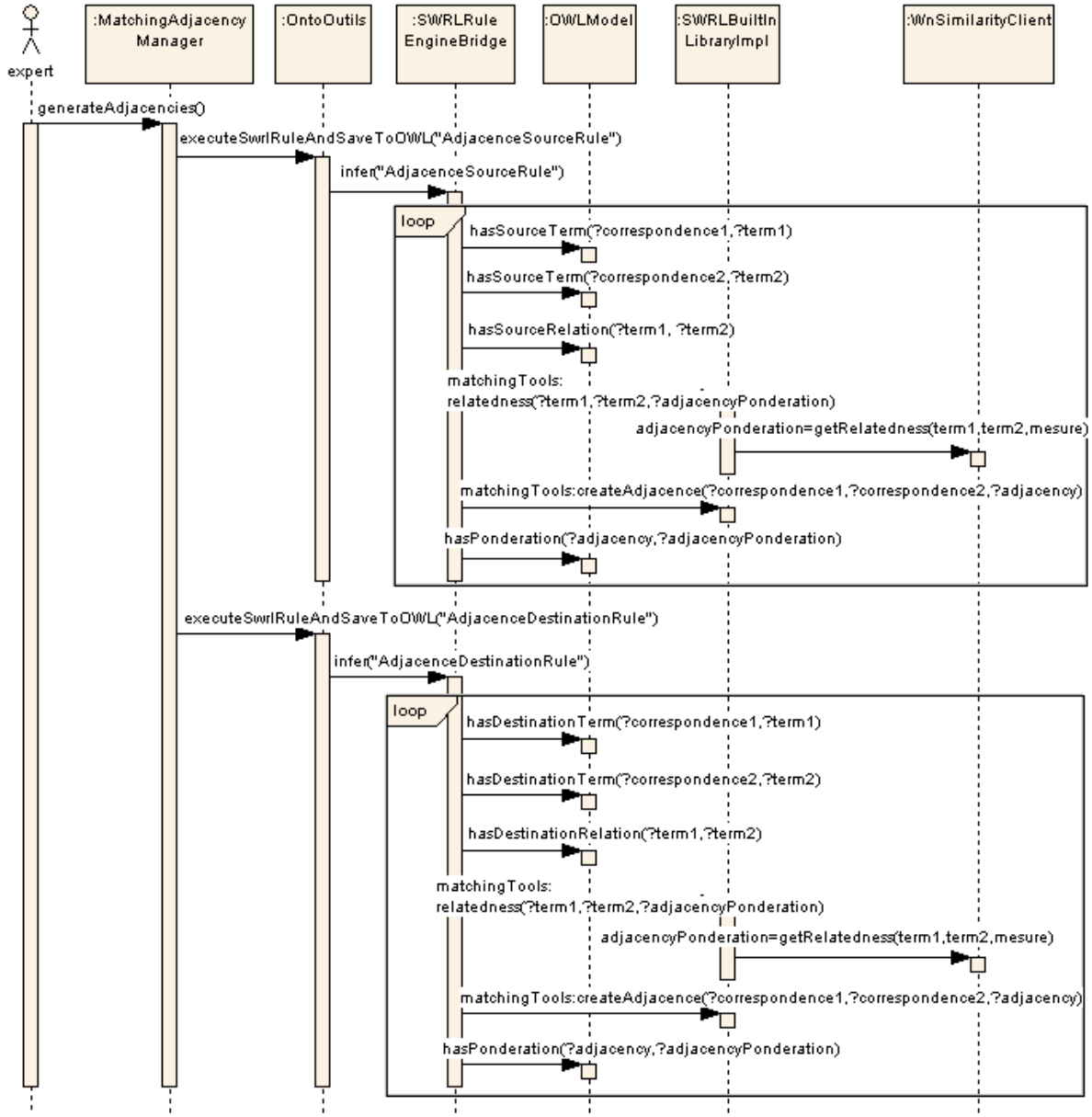

Figure 52 – Diagramme de séquence du cas d'utilisation "générer les correspondances"

1.2.2 Génération des correspondances indirectes

Cette étape s'intéresse à la recherche des correspondances entre l'ontologie du client et l'ontologie de qualité de service, puis les correspondances entre l'ontologie de qualité de service et l'ontologie du fournisseur. Dans cette étape, nous réutilisons le même principe de la recherche de correspondances directes étudiées dans la section précédente en l'appliquant sur les termes de l'ontologie du client et ceux de l'ontologie de qualité de service dans un premier lieu. Puis, entre les termes de l'ontologie de qualité de service et l'ontologie du fournisseur dans un deuxième lieu.

1.3 RAFFINEMENT DES CORRESPONDANCES

1.3.1 Génération des adjacences

Pour implanter l'algorithme de génération des adjacences, nous nous sommes basés sur deux règles SWRL principales : *AdjacencySourceRule* et *AdjacencyDestinationRule*. Le principe de ces règles consiste à parcourir tous les couples de correspondances possibles générées dans l'étape précédente. Pour chaque couple de correspondance *(Ci,Cj)* :

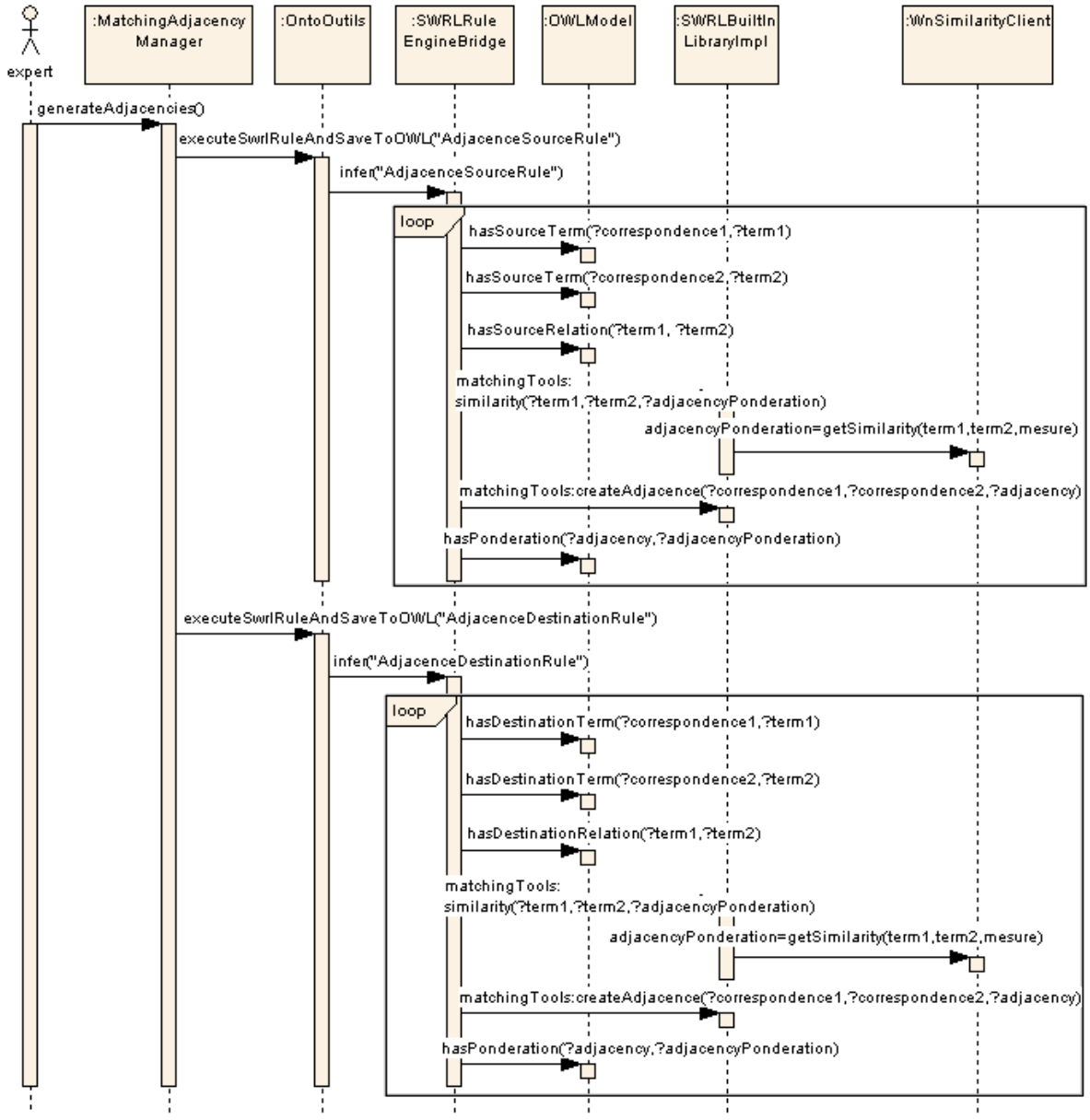

Figure 53 – Diagramme de séquence du cas d'utilisation "générer les adjacences"

- - La méthode *hasSourceRelation(?sTerm1, ?sTerm2)* permet de vérifier s'il existe une relation entre la source de la correspondance *Ci* et la source de la correspondance *Cj*. La méthode

hasDestinationRelation(?dTerm1, dTerm2) vérifie l'existence d'une relation entre les termes destination de ces correspondances.

En cas d'existence d'une relation, le moteur d'inférence appelle une méthode *relatedness* définie dans une classe *SWRLBuiltInLibraryImpl*. Cette méthode interagit avec un module externe qui permet de calculer la proximité linguistique entre *sTerm1* et *sTerm2* d'une part et *dTerm1* et *dTerm2* d'autre part.

- La méthode *createAdjacency(?correspondance1, ?correspondance2, ?adjacency)* assure la création d'une nouvelle instance du concept *adjacency* ayant comme source la première correspondance et comme destination la deuxième correspondance.
- la méthode *hasPonderation(?adjacency, ?adjacencyPonderation)* affecte la moyenne des proximités linguistiques entre les termes sources et les termes destinations des correspondances considérées à la nouvelle adjacence créée.

La Figure 53 présente le diagramme de séquence de l'étape de génération des adjacences.

1.3.2 Stabilisation des correspondances

L'étape de stabilisation des correspondances consiste à corriger les erreurs isolées des mesures de similarité sémantique. En effet, puisqu'un terme peut avoir plusieurs significations, quelques mesures erronées de similarité sémantique peuvent exister. Pour corriger ce type d'erreurs, nous procédons à une propagation des valeurs des similarités des correspondances en utilisant les adjacences générées entre elles.

Dans un premier lieu, nous commençons par initialiser les correspondances à « non stabilisées » sauf celles qui ont une mesure de similarité égale à 1. Dans ce cas, nous considérons que la correspondance est déjà stabilisée puisqu'elle est une évidence avec une probabilité d'existence égale à 1. Cette initialisation est réalisée par une règle SWRL que nous avons nommée *InitializeEvidenceCorrespondence*. Si le nombre de correspondances non stabilisées (nb sur Figure 54) est différent de 0, nous exécutons deux règles SWRL *FloodingSource* et *FloodingDestination* qui permettent de récupérer toutes les variables nécessaires pour calculer la nouvelle probabilité d'existence de chaque correspondance non stabilisée grâce à la règle SWRL *calculateNewCertainty*. La correspondance est marquée comme stabilisée s'il n'y a pas de différence significative entre sa nouvelle et son ancienne probabilité d'existence. Nous réitérons l'exécution de ces règles SWRL jusqu'à ce qu'il n'y a plus de correspondances non stabilisées (nb dans la Figure 54 devient 0). La Figure 54 présente le diagramme de séquence qui illustre le processus de stabilisation.

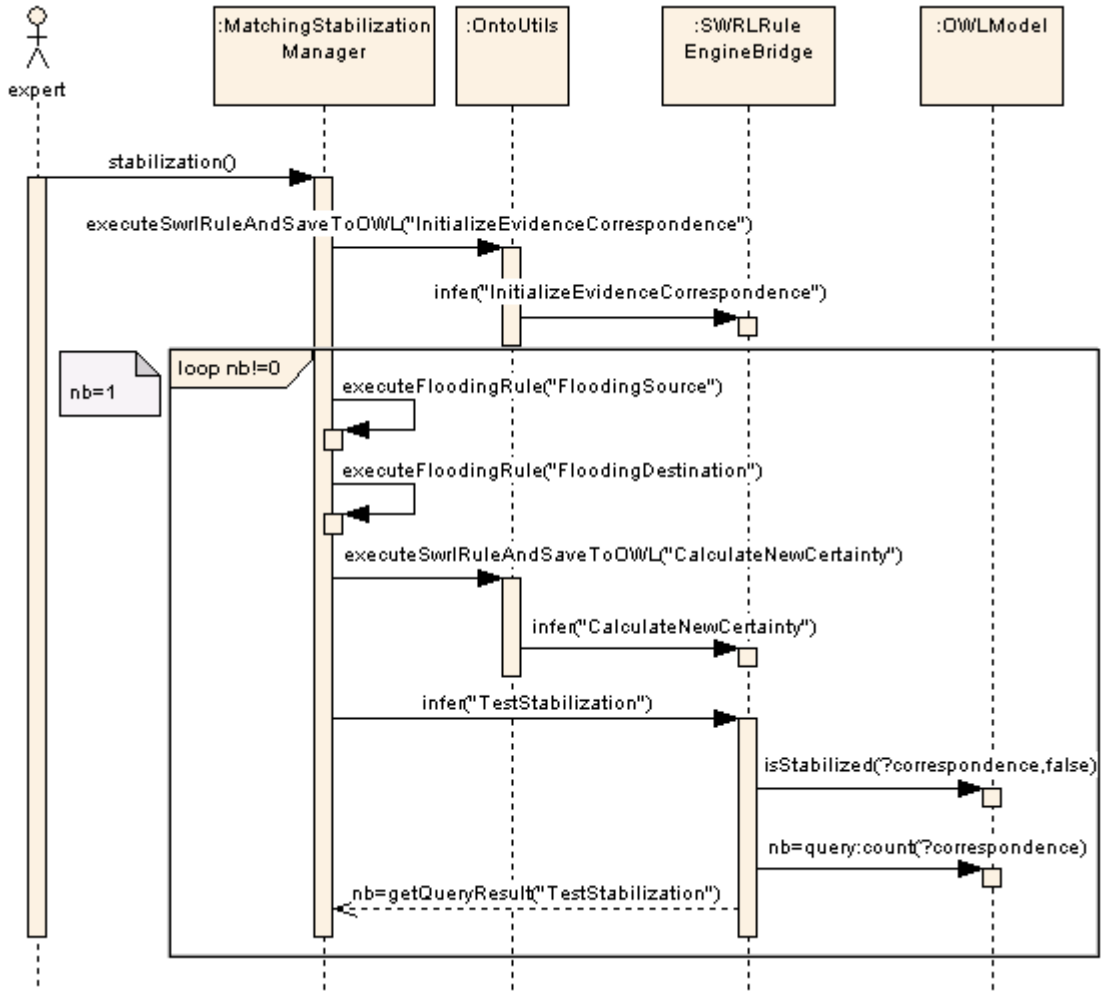

Figure 54 – Diagramme de séquence du cas d'utilisation "stabiliser les correspondances"

1.4 VERIFICATION DE COMPATIBILITE

L'évaluation des contraintes constitue l'étape la plus importante de notre travail. Elle permet de vérifier la compatibilité de toutes les contraintes imposées par le client avec les offres du fournisseur. Cette étape s'intéresse particulièrement au concept *Constraint* de l'ontologie du client.

Les étapes de cet algorithme sont :

1. La méthode *getProperties()* permet de récupérer les propriétés *clientProperty* de toutes les contraintes *Constraint* du client ainsi que leurs seuils *threshold* et leurs opérateurs *Operator* par l'exécution de la règle SWRL *getPropertiesParams* (Figure 55).

2. Traitement des propriétés qui ont une correspondance directe ou indirecte avec les variables du fournisseur :

- La règle *PropertyClientProvider* permet de récupérer toutes les variables de l'ontologie du fournisseur qui ont une correspondance sémantique directe avec les propriétés du client. Cette règle renvoie comme résultat tous les couples *(ClientProperty, ProviderVariable)*.

- La règle *PropertyQosProvider* permet de récupérer toutes les variables de l'ontologie du fournisseur qui ont une correspondance directe avec les opérandes (*QoSOperand*) représentées par le concept Parameter dans l'ontologie de qualité de service. Cette règle renvoie comme résultat tous les couples *(QosOperand, ProviderVariable)*.

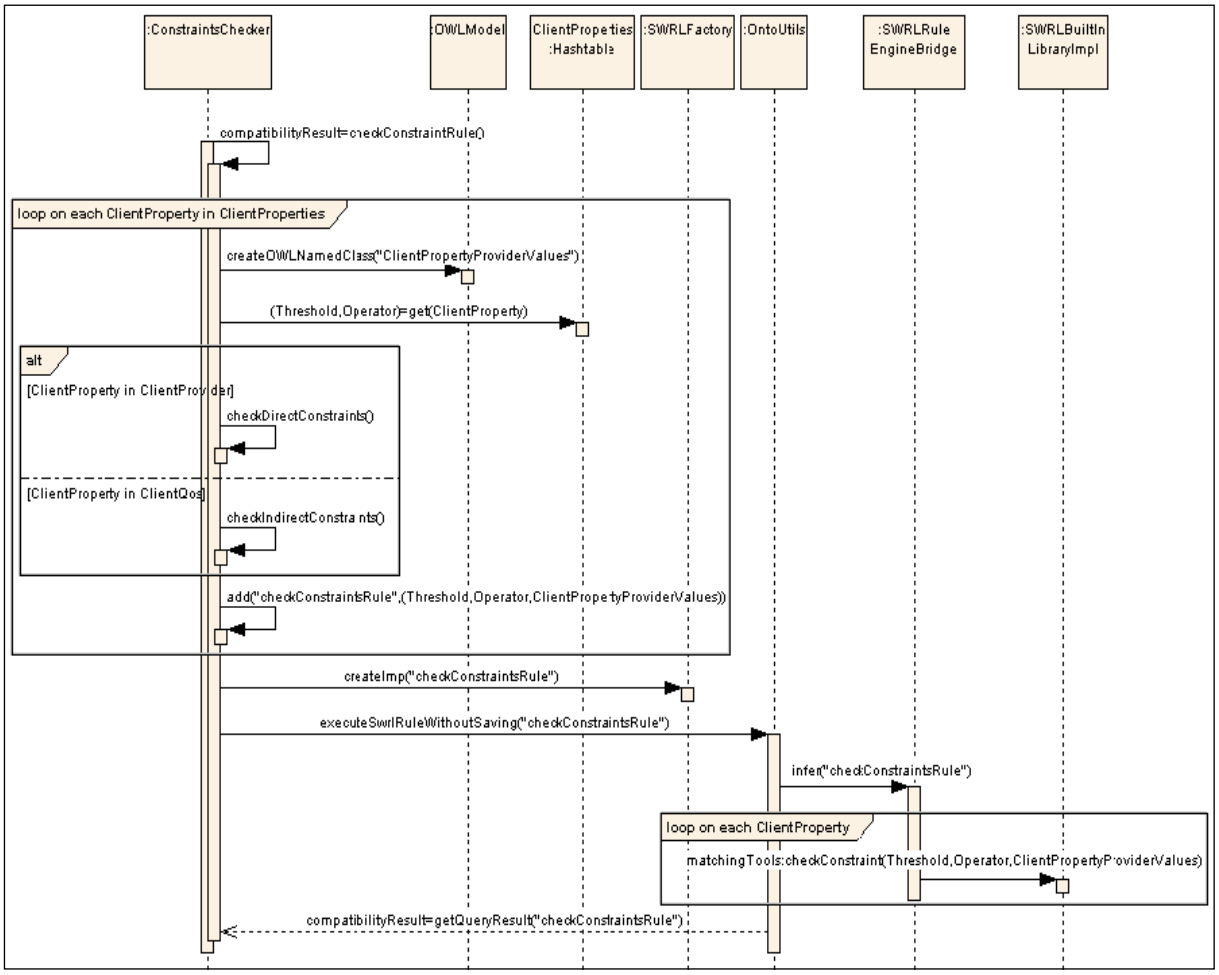

Figure 55 – Diagramme de séquence de la méthode "*checkConstraintRule*"

- La règle *PropertyClientQos* permet de récupérer tous les concepts *QosMetric* de l'ontologie de qualité de service qui ont une correspondance directe avec les propriétés de l'ontologie du client. Cette règle renvoie comme résultat tous les couples *(ClientProperty, QosMetric)*.

Après l'exécution de cette dernière règle, et après avoir récupéré tous les couples possibles *(ClientProperty, QosMetric)*, l'étape suivante sera la vérification des opérandes de chaque *QosMetric* trouvée.

Ainsi, pour chaque *QosMetric* il faut :

* Créer et exécuter une règle nommée *QosMetricOperandsRule*. Cette règle permet de récupérer tous les opérandes (*QosOperands*) qui permettent de calculer la valeur de cette *QosMetric*.

* Vérifier si tous ces opérandes existent dans les couples (*QosOperand, ProviderVariable*) récupérés précédemment.

* Ne conserver que les couples *(ClientProperty, QosMetric)* qui vérifient cette dernière condition.

3. La méthode *checkConstraintsRule* permet de vérifier la validité de toutes les contraintes du client.

La méthode *CheckConstraintRule* détaillée par le diagramme de séquence présentée dans la Figure 55, se déroule selon les étapes suivantes :

3.a Pour chaque propriété du client *ClientProperty* récupérée dans l'étape 1 :

* La méthode *createOWLNamedClass("ClientPropertyProviderValues")* crée un nouveau concept ayant comme nom *ClientPropertyProviderValues* qui est destiné à contenir toutes les valeurs correspondantes à cette propriété chez le fournisseur.

* La méthode *get(ClientProperty)* permet de récupérer pour chaque *ClientProperty* son opérateur *Operator* et son seuil *threshold* pour pouvoir comparer les valeurs des seuils avec celles récupérées à partir de l'ontologie du fournisseur.

3.a.1 Si *ClientProperty* appartient aux couples *(ClientProperty, ProviderVariable)*, la méthode *checkDirectConstraints* (présentée dans le diagramme de séquence de la Figure 56) crée une règle *ProviderVariableValuesRule* dont son exécution permet de copier toutes les valeurs du concept *ProviderVariable* correspondant, dans les variables du nouveau concept *ClientPropertyProviderValues* approprié à cette propriété du client.

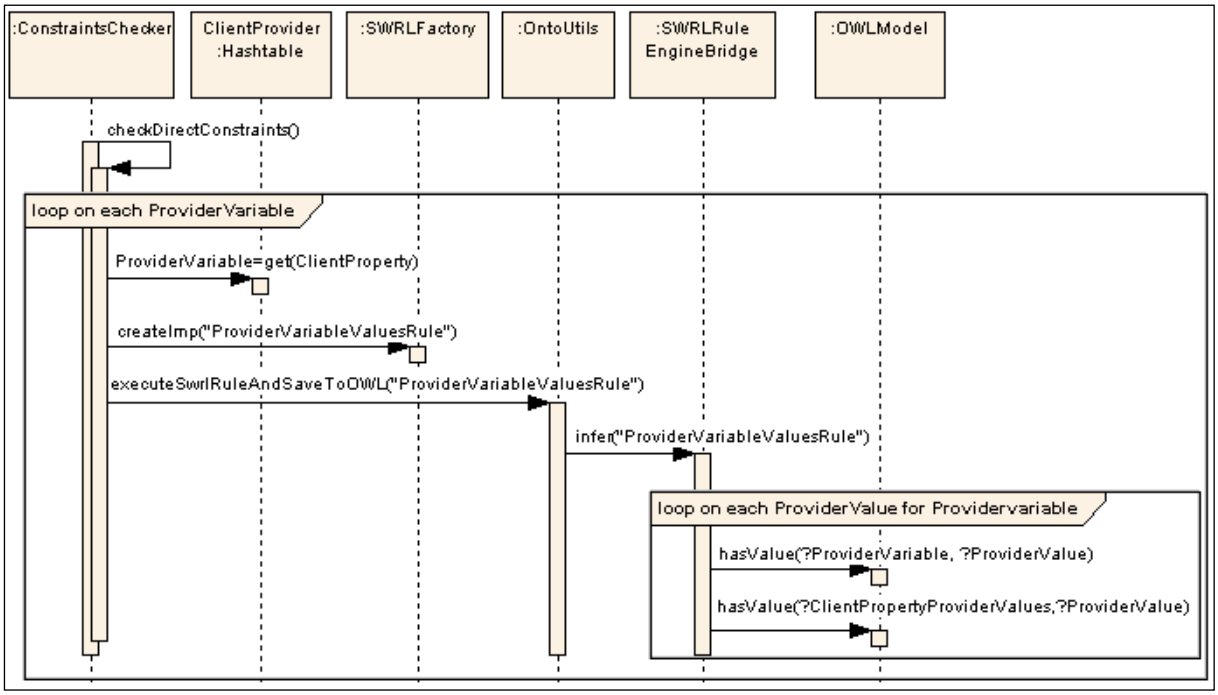

Figure 56 – Diagramme de séquence de la méthode "*checkDirectConstraints*"

3.a.2 Si *ClientProperty* appartient aux couples *(ClientProperty, QosMetric)*, l'exécution de la règle *QosMetricOperandsRule* créée dans l'étape 2 permet de récupérer toutes les opérandes de cette

variable. Ces opérandes vont servir à calculer la fonction permettant de calculer la valeur de *QosMetric*.

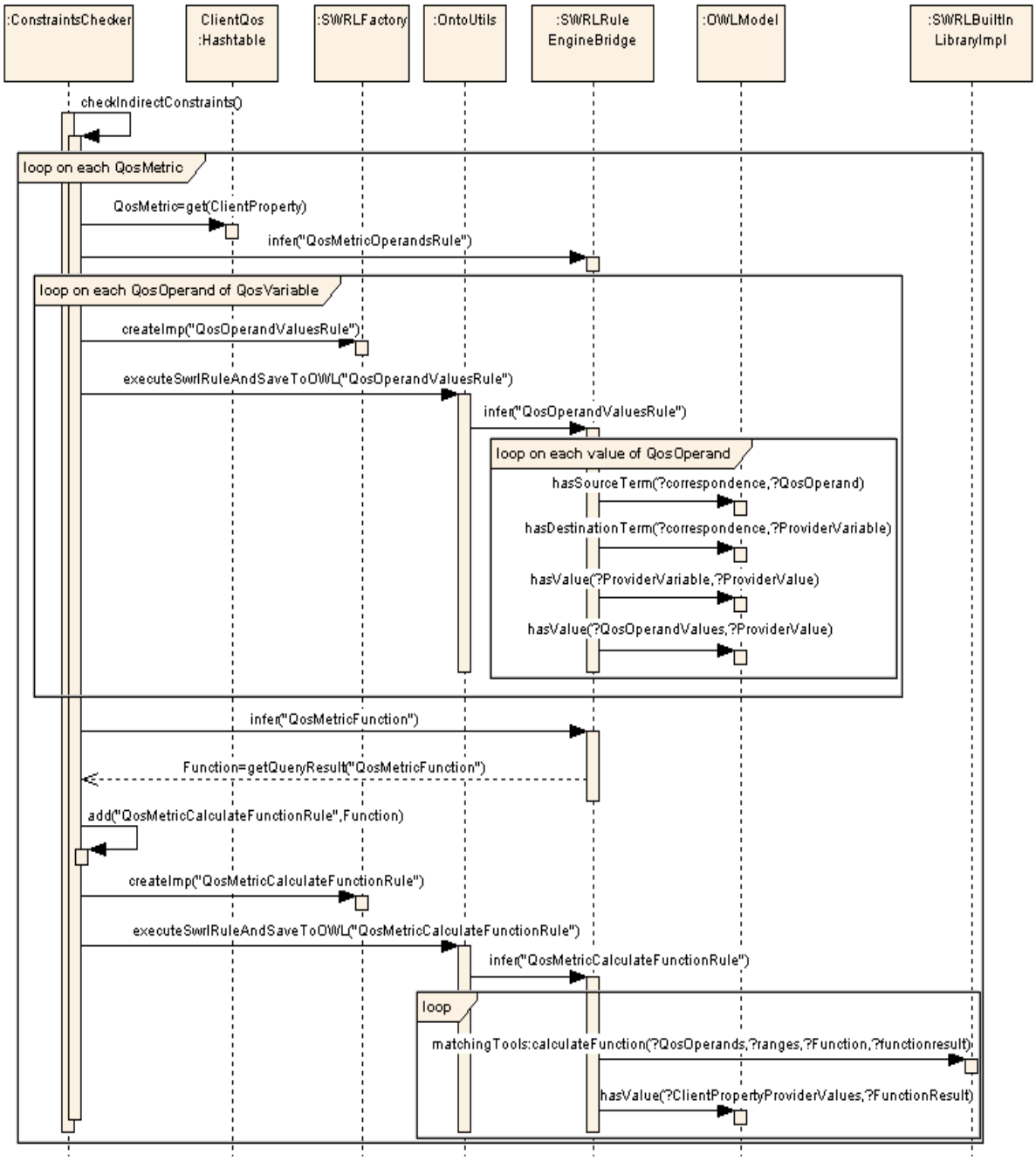

Figure 57 – Diagramme de séquence de la méthode "*checkIndirectConstraints*"

Pour chaque opérande *QosOperand* :

* La méthode *createOWLNamedClass("QosOperandValues")* crée un nouveau concept nommé *QosOperandValues* qui va contenir toutes les valeurs possibles de cette opérande chez le fournisseur.

*La méthode *createImp("QosOperandValuesRule")* crée une nouvelle règle *QosOperandValuesRule*

qui va permettre de récupérer toutes les valeurs de l'ontologie du fournisseur qui correspondent à cette opérande.

* L'opérande est ajoutée à la règle *QosMetricCalculateFunctionRule* pour pouvoir calculer le résultat de la fonction de la variable *QosMetric* considérée.

3.b Après avoir terminé le traitement de toutes les opérandes, la règle *QosMetricCalculateFunctionRule* est exécutée pour retourner toutes les valeurs possibles de la variable *QosMetric* selon les valeurs proposées par le fournisseur. Toutes ces valeurs seront ajoutées en tant que instances du concept *ClientPropertyProviderValues* de la propriété considérée du client comme illustré sur le diagramme de séquence de la Figure 57.

Après la récupération des valeurs possibles (directes et indirectes) de chaque propriété des contraintes du client, le seuil, l'opérateur et le concept *ClientPropertyProviderValues* sont ajoutés à la règle *checkConstraintsRule* qui assure le test de toutes les combinaisons de ces valeurs par rapport au seuil demandé par le client.

En cas de compatibilité pour l'une des combinaisons possibles, l'algorithme de génération du contrat est exécuté. Si aucune combinaison n'est satisfaisante, un message est affiché à l'expert commercial du fournisseur pour lui donner les causes de l'incompatibilité entre les intentions du client et les offres disponibles. Dans la section suivante, nous détaillons la conception technique qui nous a permis d'implanter l'algorithme de génération de contrats de qualité de service.

1.5 GENERATION DE CONTRATS DE QUALITE DE SERVICE

Dans cette section, nous détaillons la génération des éléments principaux d'un contrat de qualité de service en cas de compatibilité entre les intentions du client et les offres du fournisseur.

Comme illustré dans la Figure 58, les instances du concept *PropertyMesurementDirective* de l'ontologie du fournisseur ainsi que leurs protocoles et leurs opérations associés vont constituer respectivement les directives de mesure *MesurementDirective*, leurs protocoles et leurs opérations du nouveau contrat à l'aide du *built-in createMesurementDirective*.

Le *built-in CreateMetric* crée l'instance du concept *Metric* du contrat et lui attribue la directive de mesure correspondante précédemment créée.

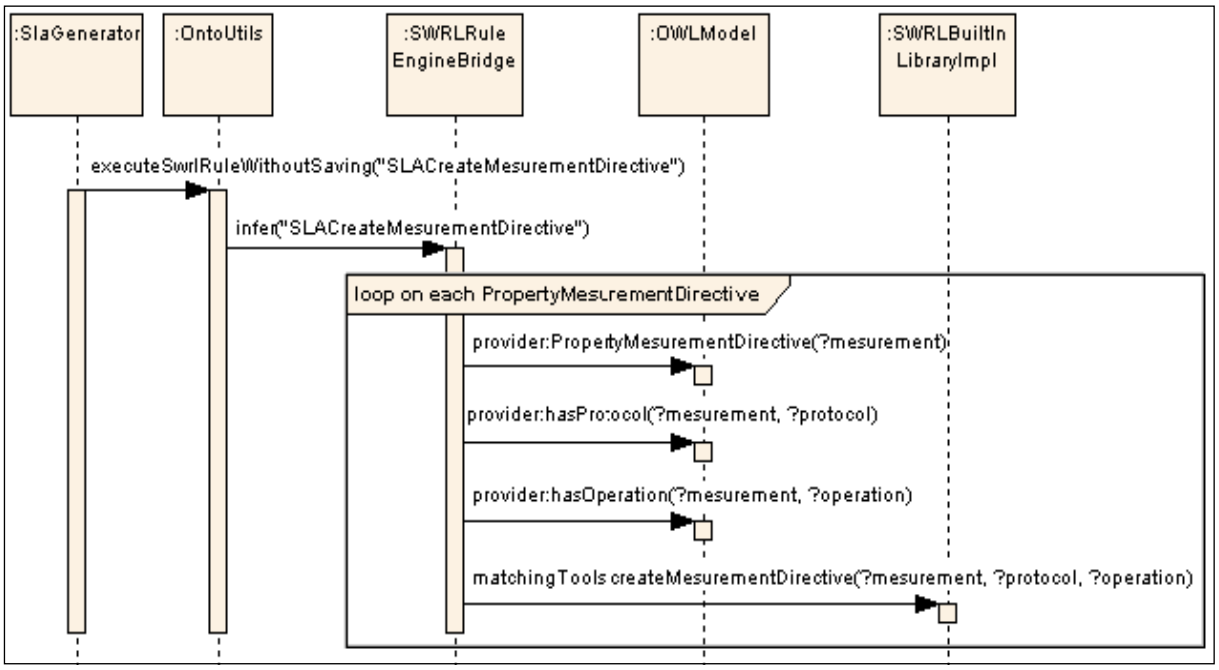

Figure 58 – Diagramme de séquence de l'étape de génération des directives de mesure

• Dans le cas d'une correspondance directe entre l'ontologie du client et l'ontologie du fournisseur, la valeur de la métrique est récupérée à partir des propriétés du fournisseur *providerProperty* qui ont une correspondance valide avec une propriété du client. Ceci correspond à l'exécution de la règle SWRL *SLACreateSimpleMetric* présenté dans la Figure 59.

• Dans le cas d'une correspondance indirecte, la valeur de la métrique est récupérée à partir des propriétés du fournisseur *providerProperty* qui ont une correspondance valide avec une variable de l'ontologie de QdS dont sa métrique *QoSMetric* a une correspondance valide avec une propriété du client. Ceci correspond à la règle SWRL *SLACreateComposedMetric* présenté dans la Figure 59.

Chaque instance du concept *Property* de l'ontologie du client va constituer un *SLAParameter* dans le contrat. Nos distinguons deux cas :

• Dans le cas d'une correspondance directe entre l'ontologie du client et l'ontologie du fournisseur, le *built-in createSimpleSLAParameter* (présenté d ans la Figure 60) va créer l'instance du concept *SLAParameter* à partir des propriétés de l'ontologie du client *clientProperty* qui ont une correspondance valide avec une propriété du fournisseur. La métrique correspondante créée dans l'étape précédente est associée à ce paramètre.

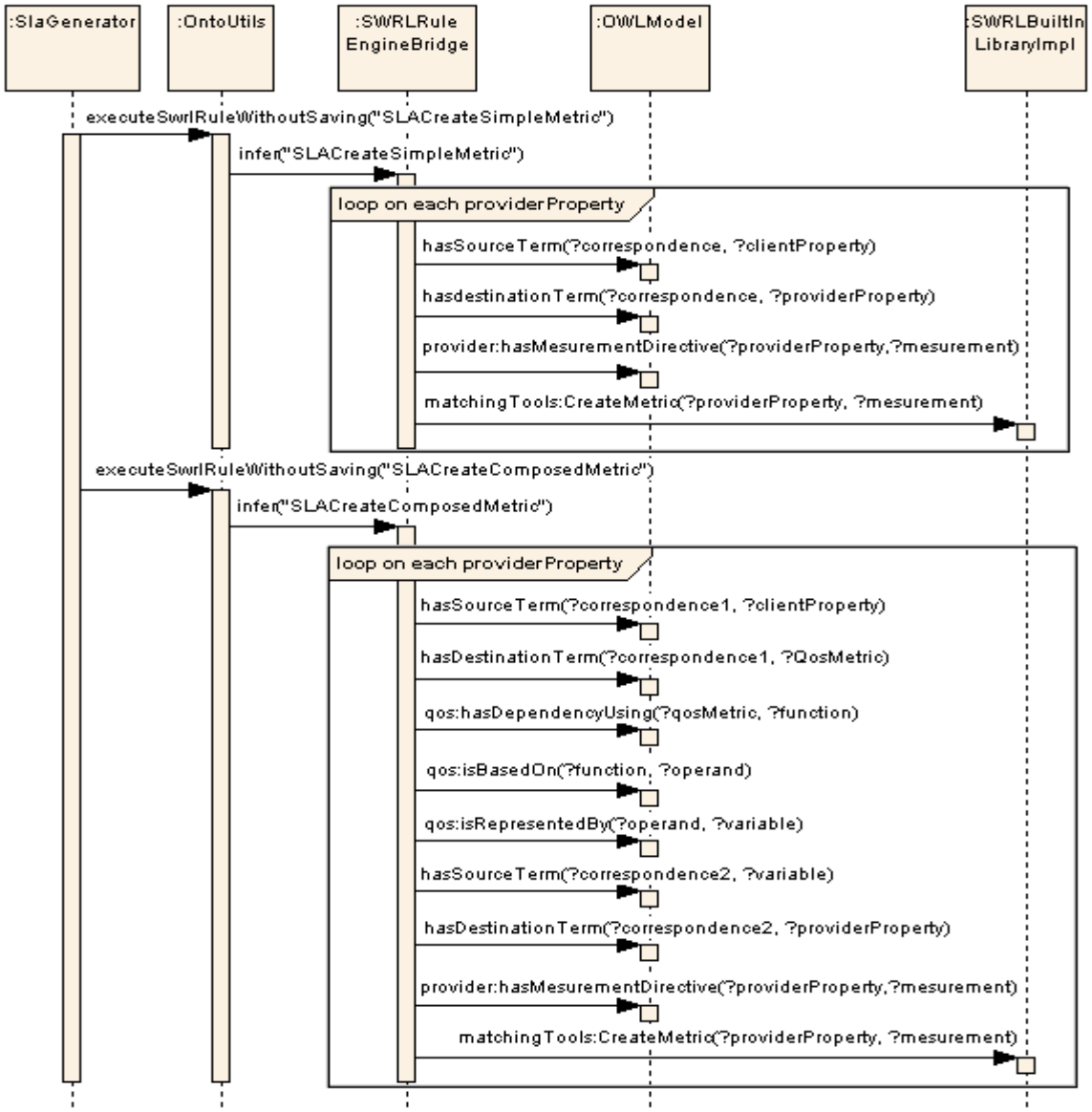

Figure 59 – Diagramme de séquence de l'étape de génération des métriques de QdS

• Dans le cas d'une correspondance indirecte, le *built-in createComposedSLAParameter* (présenté dans la Figure 60) va créer l'instance du concept *SLAParameter* à partir des propriétés de l'ontologie du client qui a une correspondance directe avec une instance de *QoSMetric* de l'ontologie de QdS. Ce dernier admet une fonction et un ensemble d'opérandes. Cette fonction sera attribuée à l'instance du *SLAParameter* ainsi créée afin de pouvoir calculer ses valeurs dans la phase de surveillance.

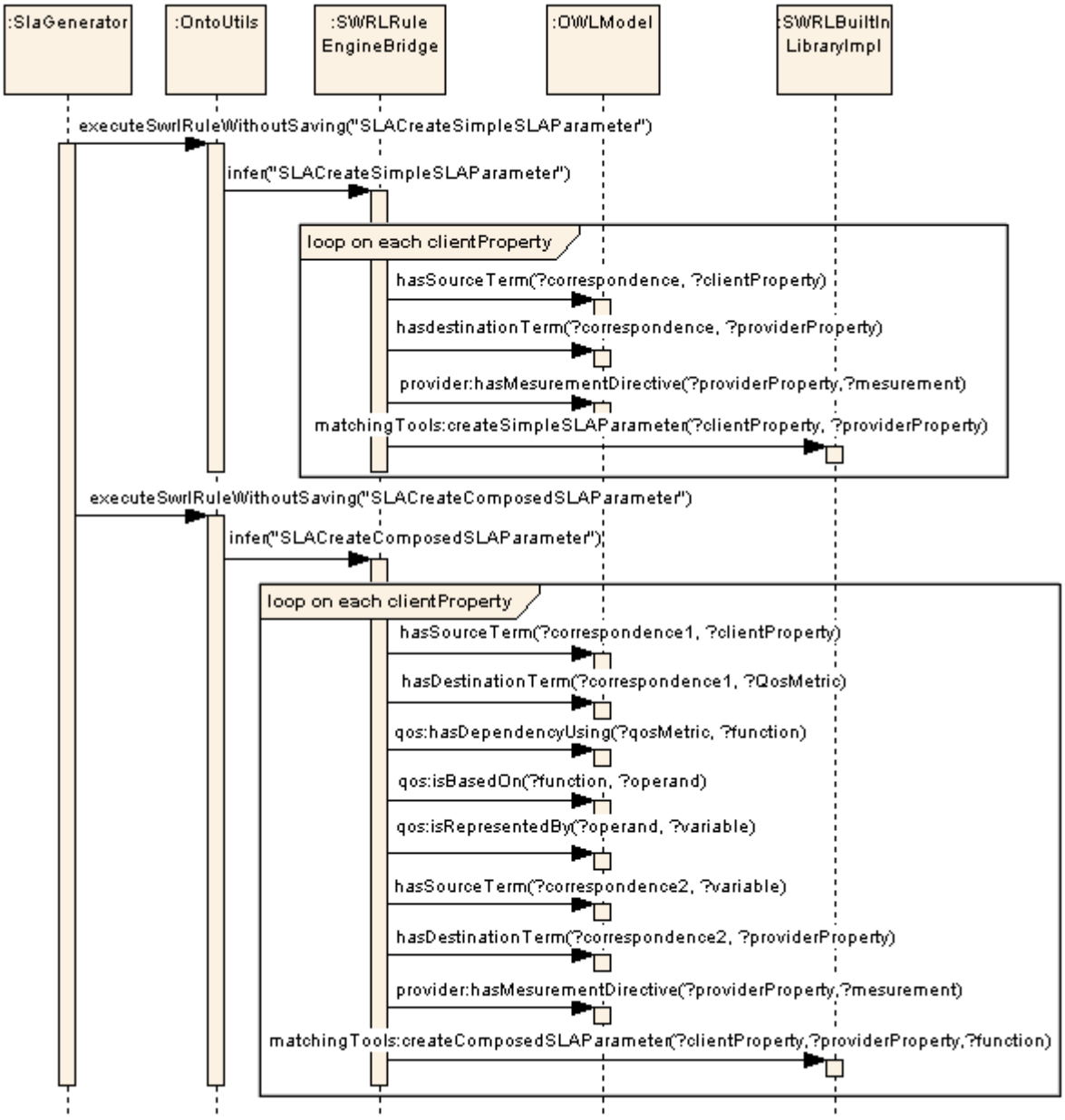

Figure 60 – de séquence de l'étape de génération des Paramètres de QdS

Les valeurs des métriques associées à ce *SLAParameter* correspondent aux propriétés du fournisseur qui ont une correspondance valide aves les opérandes associées à ce *QoSMetric* dans l'ontologie de QdS.

La méthode *generatePredicateRule()* récupère pour chaque contrainte du client, sa propriété, la valeur de son seuil et son opérateur en exécutant la règle SWRL *getSLAPredicateParams*. Cette méthode créée une règle *propertyPredicateEvaluationRule* pour chaque contrainte dans le contrat généré. L'exécution de cette règle permettra à la phase de surveillance de détecter les violations du contrat. La génération de cette règle est assurée par la méthode *generatePredicateRule* illustrée dans la Figure 61.

Nous présentons la conception technique de notre outil de surveillance d'accords de QdS dans la

section suivante.

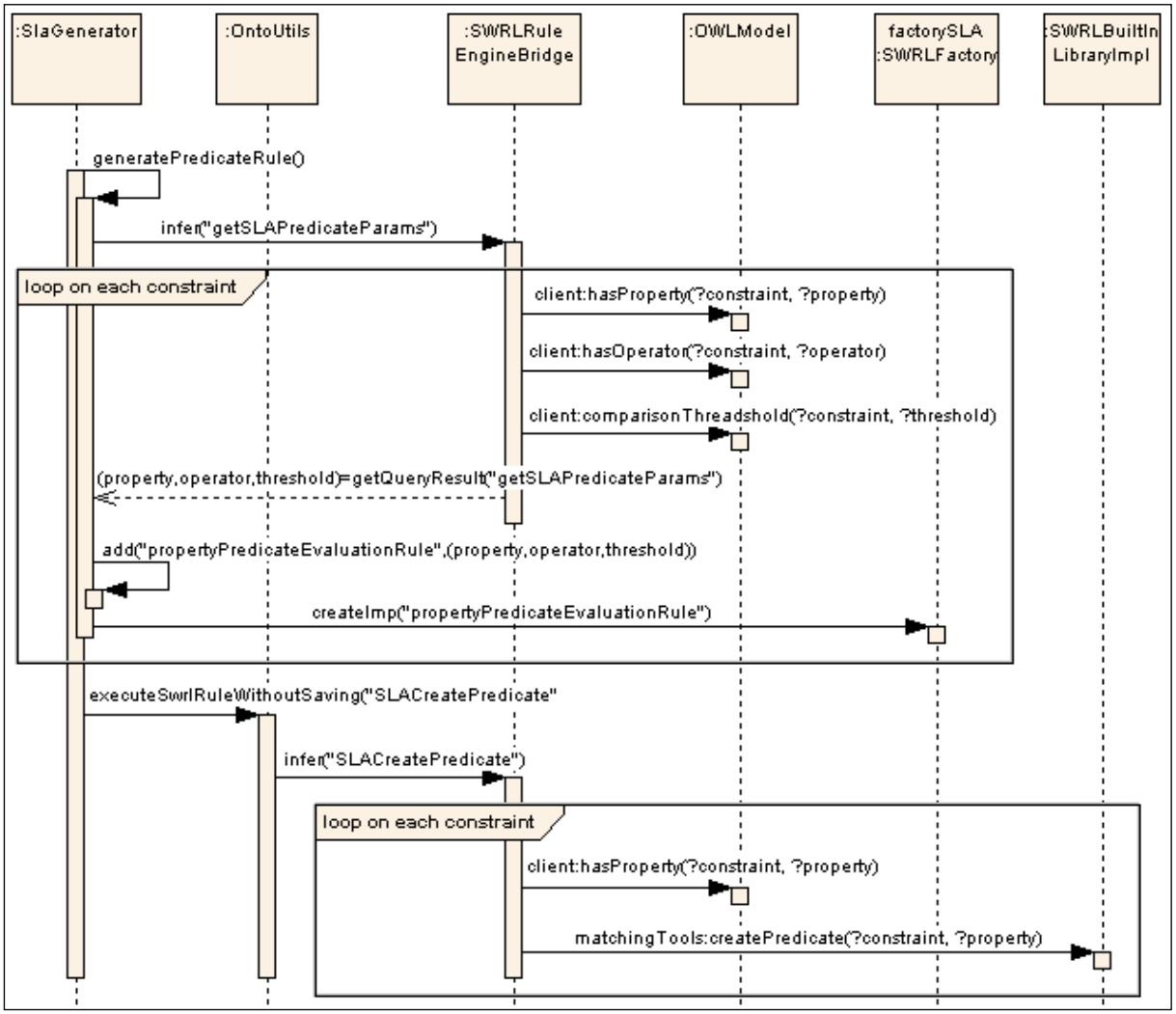

Figure 61 – Diagramme de séquence de l'étape de génération des clauses du contrat

2. Conception d'outil de surveillance d'accords de QdS

Dans cette section, nous présentons la conception technique de notre outil de surveillance des contrats générés par notre outil de négociation d'accords de qualité de service.

2.1 FONCTIONS OFFERTES AUX UTILISATEURS

Notre outil de surveillance d'accords de QdS est destiné à une partie tierce de confiance qui va assurer la surveillance des clauses de ces accords. Avec cet outil, un utilisateur peut déployer un contrat signé par les parties concernées qui sont généralement le client et le fournisseur. Dès son déploiement, les niveaux de qualité qui ont fait l'objet de cet accord vont être régulièrement calculés, vérifiés et évalués par notre outil.

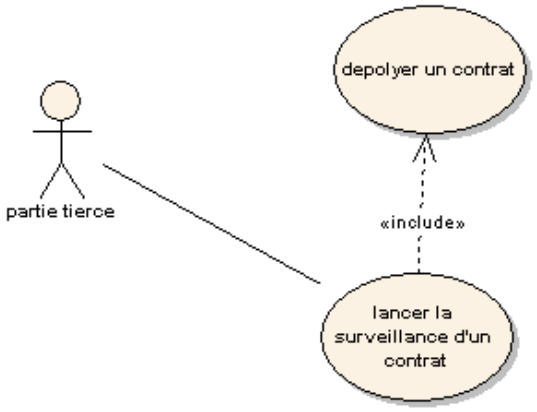

Figure 62 – Diagramme de cas d'utilisation d'outil de surveillance de contrats

La Figure 62 illustre le diagramme de cas d'utilisation de cet outil où l'acteur (partie tierce) peut déployer un contrat de QdS. La seule condition pour déclencher la surveillance du contrat est que ce dernier doit respecter la structure *SLAOnt* (voir section II.4 du chapitre III de ce mémoire). Tout le processus de surveillance est effectué automatiquement le long de la durée de validité du contrat.
Nous donnons plus de détails sur l'implantation du processus de surveillance dans les sections suivantes.

2.2 MODELISATION STATIQUE DE NOTRE OUTIL DE SURVEILLANCE

Dans cette section, nous présentons les classes principales de notre outil de surveillance ainsi que leurs différentes relations. Nous distinguons deux catégories de classes : les classes prédéfinies dans les API utilisées et les classes que nous avons développées pour assurer la surveillance des contrats de QdS. Les classes prédéfinies que nous avons utilisées sont *OWLModel*, *SWRLRuleEngineBridge* et *SWRLFactory*. Elles appartiennent à l'API *Protégé-OWL* que nous avons présentée dans la section II.5 de ce chapitre.

* La classe *OWLModel* offre les méthodes nécessaires pour assurer l'interaction avec l'ontologie représentant le contrat à surveiller. Ces méthodes permettent d'interroger les différentes instances de cette ontologie. Elles permettent aussi de créer d'autres instances en cas de besoin.

* La classe *SWRLRuleEngineBridge* est une passerelle entre les instances du modèle OWL *OWLModel* qui contient des règles *SWRL* et les moteurs d'inférence. Son objectif est de fournir l'infrastructure nécessaire pour intégrer les moteurs d'inférence dans l'API *Protégé OWL* afin de pouvoir exécuter les règles SWRL. Elle permet (1) d'exporter les règles SWRL et les instances OWL vers le moteur d'inférence, (2) de récupérer les résultats de l'exécution de ces règles et (3) d'injecter les nouvelles connaissances inférées dans les instances du modèle OWL. La méthode principale de cette classe est *infer()* qui assure l'exécution des règles SWRL activées dans l'ontologie.

* La classe *SWRLFactory* fournit une API Java qui peut être utilisée pour créer et modifier les règles

SWRL définies dans une ontologie OWL. Les méthodes principales de cette classes sont *createImp()* pour créer une règle SWRL et *getImp()* pour récupérer une règle SWRL définie dans l'ontologie.

La classe la plus importante de notre outil de surveillance est *SWRLBuiltinLibrary*. Elle fournit un mécanisme très puissant d'extension qui permet l'utilisation de méthodes définies par l'utilisateur dans les règles SWRL. Ces méthodes correspondent aux *built-ins* que nous avons présentés dans la section II.3 de ce chapitre.

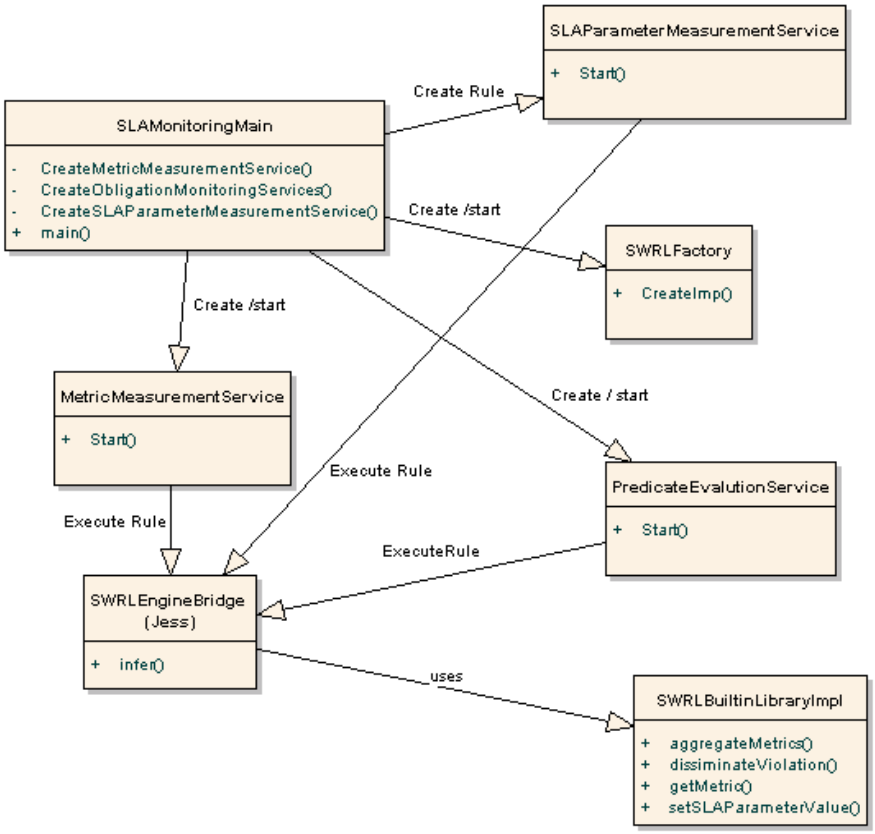

Figure 63 – Diagramme de classes de notre outil de surveillance de contrats de QdS

Chaque *built-in* défini par l'utilisateur dans les règles SWRL doit correspondre à une méthode dans cette classe. Nous avons défini quatre *built-ins* principaux dans notre outil. Le premier est *getMetric* qui invoque la directive de mesure d'une métrique spécifiée et envoi sa valeur à l'historique de mesures. Le deuxième est *aggregateMetrics* qui invoque la fonction d'agrégation nécessaire pour calculer les paramètres de QdS à partir des valeurs des métriques précédemment stockées dans l'historique. Le troisième est *disseminateViolation* qui envoie un message de notification en cas de violation. Ce message contient la cause de la violation détectée. Enfin, le quatrième *built-in* est *applyPenalty* qui applique la pénalité spécifiée dans le contrat lors de la détection d'une violation.

A part, la classe SWRLBuiltinLibrary de notre outil contient quatre autres classes principales : SLAMonitoringMain, MetricMeausrementService, SLAParameterMeasurementService et ObligationMonitoringService.

- *SLAMonitoringMain* permet de créer tous les services de mesures nécessaires pour la surveillance des contrats déployés. Les méthodes principales de cette classe sont : *createMetricMeasurementService()* pour créer un service de mesure d'une métrique donné, *createSLAParameterMeasurementService()* pour créer un service de mesure d'un paramètre de qualité de service et *createObligationMonitoringService()* pour créer un service d'évaluation d'une obligation donnée.
- *MetricMeasurementService* assure la mesure des métriques définies dans le contrat. Pour chaque métrique, une instance de service est créée par la classe *SLAMonitoringMain*. Le nom de la métrique et sa fréquence de mesure doivent être spécifiés lors de cette création.
- *SLAParameterMeasurementService* assure la mesure des paramètres de haut niveau de qualité de service. Il est instancié pour chaque *SLAParameter* défini dans le contrat. Lors de son instanciation, il prend en entrée le nom du paramètre, la fonction et les noms des métriques nécessaires pour calculer sa valeur. La fréquence de mesure du paramètre de QdS doit être aussi récupérée du contrat et passée à la création de ce service.
- *ObligationMonitoringService* permet de vérifier la validité des clauses du contrat par rapport aux mesures effectuées par le service de mesure des paramètres de QdS. Il est instancié pour chaque obligation du contrat représentée par une règle SWRL définissant le niveau de service qui a été négocié et approuvé par les deux parties. Le nom de la règle SWRL ainsi que sa fréquence d'évaluation sont passés à ce service lors de son instanciation.

La Figure 63 illustre les classes principales de notre outil de surveillance de contrats de QdS ainsi que les différentes relations qui existent entre elles.

2.3 MODELISATION DYNAMIQUE DE NOTRE OUTIL DE SURVEILLANCE

Dans cette section, nous présentons la conception dynamique du processus de surveillance en se basant sur les classes que nous avons présentées dans la section 2.2 de ce même chapitre.

Pour chaque métrique de QdS définie dans le contrat, la classe principale de notre outil *SLAMonitoringMain* crée une règle SWRL qui assure l'invocation de sa directive de mesure afin de récupérer sa valeur.

Cette classe principale, crée ensuite un service de mesure pour cette métrique en lui passant la règle SWRL à appliquer. Puis, pour chaque instance du concept *SLAParameter* du contrat à surveiller, *SLAMonitoringMain* crée une règle SWRL qui permet de calculer la valeur du paramètre. Cette règle est ensuite utilisée pour créer un service de mesure pour ce paramètre. La classe principale, récupère aussi les différentes obligations définies dans le contrat. Pour chaque obligation, elle récupère son prédicat et la passe à une nouvelle instance du service *ObligationMonitoringService*. La Figure 64 illustre le déroulement dynamique de la phase de surveillance déclenchée par le déploiement d'un contrat de QdS dans *SLAOnt-Monitoring*.

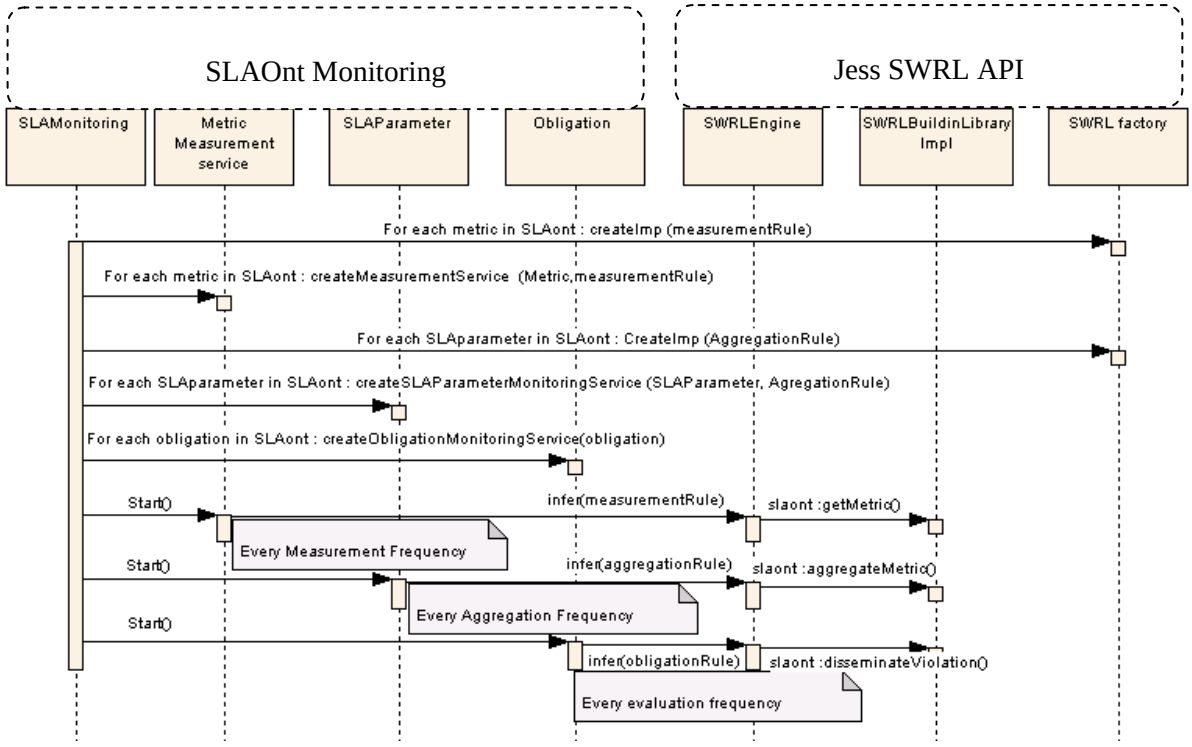

Figure 64 – Diagramme de séquence de notre outil de surveillance de contrat de QdS

IV. CAS D'ETUDES

Dans cette section, nous détaillons l'utilisation de nos outils dans deux cas d'études. Le premier concerne un fournisseur d'une bibliothèque numérique de vidéos et le deuxième concerne un fournisseur d'hébergement de services.

1. Service de téléchargement d'une bibliothèque numérique de vidéos

Pour valider notre approche d'alignement des intentions des clients avec les offres des fournisseurs, nous avons utilisé notre outil d'alignement sémantique pour l'appliquer sur un cas d'étude concret. Dans notre cas, le client a une intention de télécharger des films et le fournisseur dispose d'un ensemble d'offres de films à télécharger.

1.1 DESCRIPTION DES ONTOLOGIES UTILISEES

Dans notre cas d'étude, nous avons défini les instances des ontologies représentant l'intention du client, les offres du fournisseur ainsi que les relations entre les différentes qualités de service.
Dans l'exemple étudié, le client veut télécharger un film. Il impose trois contraintes dans son intention :

• Contrainte sur le prix : Le client veut que le prix du film ne dépasse pas 3 Euros.

• Contrainte sur le type du film : Le client veut que le film soit de type comédie.

• Contrainte sur le temps de téléchargement du film : Le client veut que le temps de téléchargement ne dépasse pas 10 minutes.

La Figure 65 décrit l'ontologie du client que nous avons utilisée.

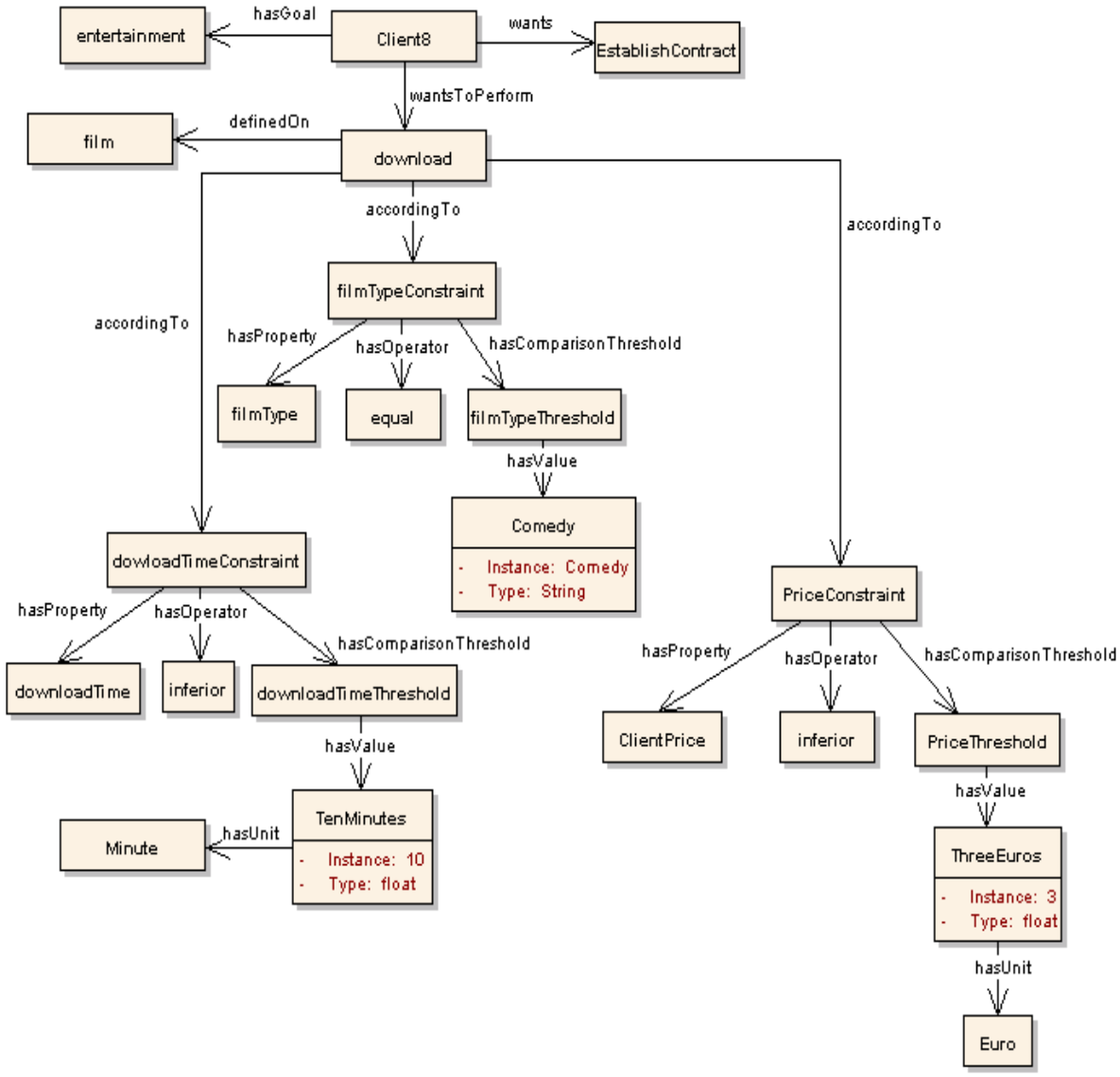

Figure 65 – instance de l'ontologie du client

Dans notre exemple, le fournisseur offre deux services : le téléchargement des films *DownloadFilm* et la recherche des films *SearchFilm*. Chaque film admet une taille, un nom et un type. Le fournisseur impose trois contraintes pour définir une offre *FilmGoldOffer* parmi d'autres offres.

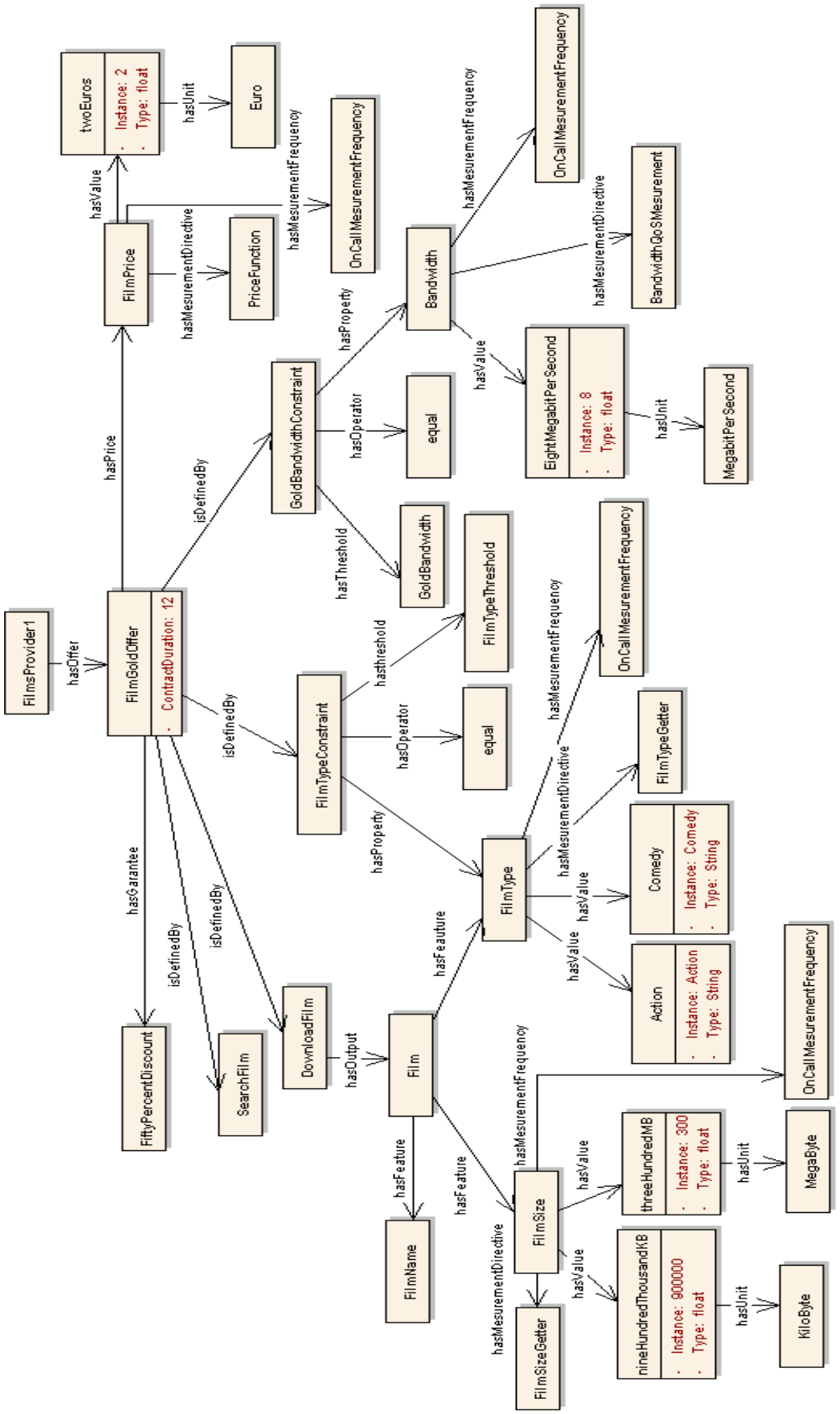

Figure 66 – Instance de l'ontologie du fournisseur

Une contrainte fonctionnelle sur les types de ses films qui peuvent être de type comédien ou d'action, une contrainte non fonctionnelle concernant la mesure de la bande passante qui est fixée à 8 Mb/s (8 Méga bits par seconde) et enfin une contrainte sur le prix des films de cette catégorie qui est fixé à 2 Euros par film. Les fichiers de ces films peuvent avoir deux tailles possibles : 900000Ko et 300Mo. Nous avons utilisé des unités différentes afin de vérifier la fonctionnalité de conversion d'unités. La Figure 66 montre une modélisation simplifiée des principales instances de l'ontologie du fournisseur que nous avons utilisée dans notre cas d'étude.

Après avoir présenté les besoins du client et les offres du fournisseur, nous avons remarqué que les temps de téléchargement des films chez le fournisseur sont exprimés indirectement. En effet, le fournisseur spécifie la taille de chaque film ainsi que la bande passante. Ces deux paramètres permettent de calculer le temps de téléchargement en appliquant la formule suivante (F3) :

$$\text{Temps de téléchargement du film} = \frac{\text{taille du film}}{\text{bande passante}} \quad \text{(F3)}$$

Le besoin d'un outil qui permet de calculer la valeur du temps de téléchargement d'un film chez le fournisseur, nous a poussé à créer un exemple de l'ontologie de qualité de service. Cette ontologie montre que le temps de téléchargement *downloadTime* exprimé en secondes peut être calculé à partir de la division de la taille du film *TransferSize* (taille du film) exprimée en MB (Méga octet), par la bande passante *Bandwidth* (bande passante) exprimée en MB/s (Méga octet par seconde).

L'exemple de l'ontologie de qualité de service utilisé dans notre cas d'étude est présenté dans la Figure 67.

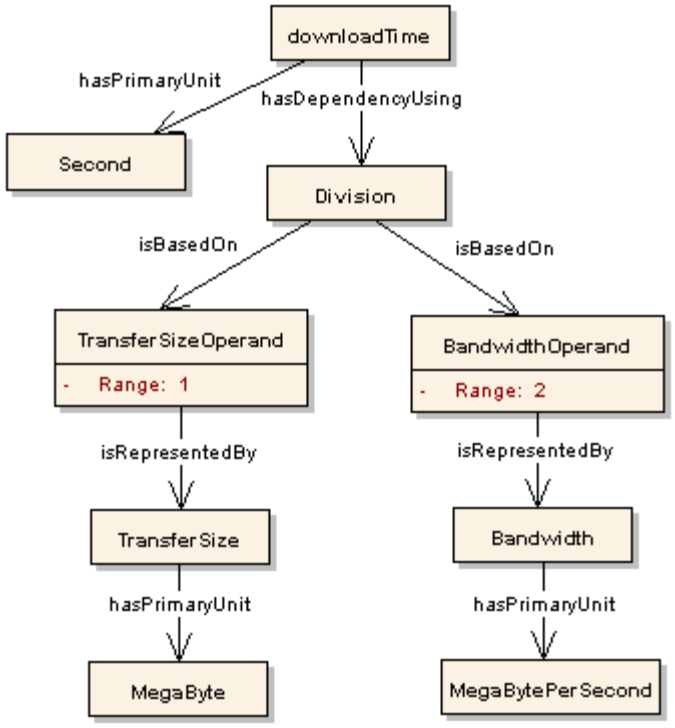

Figure 67 – Instance de l'ontologie de qualité de service

Nous avons aussi défini une ontologie de conversion d'unités de mesure *UnitsOnto* qui assure la correspondance des unités des valeurs requises par le client d'une part et celles offertes par le fournisseur d'autre part.

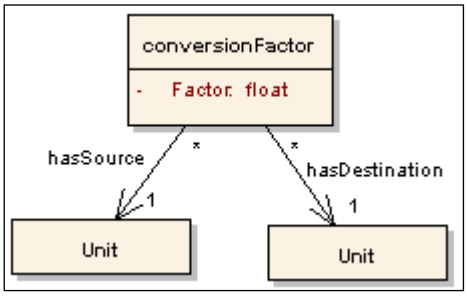

Figure 68 – Structure générale de l'ontologie « UnitsOnto »

Le concept principal de cette ontologie est *conversionFactor* qui admet une source et une destination représentant respectivement la première unité à convertir et la deuxième unité qui constitue le résultat de la conversion. Ce concept admet aussi un facteur de conversion qui permet le passage de la première unité à la deuxième (voir Figure 68).

UnitsOnto doit être référencée dans toutes les autres ontologies en utilisant les mécanismes d'importation OWL. La Figure 69 montre deux instances du concept *conversionFactor* que nous avons utilisées dans notre cas d'étude. La première montre la conversion du Kilo octet au Méga octet et la deuxième montre la conversion de seconde en heure.

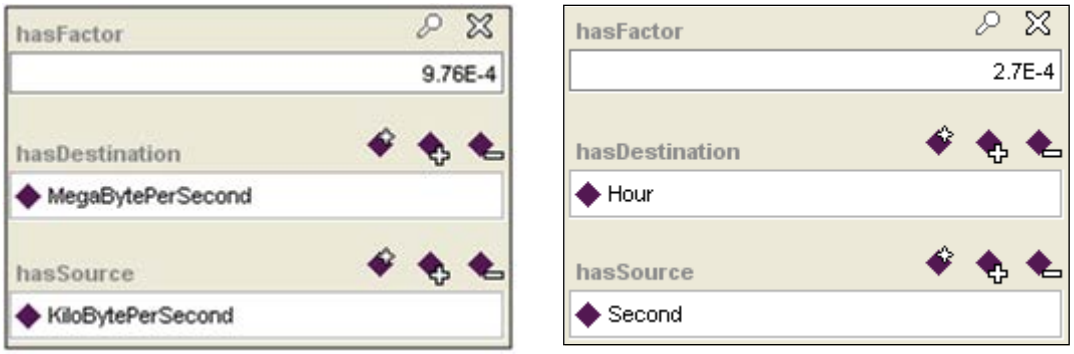

Figure 69 – Exemples de conversion d'unités à l'aide de "UnitsOnto"

La conversion des unités est utilisée dans l'étape d'évaluation des contraintes de notre approche d'alignement sémantique entre les intentions du client et les offres du fournisseur. Cette conversion est effectuée selon le type des correspondances trouvées entre les termes du client et celles du fournisseur :

• Dans le cas d'une correspondance directe, la conversion sera affectée aux unités du fournisseur pour qu'elles soient conformes à celles fixées par le client.

• Dans le cas d'une correspondance indirecte, les unités du fournisseur seront converties à celles de

l'ontologie de qualité de service. Puis, l'unité du résultat de la fonction de l'ontologie de la qualité de service sera convertie à celle définie par le client.

1.2 UTILISATION DE NOTRE OUTIL DE NEGOCIATION DE QDS

Cette section présente quelques interfaces de notre outil de négociation de QdS en utilisant les instances des ontologies présentées dans la section IV. Dans cette section, nous retrouvons les différents cas d'utilisation illustrés par la Figure 51.

Afin de pouvoir utiliser notre outil de négociation de QdS, il faut importer les ontologies représentant les intentions du client et les offres du fournisseur. Ensuite le message *Please wait while loading the necessary ontologies* indique que les ontologies sont en cours d'importation. Si leur structure est correcte, le message *All ontlogies successfully loaded !* s'affiche (voir Figure 70).

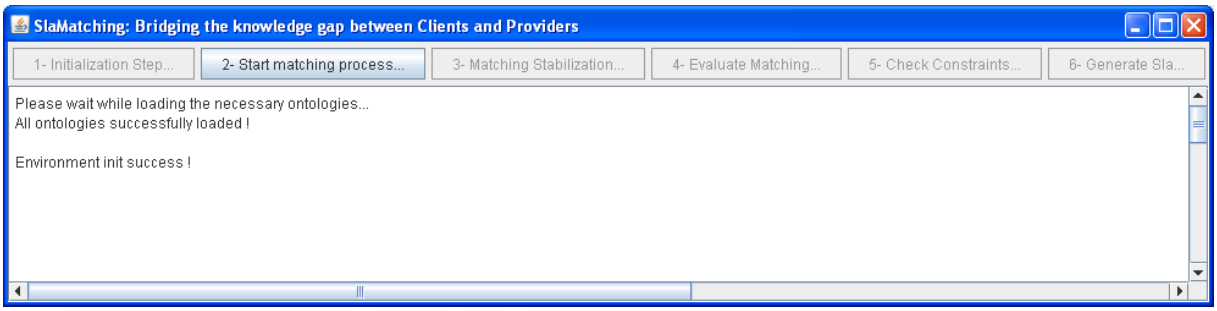

Figure 70 – interface d'initialisation de notre outil de négociation de QdS

Dès lors, le processus d'alignement peut être lancé suivi par l'étape de raffinement des correspondances trouvées ainsi que l'étape de leur évaluation globale. Le déroulement de ces étapes dans notre cas d'étude est illustré dans la Figure 71.

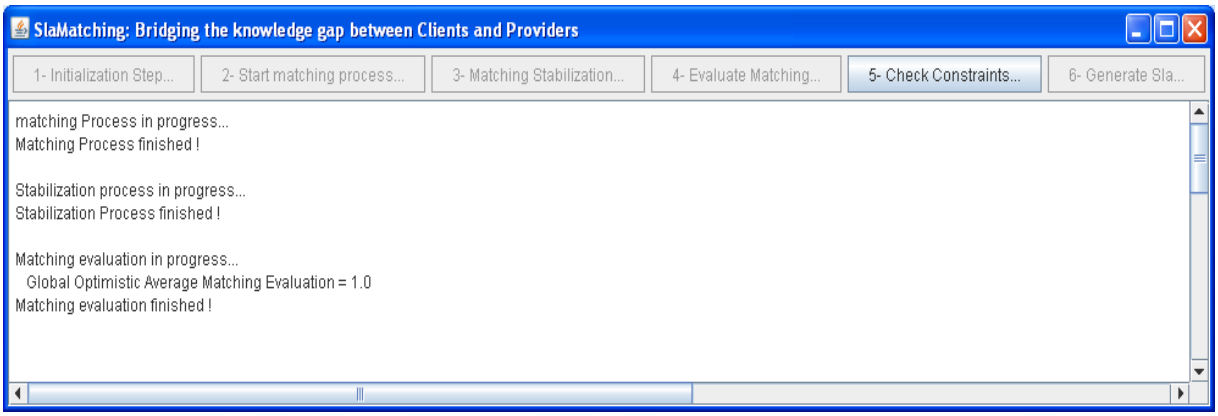

Figure 71 – Déroulement du processus d'alignement pour notre cas d'étude

Ensuite, l'expert peut lancer la vérification des contraintes du client par rapport aux offres du fournisseur à l'aide du bouton *Check Constraints*. Si toutes les contraintes sont satisfaites, le message *The property PropertyName is satisfied* sera affiché pour chaque propriété vérifiée. Dans ce cas,

l'utilisateur peut passer à l'étape suivante qui est la génération automatique d'une première version du contrat de qualité de service entre le client et le fournisseur. Dans le cas contraire, un message indiquant l'incompatibilité partielle ou totale sera affiché et l'accès à l'étape suivante sera impossible. La Figure 72 illustre le déroulement de la phase d'évaluation des contraintes pour notre cas d'étude.

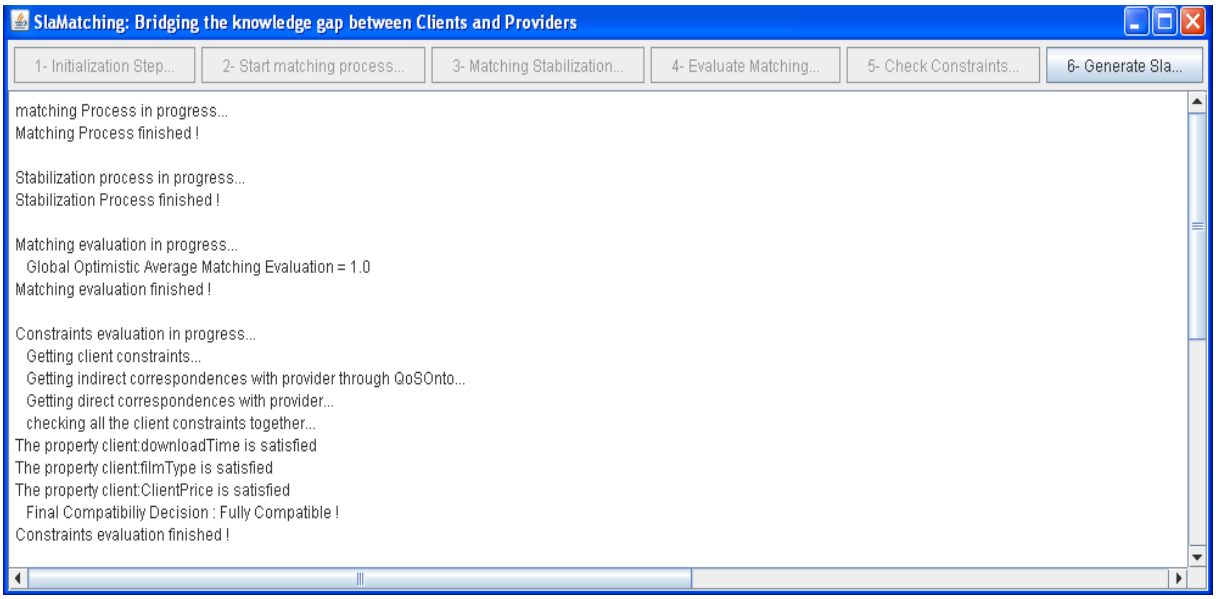

Figure 72 – Interface de vérification de compatibilité des contraintes

En cas de compatibilité, une première version d'un contrat peut être générée grâce au bouton *Generate SLA*. Cette version doit être validée par l'expert commercial du fournisseur pour pouvoir la proposer au client. La Figure 73 illustre les différentes sous-étapes de l'étape de génération de contrat.

Le contrat généré est une instance de la structure *SLAOnt* comme présenté dans la section II.4 du chapitre III de ce mémoire. Cette instance importe toutes les ontologies présentées dans la section 1.1 de ce chapitre. La Figure 74 illustre les principales instances du contrat généré concernant notre cas d'étude.

Le contrat généré s'intéresse au téléchargement des films. Les parties signataires sont le client et le fournisseur. Le service offert par le fournisseur se compose de trois paramètres selon les propriétés du client *: filmTypeParameter*, *downloadTimeParameter* et *ClientPriceParameter*. Pour chacune des trois contraintes imposées par le client correspond un prédicat.

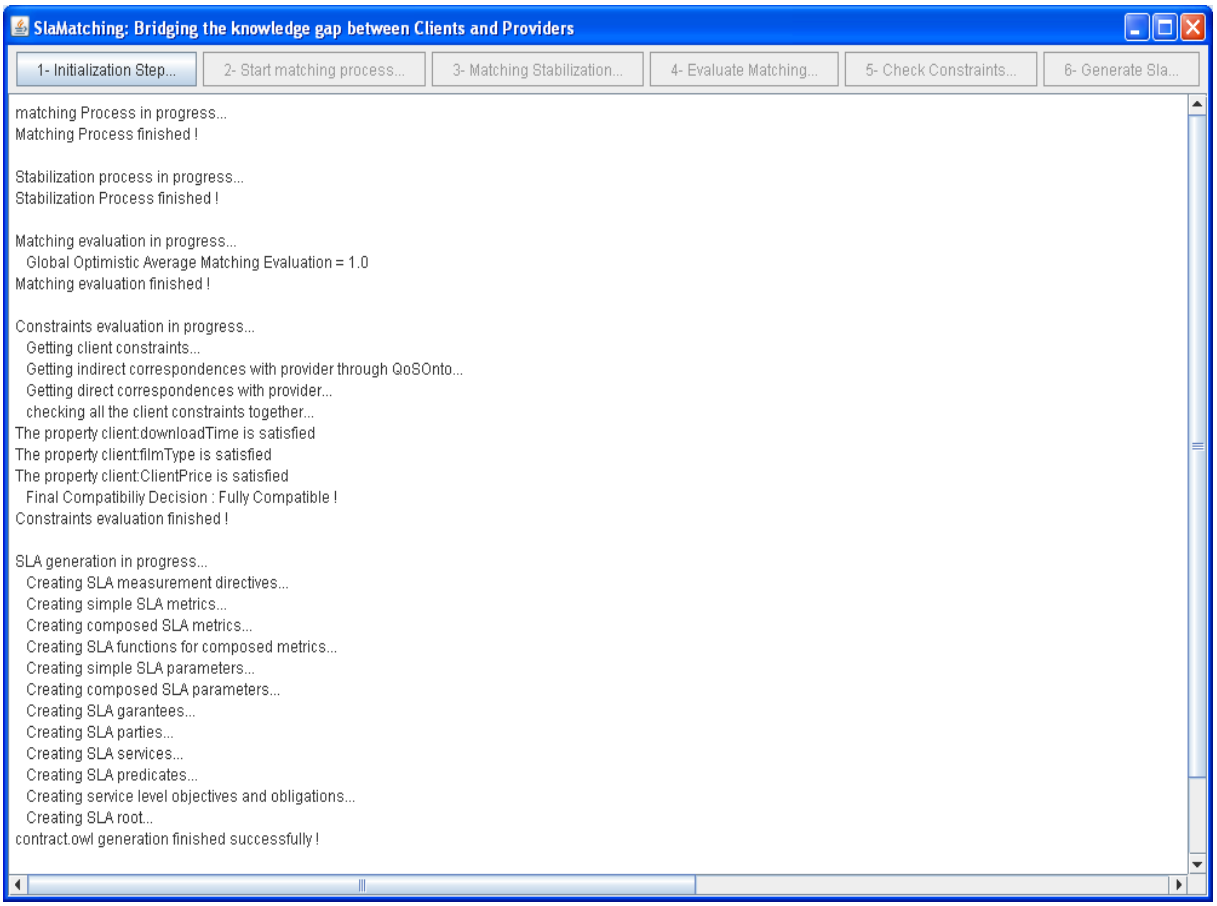

Figure 73 – Déroulement de génération du contrat dans notre cas d'étude

Chaque prédicat est exprimé par une règle SWRL qui va faciliter le processus de détection des violations du contrat lors de la phase de surveillance des contrats de QdS.

Les règles générées sont les suivantes :

filmTypeEvaluationRule

slaont:hasEvaluation(filmTypeParameter, ?x) ^ swrlb:notEqual(?x, "Comedy") →
slaActions:disseminateViolation("filmTypeConstraint", "false", filmTypeParameter, ?x)

downloadTimeEvaluationRule

slaont:hasEvaluation(downloadTimeParameter, ?x) ^ swrlb:greaterThanOrEqual(?x, 10) →
slaActions:disseminateViolation("dowloadTimeConstraint","false",downloadTimeParameter, ?x)

ClientPriceEvaluationRule

slaont:hasEvaluation(ClientPriceParameter, ?x) ^ swrlb:greaterThanOrEqual(?x, 2.0) →
slaActions:disseminateViolation("PriceConstraint", "false", ClientPriceParameter, ?x)

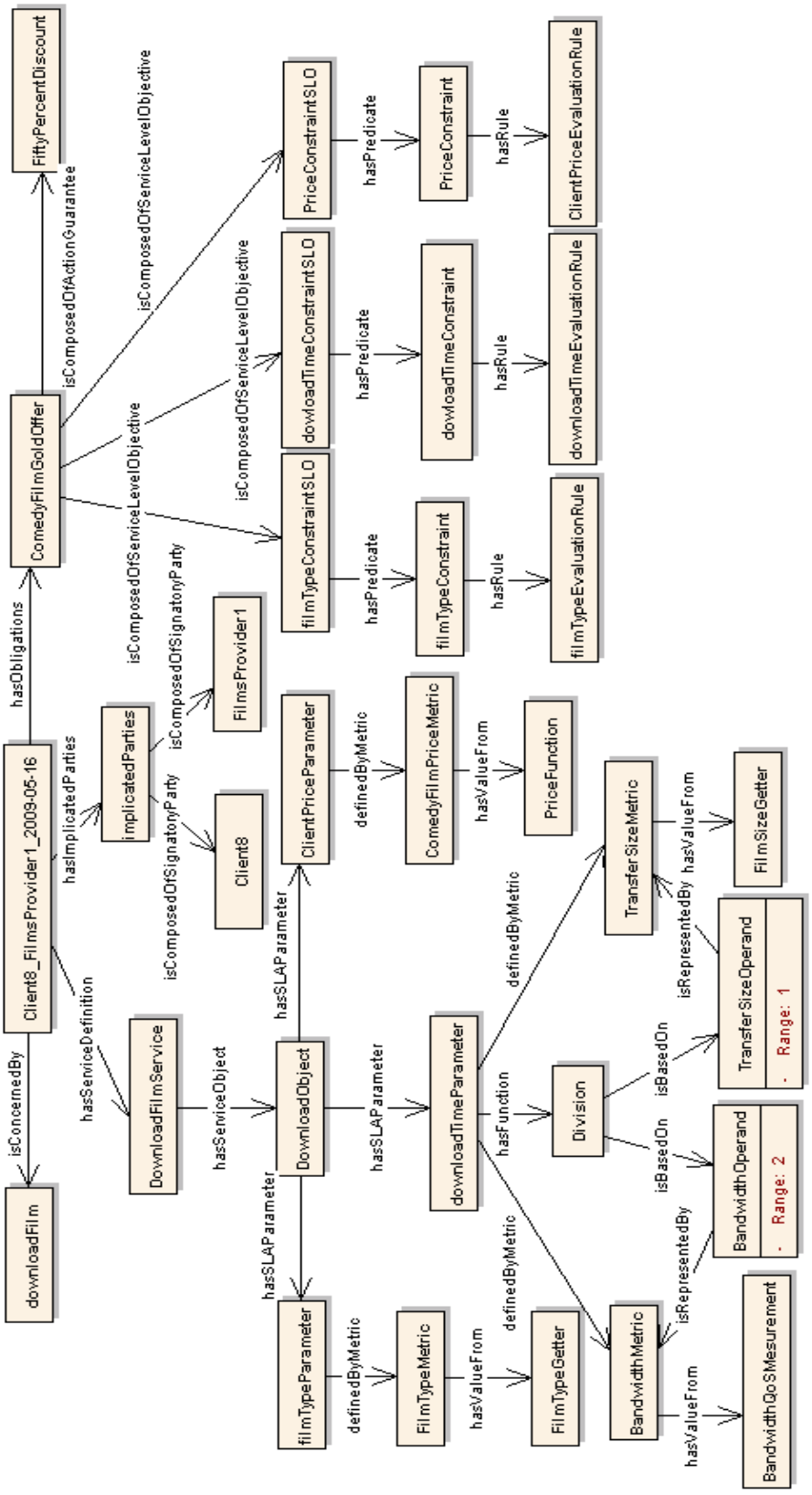

Figure 74 – Instance du contrat généré pour notre cas d'étude

2. Fournisseur d'hébergement de services

Afin de valider notre approche de surveillance de contrats de QdS, nous avons utilisé l'outil que nous avons développé sur un cas d'étude que nous avons nommé *FlightSLA*. Dans ce cas d'étude, une agence de voyage a établi un contrat de QdS avec un fournisseur d'hébergement Web afin de déployer son service de réservation de vols en ligne. Dans cette section, nous commençons par décrire l'ontologie représentant le contrat de QdS établi entre les deux parties. Ensuite, nous exposons quelques interfaces qui montrent le déroulement de la phase de surveillance concernant ce cas d'étude.

2.1 DESCRIPTION DES INSTANCES DU CONTRAT DE QDS

Dans l'exemple étudié, le client impose une contrainte de QdS sur le service de réservation de vol : le temps de réponse moyen ne doit pas dépasser 100 ms. Cette contrainte correspond à un prédicat dans notre contrat de QdS. Ce prédicat est exprimé par une règle SWRL (voir Figure 75) qui facilite le processus de détection des violations du contrat et l'application de la pénalité correspondante.

```
slaont:hasEvaluation(average_response_time, ?x) ^
swrlb:greaterThanOrEqual(?x, 100.0) ^ hasPenalty(lessThan100msPredicate, ?penalty)
→ slaActions:disseminateViolation(lessThan100msPredicate, "false", average_response_time, ?x) ^
slaActions:applyPenalty(TenPerCentDiscount)
```

Figure 75 – Prédicat de QdS de notre cas d'étude

Cette règle déclenche les *built-ins applyPenalty* et *disseminateViolation* dans le cas où le temps de réponse moyen est supérieur à 100 ms. *disseminateViolation* permet d'informer les parties signataires de la violation du contrat. *applyPenalty* assure l'application de la pénalité associée à ce type de violation. Dans notre cas, la pénalité appliquée est une réduction de prix de 10% *TenPercentDiscount*. Nous avons déclaré ces built-ins dans une ontologie externe [118] qui est importée automatiquement par l'ontologie SLAOnt. Nous rappelons que cette dernière représente la structure des contrats qui peuvent être surveillée par notre outil. La Figure 76 illustre les différentes instances du contrat de QdS que nous avons considérées pour notre cas d'étude.

Dans ce contrat, des fréquences de mesure sont spécifiées pour mesurer le temps de réponse à des intervalles réguliers et pour calculer la moyenne des dernières mesures relevées ainsi que l'évaluation du prédicat présenté dans la Figure 75. Ce prédicat permet de vérifier si cette moyenne est inférieure à 100 ms comme requis par l'agence de voyage.

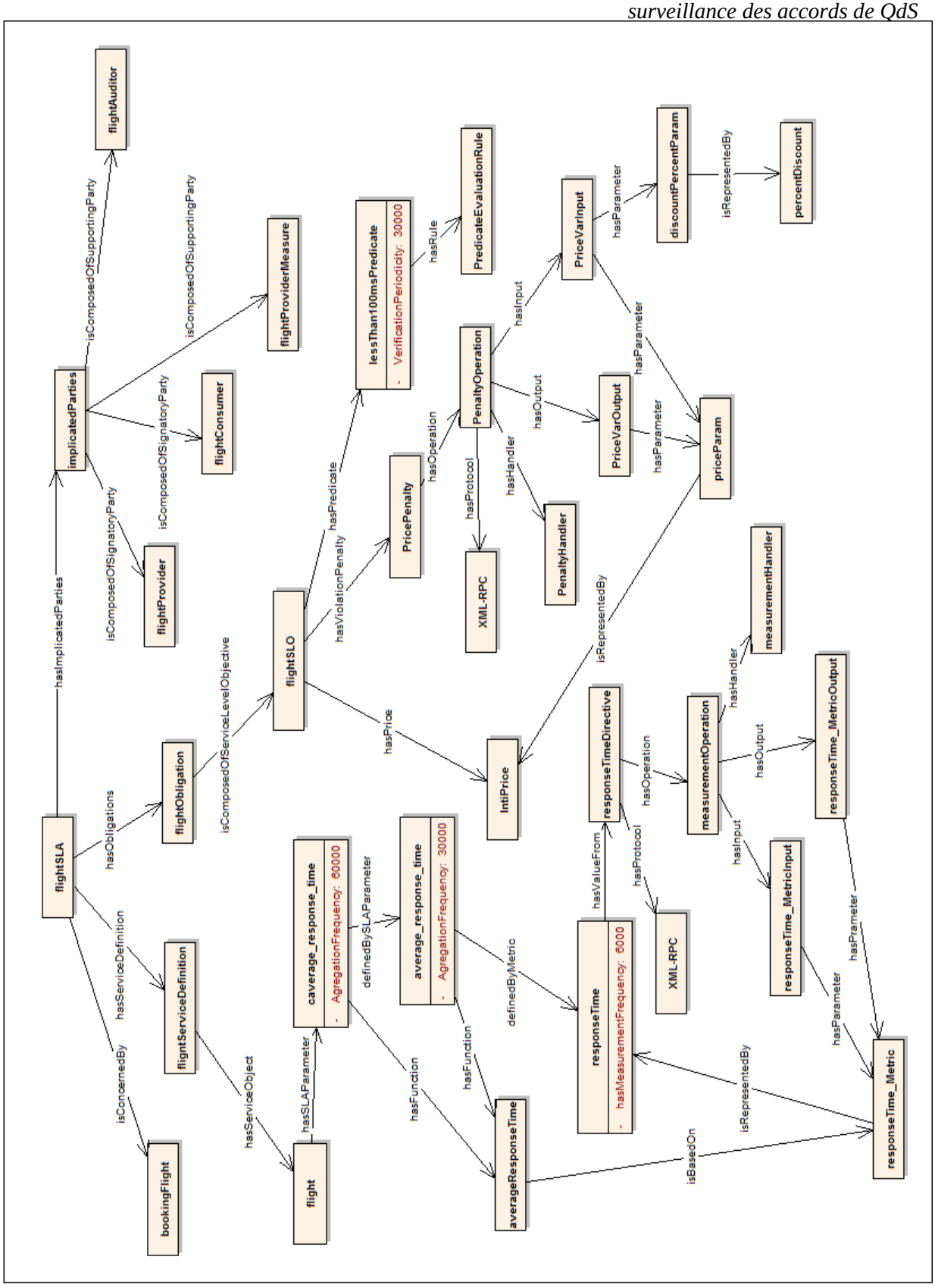

Figure 76 – Contrat de QdS de notre cas d'étude

2.2 UTILISATION DE NOTRE OUTIL DE SURVEILLANCE DE QDS

En utilisant les instances de l'ontologie présentée dans la Figure 76, nous présentons quelques interfaces de notre outil qui montrent le déroulement de la surveillance du contrat représenté par ces instances.

Pour pouvoir surveiller les contrats de QdS, il suffit de les déployer dans notre outil en utilisant le bouton *OpenSLAFile* de son interface principale (voir Figure 77). Il est à noter que le contrat doit être une instance de la structure *SLAOnt*, comme présenté dans la section II.4 du chapitre III de ce mémoire, pour qu'il puisse être utilisé par notre outil. L'ontologie *SLAOnt* est mise en ligne sur le lien [119]. Il suffit d'utiliser le mécanisme d'importation OWL pour définir de nouveaux contrats.

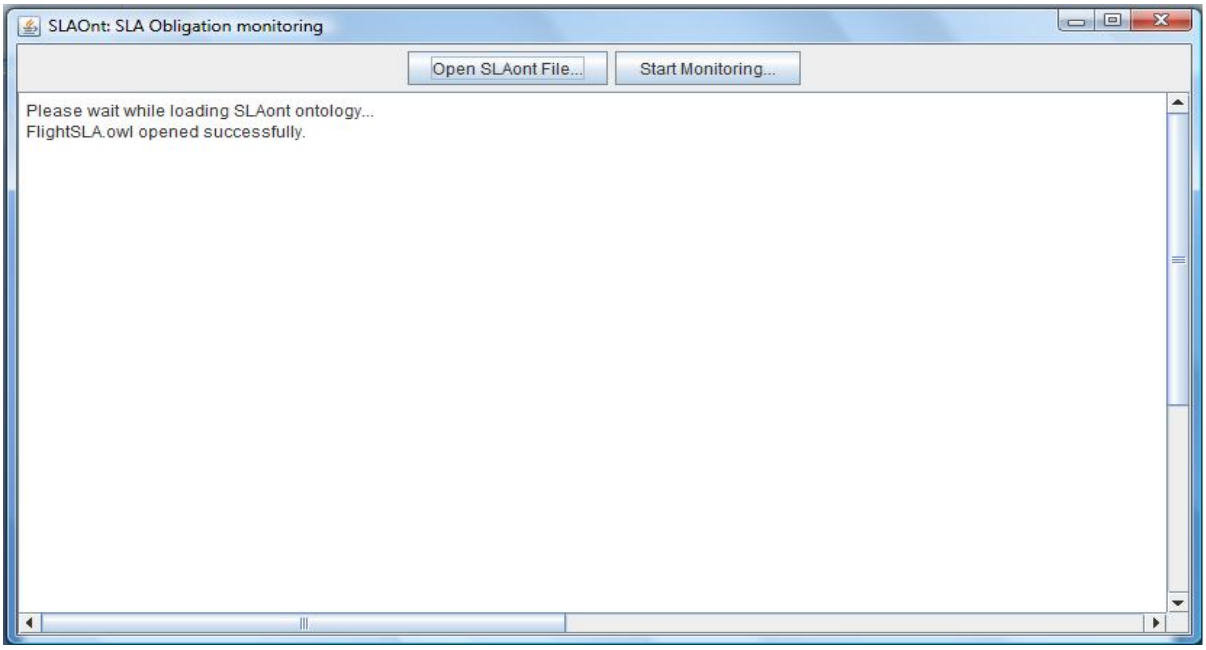

Figure 77 – Interface principale de notre outil de surveillance de QdS

Une fois correctement chargé, le contrat peut ainsi être surveillé grâce au bouton *StartMonitoring* de l'interface principale de notre outil (voir Figure 77).

Nous rappelons que notre outil génère un ensemble de services pour surveiller les valeurs des métriques élémentaires, des paramètres composés de QdS ainsi que les prédicats du contrat. Ces services peuvent être déclenchés soit selon une fréquence définie dans le contrat de QdS soit à l'appel du service demandé par le client.

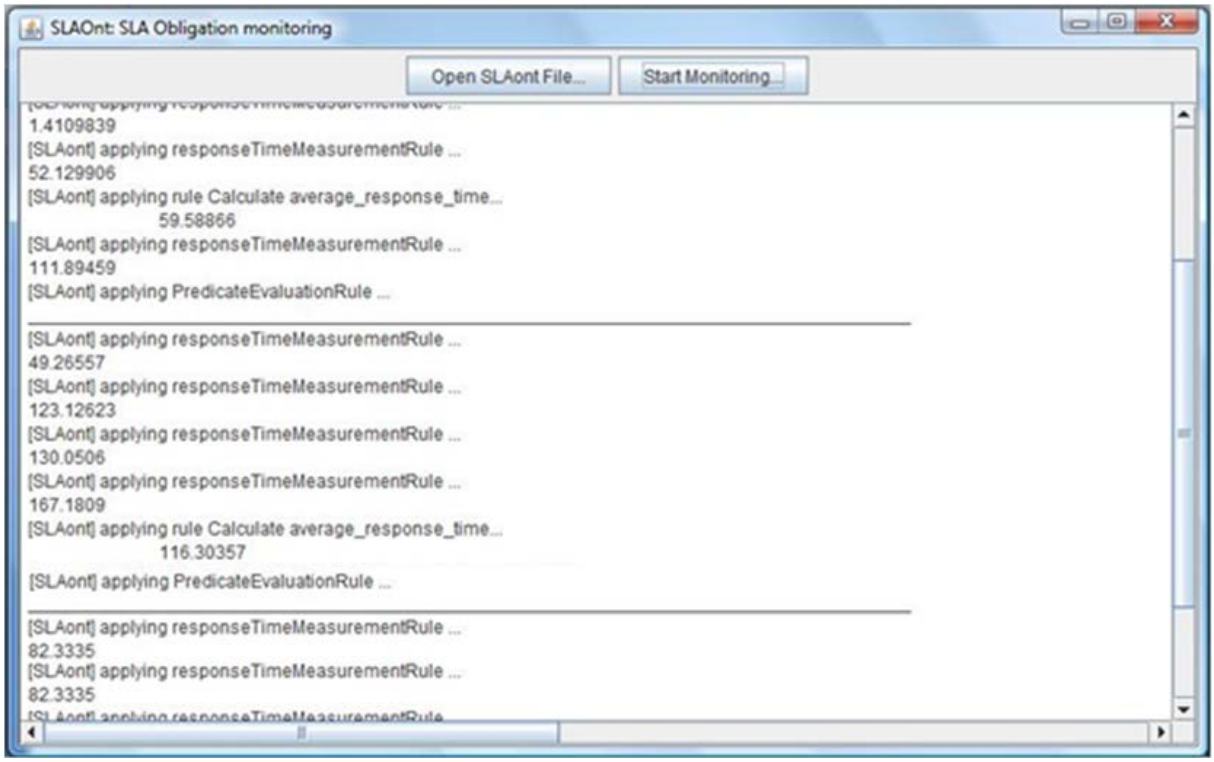

Figure 78 – Déroulement de la surveillance du contrat de notre cas d'étude

Dans notre cas d'étude, la surveillance est effectuée selon des fréquences définies dans le contrat :

- Le service de mesure des métriques élémentaires de QdS (temps de réponse dans notre cas) est exécutée chaque 6000 ms.
- La fréquence de mesure des paramètres composés de QdS (temps de réponse moyen) est 30000 ms.
- La fréquence de vérification des obligations de QdS est la même que le service de mesure de paramètres composés de QdS afin de vérifier si le temps de réponse moyen est inférieur à 100 ms comme demandé par l'agence de voyage.

Il est à noter qu'une synchronisation est définie lors de l'exécution des services de surveillance. Cette synchronisation est assurée par une priorité d'exécution élevée pour le service de mesure des métriques élémentaires, une priorité moyenne pour le service de mesure des paramètres composés de QdS et une priorité basse pour le service de vérification des prédicats définis dans le contrat. Ainsi, ce dernier service sera toujours exécuté après celui de mesure des paramètres composés de QdS.

La Figure 78 illustre le déroulement de la surveillance pour notre cas d'étude avec les résultats d'exécution des différents services précédemment mentionnés.

Pour pouvoir tester tous les cas possibles de la phase de surveillance, nous avons injectés des mesures élevées du temps de réponse pour vérifier le comportement de notre outil lors des violations du contrat (voir Figure 79).

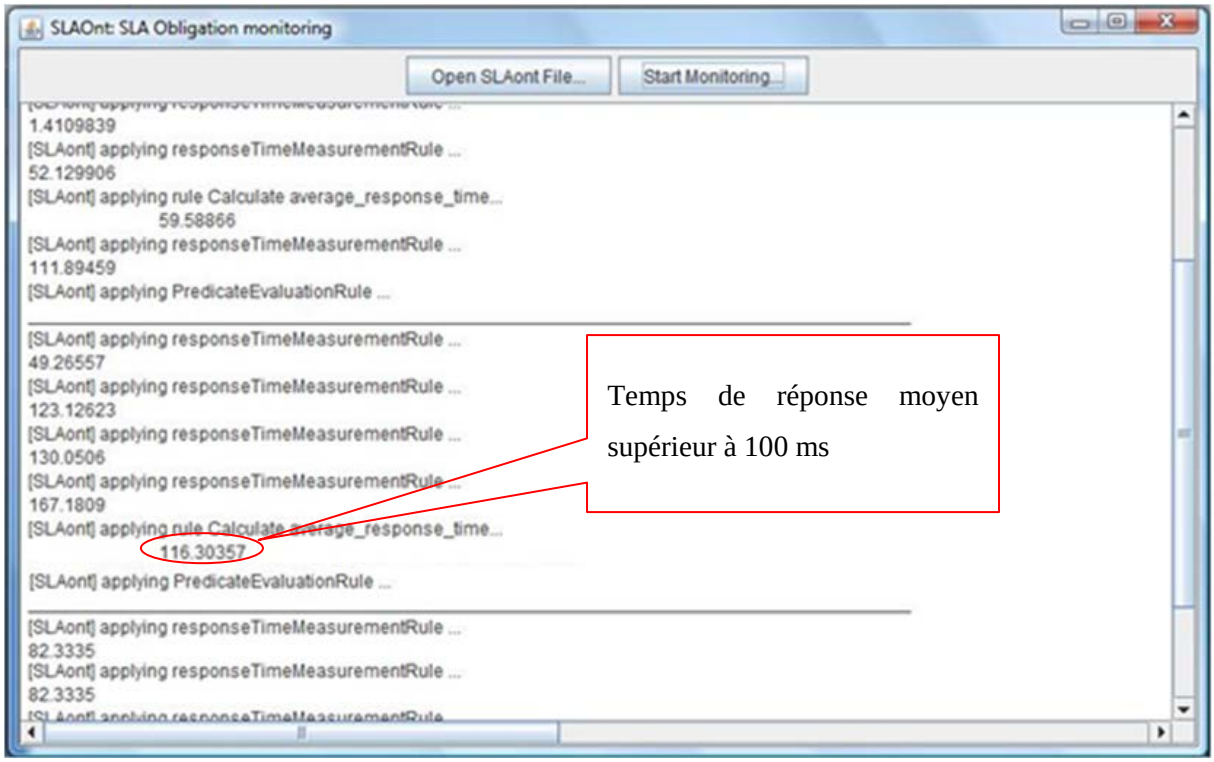

Figure 79 – Mesure causant une violation de contrat

Dans le cas où le contrat est violé, une notification est affichée par notre outil de surveillance (Figure 80). Un email est aussi envoyé automatiquement aux parties signataires pour leur indiquer la violation ainsi que l'application automatique de la pénalité correspondante.

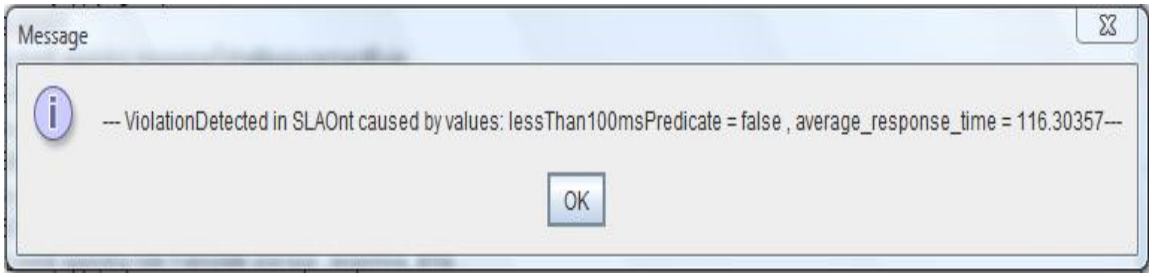

Figure 80 – Message indiquant la violation du contrat

En cas de violation, la pénalité est récupérée du contrat puis appliquée automatiquement. Dans notre cas, la pénalité est une réduction du prix de 10% comme illustré dans la Figure 81. En conséquence, le prix est automatiquement réduit dans le contrat de sa valeur initiale 10 euros à la nouvelle valeur 9 euros.

Figure 81 – Application de la Pénalité indiquée dans le contrat de QdS

V. ETUDE COMPARATIVE ET RESULTATS EXPERIMENTAUX

1. Situation de notre approche d'alignement dans les approches existantes

Après avoir élaboré et validé notre approche de négociation de contrats de QdS par alignement des intentions des clients avec les offres des fournisseurs, nous avons repris les contributions existantes que nous avons analysées dans le chapitre 2 de ce mémoire de thèse afin de les comparer avec notre approche. Le Tableau 2 présente une comparaison selon quatre critères. Le premier s'intéresse à la complexité et à la complétude du modèle de la requête du client. Le second s'intéresse au type du protocole utilisé lors de la négociation. Le troisième étudie ses messages échangés et le quatrième vérifie si l'approche utilisée permet de détecter des QdS composées demandées par le client et calculées ou déduites à partir d'autres QdS élémentaires.

Sur le premier critère, les approches existantes de négociation de QdS utilisent un seul modèle technique complexe pour représenter les demandes des clients et les offres du fournisseur. Ce modèle n'est pas adéquat pour la majorité des clients qui sont généralement non spécialistes dans les nouvelles technologies des sciences de l'information et des télécommunications. Le modèle que nous avons défini dans notre approche *ODACE-SLA* se distingue par sa simplicité et sa richesse sémantique par rapport aux contributions existantes. Il est exprimé dans un langage différent que celui du fournisseur offrant plus de facilité d'utilisation pour les clients surtout qu'il est basé sur la notion d'intention humaine.

Sur le deuxième critère, *ODACE-SLA* offre une solution d'alignement sémantique automatique entre les intentions des clients et les offres des fournisseurs alors que les solutions existantes se basent sur un refus ou une acceptation de l'offre ce qui peut réclamer un effort considérable pour le client afin qu'il puisse comprendre et décider si l'offre est adéquate pour lui.

	Modèle de la requête du client	Type de la négociation	Langage de la négociation	Composition de la QdS
Ws-agreement Framework [36]	Requêtes techniques complexes (même langage que les offres)	Accepter/ Rejeter	XML (basé sur WS-agreement)	Non
BEINGRID [37]	Requêtes techniques complexes (même langage que les offres)	Accepter/ Rejeter	XML (basé sur WS-agreement)	Non
TrustCOM [38]	Requêtes techniques complexes (même langage que les offres)	Accepter/ Rejeter	XML propriétaire	Non
NextGRID [39]	Requêtes techniques complexes (même langage que les offres)	Accepter/ Rejeter	XML propriétaire	Non
ODACE-SLA	Intentions de haut niveau (différentes des offres)	Alignement sémantique et automatique des requêtes et des offres	Détection automatique des correspondances directes et indirectes	Oui

Tableau 2 – Comparaison de notre approche de négociation avec des approches existantes

Sur le troisième critère, les langages existants de négociation ne contiennent pas assez d'éléments sémantiques pour pouvoir automatiser la correspondance des demandes des clients avec les offres des fournisseurs. Dans *ODACE-SLA*, nous avons défini et utilisé des langages sémantiques évolués dérivés d'*OWL* [70] ce qui nous a permis d'obtenir un très haut degré d'automatisation du traitement des demandes des clients qui sont exprimées librement selon leurs propres connaissances.

Sur le dernier critère, contrairement aux approches existantes, *ODACE-SLA* permet d'assurer des agrégations de QdS afin de pouvoir détecter les correspondances sémantiques indirectes entre les intentions des clients et les offres des fournisseurs. Cette agrégation peut se baser sur des calculs mathématiques (comme la mesure du temps de réponse d'un service), des inférences logiques (déduction si un service ne répond pas à cause d'une panne temporaire ou plutôt d'un dysfonctionnement interne) ou même des résultats statistiques (comme le taux de disponibilité d'un service). Elle peut être une agrégation simple ou même composé dans le cas ou il faut faire plusieurs

agrégations pour aboutir à la QdS de haut niveau demandée par le client comme par exemple la moyenne des temps de traitement d'une requête dans un serveur. Cette moyenne se base sur un premier calcul qui est la différence du temps de départ de la réponse du serveur par rapport au temps d'arrivé de sa requête en un premier lieu puis, sur le calcul de la moyenne des mesures effectuées dans un second lieu.

Pour conclure cette mise en situation, *ODACE-SLA* remplit tous les critères principaux qui sont fondamentaux pour l'automatisation de la négociation de QdS entre le client et le fournisseur surtout dans le cas où ces deux derniers ne partagent pas le même degré de connaissances techniques dans le domaine. Il est à noter que notre approche assure une génération complète d'un contrat de QdS en cas de compatibilité des intentions des clients avec les offres des fournisseurs.

2. Situation de notre approche de surveillance dans les approches existantes

Les approches de surveillance des contrats de QdS se basent essentiellement sur le modèle utilisé pour représenter les contrats. Le processus de surveillance sera plus précis et plus complet autant que le modèle du contrat soit plus riche et plus cohérent. Afin de pouvoir situer notre modèle parmi les principales contributions existantes que nous avons présentées dans la section III du chapitre 2 de ce mémoire de thèse, nous avons effectué une étude comparative générale de tous ces modèles selon trois critères (voir Tableau 3). Le premier critère "Portée" illustre le degré de complétude de chaque modèle en listant leurs concepts principaux. Le deuxième critère "Implantation" montre si des exemples concrets ont été élaborés afin de valider ces modèles. Le troisième critère "Facilité d'utilisation automatique" illustre le degré de structuration des informations dans les modèles afin de faciliter leurs interprétations et leurs utilisations automatiques dans des applications de surveillance et d'administration d'accords de qualité de service. Sur le premier critère et selon notre comparaison, *SLAOnt* est le modèle le plus complet en termes de spécifications d'accords et d'exigences de qualité de service. En effet, la majorité des modèles existants se focalisent sur la spécification et la mesure des QdS. Peu de modèles s'intéressent aux accords et aux contrats de QdS. Ceci donne généralement des spécifications vagues et incomplètes pour exprimer les obligations des acteurs impliqués dans les contrats. Notre ontologie *SLAOnt* présente aussi l'avantage de la réutilisation de l'ontologie OWL-S pour la description des services évalués dans les SLA. Sur le deuxième critère, nous avons cherché des exemples concrets utilisant les différents modèles existants. Nous avons aussi essayé de réaliser la même instance de notre exemple *FlightSLA* avec ces modèles. Nous avons rencontré des difficultés diverses avec la plupart d'eux à cause des incomplétudes au niveau des spécifications des obligations et des contraintes de QdS dans le contrat. Par exemple, le modèle de *MOQ* est d'ordre très générique ce qui augmente la distance entre les modèles et leur implantation. Un effort conceptuel considérable

doit être fourni pour pouvoir décrire des contrats concrets avec ce genre de modèles. Même si nous supposons que les instances des modèles existants ont été déjà élaborées, leur maintenance et leur utilisation automatiques dans des applications de gestion de contrats de QdS restent difficiles (troisième critère du Tableau 3). Nous pouvons citer l'exemple de *WSQoS* qui utilise un langage XML spécifique pour la description des métriques ce qui rend le développement d'outils d'analyse automatique de ces modèles difficile et lié à ces langages propriétaires.

	Portée	Implantation	Facilité d'utilisation automatique
OWL-QoS [67]	Métrique, Unité, Fonctions de mesure, Profil de QdS, accords, Acteurs, Service	Présence d'exemples d'implantation partielle	Contraintes de QdS en chaînes de caractères
QoSOnt [69]	Profil du Service, Profil de QdS, Métrique, Unité, Acteurs, Fonctions de mesure, Service	Présence d'exemples d'implantation partielle	Langage spécifique
SL-Ontology [72]	Unité, Métrique, QdS, Services	Présence d'un exemple d'implantation partielle	Modèle bien structuré pour l'établissement des contrats mais pas pour leur suivi
WS-QoS [73]	Métrique, Fonctions, QdS, accords, Acteurs	Ne dispose pas d'implantation OWL	Langage spécifique
FIPA QoS [74]	Quality of Service Description, Rate Value, Probability Value, Transport Protocol Description	Ne dispose pas d'implantation OWL	Langage spécifique
MOQ [78]	Exigences de QdS, Traçabilité, Administration	Fournit un fondement théorique sans implantation	Interprétation ad hoc des prédicats
SLAOnt	SLA, Parties, OWL-S Contextualisation, Obligations, Fonctions, Métriques directives de mesure, Unités, Prédicats, Expressions,	Présence d'un exemple d'implantation complet	Modèle structuré et riche en relations pour des interrogations automatiques faciles

Tableau 3 – Situation de SLAOnt dans les modèles existants

Un effort considérable d'ingénierie doit être mis en œuvre afin de pouvoir exploiter d'une façon automatique les instances de ces modèles dans des applications de gestion de contrats. Enfin, nous pensons que *SLAOnt* présente un compromis entre la complexité du modèle et sa complétude. En effet, l'exemple que nous avons élaboré dans la section IV.2 de ce même chapitre montre la facilité d'implantation du modèle d'une part et sa complétude au niveau de la structuration des informations

décrivant les contrats d'autre part contrairement à certains modèles existants (comme *OWL-QoS*) qui spécifient les prédicats sous la forme de chaînes de caractères. Ainsi, notre outil de surveillance *SLA-Monitoring* se distingue par rapport aux approches existantes (présentées dans la section VI.2 du chapitre I de ce mémoire) par sa capacité de détecter automatiquement les violations des contrats et éventuellement appliquer des pénalités adéquates. En effet, les clauses définies dans les contrats sont exprimées en SWRL ce qui facilite leur évaluation sémantique grâce à des moteurs d'inférence. Notre outil permet en plus de surveiller des paramètres QdS composées basées sur d'autres paramètres élémentaires ce qui n'a pas été réalisé dans l'existant.

3. Résultats expérimentaux

Dans cette section, nous donnons une idée plus précise sur les résultats expérimentaux que nous avons obtenus avec l'exécution de notre réalisation technique dans les deux cas d'études précédemment présentés dans ce chapitre. L'utilisation des ontologies, des règles SWRL et leurs *built-ins* nous ont offert un très haut degré d'automatisation des processus de négociation et de surveillance des accords de QdS. Les résultats obtenus par notre outil de surveillance sont très satisfaisants au niveau de la réactivité et de la précision de tous les services de mesures que nous avons développés. En plus, l'utilisation des règles SWRL nous a permis de définir des prédicats qui permettent de détecter des violations non détectables sans l'utilisation des ontologies et de la sémantique.

Correspondances	Probabilité d'existence d'une correspondance avant la stabilisation	Probabilité d'existence d'une correspondance après la stabilisation
[ClientPrice, Film Name]	0.46	0.36
[FilmType, FilmType]	1	1
[TransferSize, FilmSize]	0.63	0.75
[TransferTime, SearchFilm]	0.41	0.35
[downloadTime, TransferTime]	0.83	0.83
[filmType, SearchFilmService]	0.43	0.50

Tableau 4 – Exemple de correspondances de notre cas d'étude

Pour donner une idée sur les résultats expérimentaux de notre outil d'alignement des intentions des

clients avec les offres des fournisseurs, nous présentons quelques extraits des correspondances détectées avant et après la phase de stabilisation. Nous remarquons une amélioration générale de la probabilité d'existence de ces correspondances, mais, nous avons aussi noté qu'il existe quelques correspondances dont la probabilité doit diminuer alors qu'elle a augmenté par notre algorithme d'alignement comme par exemple dans le cas de la correspondance *[filmType, SearchFilmService]* présentée dans la dernière ligne du Tableau 4.

Ceci est dû principalement aux correspondances adjacentes qui ont des probabilités d'existence élevées. Il est à noter que cette augmentation est minimale et n'affecte pas l'étape d'évaluation des contraintes qui reste l'étape la plus importante de notre approche d'alignement.

Globalement, la précision totale des correspondances résultantes de notre approche d'alignement est très prometteuse (87,5 % dans notre cas d'étude). Cette précision est calculée en utilisant la formule suivante (F4) [120] :

$$Précision = \frac{|TP|}{|TP + FP|} \quad \text{(F4)}$$

TP (true positives) est le nombre des correspondances jugées valides par l'algorithme d'alignement et jugées de même par l'être humain. FP (false positives) est le nombre des correspondances jugées valides par l'algorithme d'alignement alors qu'elles sont jugées invalides par l'être humain. Si nous ne considérons que les correspondances qui concernent la phase d'évaluation des contraintes, nous remarquons que la précision est maximale (égale à 1) après l'étape de stabilisation dans notre cas d'étude. La Figure 82 illustre l'évolution de la précision au cours des itérations de la stabilisation des correspondances.

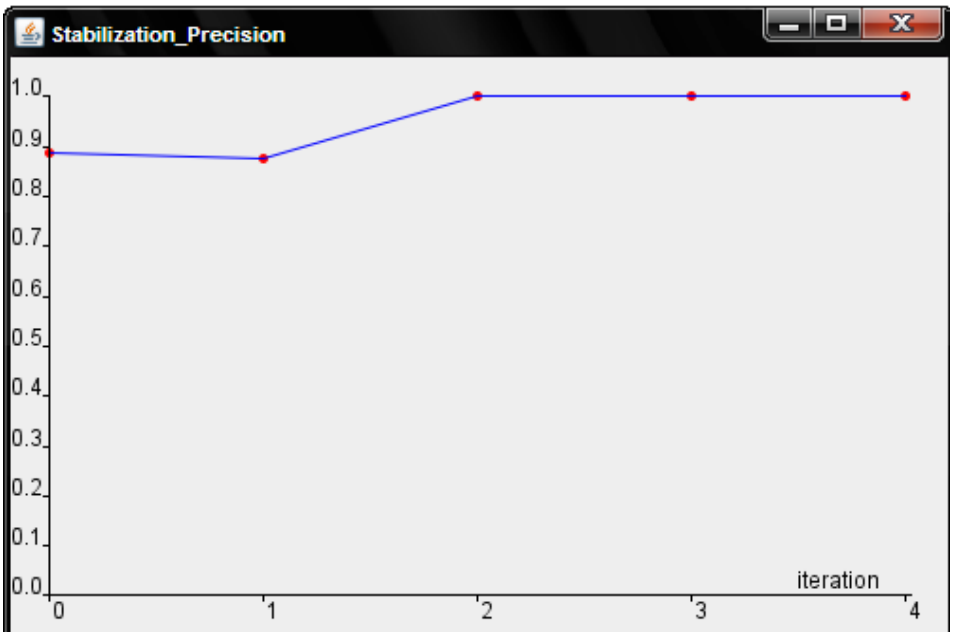

Figure 82 – Evolution de la précision de l'alignement au cours de la stabilisation des correspondances

Les premiers résultats expérimentaux ont montré que notre approche d'alignement consomme beaucoup de temps avec une complexité de l'ordre de O(c*p) où 'c' et 'p' sont respectivement le nombre de termes définis dans l'intention du client et le nombre de termes dans l'offre du fournisseur. Dans notre cas d'étude, le nombre de termes de l'intention du client est 23 alors que le nombre de termes de l'offre du fournisseur est 70 donnant un temps d'exécution global de 25 minutes et 13 secondes sur un ordinateur portable équipé d'un processeur AMD Turion 64 cadencé à 1.6 GHz avec 1 Go de mémoire vive. Ceci est principalement dû au coût de la communication de notre algorithme d'alignement avec le module *WordNet::Similarity* [103] qui permet de mesurer la similarité sémantique entre les termes du client et ceux du fournisseur. En effet, ce module ne peut pas accepter des requêtes parallèles pour calculer la similarité sémantique de plusieurs paires de termes en même temps. Il envoie rapidement un message *Busy* si nous n'injectons pas un délai entre les requêtes. Après quelques changements au niveau de la communication avec ce module développé en langage *Perl*, nous avons réussi à réduire le temps d'exécution de toutes les étapes de l'alignement à 5 minutes et 50 secondes. Nous avons obtenu ce résultat en modifiant notre outil pour détecter les paires des termes qui doivent êtres sémantiquement comparés dans une première sous étape et pour l'évaluation sémantique effective grâce à *WordNet::Similarity* dans une seconde sous étape. En effet, nous avons remarqué que le module *Perl* de *Ted Person* répond plus rapidement si toutes les paires de termes à évaluer sont données à ce module d'un seul coup dans un ficher texte. Nous avons également stocké les mesures effectuées par ce module dans un cache avec une fonction de hachage pour accélérer leur extraction par notre outil d'alignement. Si toutes les valeurs nécessaires existent dans le cache, toutes les étapes du processus d'alignement prennent moins de 3 minutes (2 minutes et 38 secondes) pour notre cas d'étude sur le même ordinateur portable (61 secondes sur un PC de bureau équipé d'un processeur I5 cadencé à 2.66 GHz et de 4 Go de mémoire vive). Nous pensons que ce temps d'exécution pourrait être encore considérablement réduit si nous remplaçons le module Perl *WordNet::Similarity* par un outil de mesures de similarité sémantique entièrement développé en Java. Actuellement, nous n'avons pas réussi à trouver un tel outil qui soit aussi complet et surtout aussi précis que celui de *WordNet::Similarity*.

Notre approche d'alignement se base sur un ensemble de seuils qui permettent de définir si la correspondance est valide ou non. Par exemple si la probabilité d'existence d'une correspondance après la phase de stabilisation est supérieure à 0.6, alors elle est considérée comme valide. Nous avons laissé le choix de ce seuil pour l'expert commercial du fournisseur. Par exemple, si cet expert peut négocier ces offres et leurs prix avec une grande marge, il peut diminuer ce seuil pour accepter plus de clients, sinon, il peut l'augmenter pour n'accepter que les clients qui cherchent exactement ce qu'il offre. Pour aider l'expert commercial dans le choix de ce seuil, nous avons mené plusieurs expériences en variant ce seuil pour voir son influence sur la qualité de l'alignement. Cette qualité est communément mesurée à l'aide de la précision (voir formule (F4) de cette section) et le rappel calculé

selon la formule (F5) [121] :

$$Rappel = \frac{|TP|}{|TP + FN|} \quad \text{(F5)}$$

Où FN représente les correspondances jugées invalides par le processus d'alignement alors qu'il est jugé valide par les êtres humains.

Il est à noter que plus le rappel et la précision soient grands, plus l'alignement est considéré de meilleure qualité. En conséquence, nous avons eu recours à une formule très utilisée dans le domaine de la recherche d'information qui combine l'utilisation du rappel et de la précision ensemble pour évaluer l'alignement. Cette formule (F6) est la mesure F [121] :

$$F = \frac{2 \times Précision \times Rappel}{Précision + Rappel} \quad \text{(F6)}$$

Ainsi, l'alignement sera meilleur si la valeur de la mesure F est plus grande. En variant les seuils de validité des correspondances, nous avons calculé cette mesure. Le maximum de F correspond au meilleur seuil à prendre pour l'alignement. Nous nous sommes basés sur un cache d'évaluation humaine effectuée par 7 chercheurs de notre unité de recherche pour calculer cette précision. Nous avons réutilisé ce cache pour développer un outil d'évaluation de l'alignement qui permet de détecter automatiquement le meilleur seuil en se basant sur la mesure F.

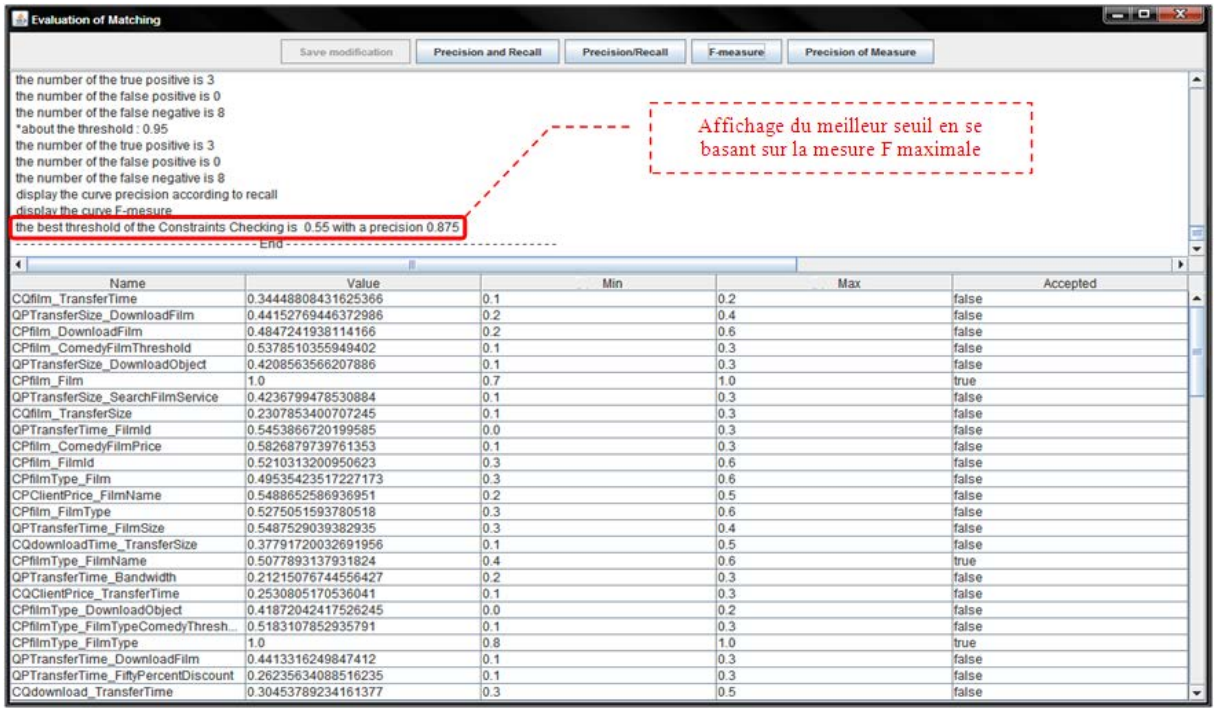

Figure 83 – Interface de note outil d'évaluation de l'alignement

La Figure 83 présente l'interface principale de cet outil que nous avons intégré dans notre outil d'alignement des intentions des clients avec les offres des fournisseurs. La partie inférieure de cette interface illustre le contenu du cache des évaluations humaines des correspondances alors que la partie

supérieure illustre le calcul de la mesure F en variant les seuils de validité des correspondances entre 0 et 1 avec un pas de 0,05. A la fin de ce calcul, le meilleur seuil est affiché sur cette interface. Ce meilleur seuil est aussi automatiquement intégré dans le processus d'alignement. La valeur du meilleur seuil pour notre cas d'étude est 0.55 comme illustré dans le rectangle rouge de la Figure 83.

VI. CONCLUSION

Dans ce chapitre, nous avons présenté les deux outils *ODACE-SLA* et *SLAOnt-Monitoring* que nous avons réalisés pour valider nos approches d'alignement des intentions des clients avec les offres des fournisseurs et de surveillance de contrats de QdS. Nous avons également présenté deux cas d'études où nous avons utilisé nos deux outils. Ces cas d'études nous ont permis d'effectuer des analyses expérimentales qui ont validé nos contributions théoriques. Dans le cadre du cas d'étude de négociation, pour pouvoir utiliser notre outil d'alignement, nous avons fourni deux fichiers OWL représentant l'intention du client et l'offre du fournisseur disponibles dans les liens suivants [122], [123]. Pour le cas d'étude de la surveillance, il suffit de fournir le contrat à surveiller disponible dans [124].

CONCLUSIONS ET PERSPECTIVES

Les contrats de qualité de service *(SLA : Service Level Agreements)* sont devenus de plus en plus complexes avec l'augmentation importante du nombre de fournisseurs et de leurs offres dans les architectures orientées services. Ces paramètres rendent la compréhension de ces contrats très difficile pour les clients qui n'ont pas les outils pour exprimer librement leurs exigences en utilisant leurs simples connaissances et leur propre langage. Dans ce travail, nous avons présenté une approche d'alignement entre les intentions du client et les offres du fournisseur pour réduire les écarts de connaissances entre eux. Nous avons commencé par modéliser les intentions du client et les offres du fournisseur en utilisant les ontologies pour faciliter l'élaboration de notre approche d'alignement automatique d'exigences de qualité de service. Cette approche se base sur quatre étapes principales. La première étape consiste à générer des correspondances entre les termes du client et les termes du fournisseur en leurs attribuant des mesures de similarité selon leur équivalence sémantique. La deuxième étape raffine et corrige ces mesures. Dans la troisième étape, nous évaluons les contraintes du client par rapport aux offres du fournisseur selon les correspondances générées lors des deux étapes précédentes. Enfin, dans la quatrième étape nous générons un contrat de QdS (*SLA*) en cas de compatibilité. Le contrat généré est une ontologie selon la structure *SLAOnt* que nous avons élaborée dans le cadre de cette thèse. Cette structure se distingue par sa complétude et sa richesse sémantique par rapport aux modèles existants de SLA surtout au niveau des paramètres composés de qualité de service.

Suite à l'absence d'une approche complète et automatique de surveillance de contrats de qualité de service, nous avons proposé aussi une approche sémantique de suivi et de vérification des obligations définies dans les contrats générés lors de la phase de négociation. Notre processus de surveillance se base principalement sur notre nouveau modèle de contrats de qualité de service *SLAOnt*. En effet, notre approche de surveillance consiste à extraire les éléments constituants les instances de *SLAOnt* et à générer automatiquement des services de mesure des métriques (simples et composées) et des obligations de QdS qui sont définies dans le contrat. Les mécanismes d'observation, d'analyse et de détection de violations de cette approche bénéficient de la même capacité sémantique de notre approche de négociation. En effet, la solution nous que avons définie se base sur un raisonnement par inférence qui nous permet de détecter des violations et même des dégradations non détectables sans sémantique. Par ailleurs, l'utilisation de la puissance sémantique des ontologies nous a permis d'obtenir une haute précision et un haut degré d'automatisation de la phase de négociation entre les intentions du client et les offres du fournisseur d'une part, et de la phase de surveillance de contrat de QdS d'autre part.

Pour mettre en œuvre nos contributions, nous avons développé un premier outil d'alignement nommé *ODACE SLA* qui prend en entrée deux ontologies représentant l'intention du client et les offres du fournisseur pour générer une première version d'un contrat de qualité de service en cas de compatibilité entre ces intentions et ces offres. En outre, nous avons développé un outil de surveillance nommé *SLAOnt Monitoring* qui prend en entrée le contrat établi et génère les services nécessaires pour

assurer la surveillance des obligations du contrat. En cas de violation, les parties signataires sont notifiées et des pénalités sont automatiquement encourues. Nous avons utilisé ces deux outils entièrement développés en Java dans deux cas d'études. Le premier concerne un service de téléchargement de films d'une bibliothèque numérique de vidéos. Le deuxième traite le cas d'un service de réservation de billets d'avion. Ces deux cas d'études nous ont permis de valider nos approches d'alignement d'objectifs sémantiques de QdS et de surveillance des contrats de QdS. Les études expérimentales que nous avons menées sur ces deux cas nous ont confirmé la haute précision et le haut degré d'automatisation des phases de négociation et de surveillance de contrats de QdS.

Comme bilan de cette thèse, nous croyons que notre contribution est bénéfique d'une part, aux clients en leurs proposant un moyen qui leurs permet d'exprimer leurs exigences et leurs besoins d'une façon simple en utilisant leurs propres langages et connaissances. D'autre part, elle est aussi bénéfique aux fournisseurs en leur facilitant l'automatisation de l'analyse des demandes des clients. Elle permet aussi de générer et de surveiller des contrats de qualité de service en cas de compatibilité des offres des fournisseurs avec ces demandes. Ce processus de surveillance informe les deux parties en cas de violation en spécifiant les clauses qui n'ont pas été respectées tout en appliquant automatiquement les pénalités. Dans cette thèse, nous nous sommes intéressés aux QdS de haut niveau qui concernent les couches supérieures du modèle OSI mais nos approches de négociation et de surveillance restent applicables sur les QdS de bas niveau.

Comme perspective à ce travail, nous pensons à élaborer une approche qui permet de transformer automatiquement les demandes des clients saisies en texte libre vers la structure des intentions que nous avons définie dans cette thèse. Nous comptons aussi intégrer des règles d'inférence supplémentaires dans ces intentions pour contôler les valeurs demandées par le client. Ceci évitera au client d'attendre l'exécution de tout le processus d'alignement pour réaliser qu'il a demandé des prestations qui sont très éloignées des offres du marché. Notre approche d'alignement peut aussi être étendue pour aider les clients à classifier et à sélectionner les offres les plus proches de leurs intentions en attribuant des scores pour chaque offre selon son degré de proximité sémantique de la demande du client. Enfin, nous comptons déployer nos architectures de négociation et de surveillance de QdS sur des environnements parallèles comme des grilles de calcul ou des nuages informatique (*cloud computing*) pour pouvoir gérer le passage à l'échelle des outils que nous avons réalisés en termes de temps d'éxécution et de charge d'utilisation.

BIBLIOGRAPHIE

[1] Baligand F. Une Approche Déclarative pour la Gestion de la Qualité de Service dans les Compositions de Services. Ecole Nationale Supérieure des Mines de Paris. Juin 2008.

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

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

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

[5] Geib J.M., Gransart C., Merle P. CORBA : des concepts à la pratique. Masson, 1997.

[6] Distributed Component Object Model From Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/Distributed_Component_Object_Model. Dernière visite le 11 juin 2010.

[7] Web Services Description Language (WSDL) 1.1 W3C Note 15 March 2001. http://www.w3.org/TR/wsdl. Dernière visite le 11 juin 2010.

[8]UDDI Spec TC. OASIS. http://www.uddi.org/pubs/DataStructure-V2.03-Published-20020719.htm. Dernière visite le 28 septembre 2010.

[9] SOAP version 1.2. http://www.w3.org/TR/soap12-part1/SOAP. Dernière visite le 25 juin 2010.

[10] The International Telecommunication Union (ITU) standard X.902, Information technology – Open distributed processing - Reference Model.

[11] Vogel A., Kerhervé B., von Bochmann G., and Gecsei J. Distributed multimedia and qos: A survey. IEEE MultiMedia, 2(2):10–19, 1995.

[12] Ben Halima R. Conception, implantation et expérimentation d'une architecture en bus pour l'auto-réparation des applications distribuées à base de services web. Thèse de doctorat. Université Toulouse III - Paul Sabatier et la Faculté des Sciences Économiques et de Gestion – Sfax. Juin 2009.

[13] http://bounatelecom.wordpress.com/2009/07/22/service-level-agrement-ou-convention-de-service. Dernière visite le 28 septembre 2010.

[14] Debusmann M., Kroger R., and Geihs K. "Unifying Service Level Management Using an MDA-based Approach". IEEE Network Operations and Management Symposium, pages 801-814, 2004.

[15] Keller A. and Ludwig H. The WSLA Framework : Specifying and Monitoring Service Level Agreements for Web Services. Journal of Network and Systems Management, 11(1), 2003.

[16] W. Sun et al., "The Role of XML in Service Level Agreements Management", Network Management Research Center, Beijing Jiaotong Univeristy, IEEE, 2005.

[17] Keller A. and Ludwig H. - Defining and Monitoring Service Level Agreements for dynamic e-Business, 2002.

[18] Sun W. et al. The Role of XML in Service Level Agreements Management. Network Management Research Center, Beijing Jiaotong University, IEEE, 2005.

[19] Tosic V., Patel K., and Pagurek B. WSOL - Web Service Offerings Language. In International Workshop onWeb Services, E-Business, and the SemanticWeb (CAiSE/WES). Springer-Verlag, 2002.

[20] Skene J., Lamanna D., and Emmerich W. SLAng : A Language for Service Level Agreements. In Proc. of Workshop on Future Trends in Distributed Computing Systems. IEEE Computer Society, 2002.

[21] Cordeiro J., Hammoudi S. and Filipe J. Towards Dynamic Service Level Agreement Negotiation: An Approach Based on WS-Agreement Web Information Systems and Technologies 4th International Conference, WEBIST 2008, Funchal, Madeira, Portugal, May 4-7, Revised Selected Papers.2008.

[22] http://tecfa.unige.ch/guides/tie/html/xml-schema/xml-schema.html. Dernière visite le 25 mai 2010.

[23] Dan A. and Davis D. and Robert Kearney and Alexander Keller and Richard P. King and Dietmar Kuebler and Heiko Ludwig and Mike Polan and Mike Spreitzer and Alaa Youssef - Web services on demand: WSLA-driven automated management. IBM Systems Journal, Volume 43. 2004.

[24] Touseau L. Accords de niveau de service dans les plateformes dynamiques à services. Rapport de Master 2 Recherche Systèmes et Logiciels. Université Joseph Fourier (Grenoble 1). Juin 2005.

[25] IBM Research. http://www.research.ibm.com/. Dernière visite le 11 juin 2010.

[26] Reid G. Smith. The Contract Net Protocol: high-level communication and control in a distributed problem solver. IEEE Transactions on Computers, C-29(12):1104-1113, December 1980.

[27] Pruitt D. Negotiation Behavior. New York: Academic Press. 1981.

[28] Sarangan V. and Chen J.-C., "Comparative study of protocols for dynamic service negotiation in next generation Internet," IEEE Communications Magazine, (Network & Service Management Series), vol. 44, no. 3, pages 151-156, Mar. 2006.

[29]Tequila Project, «Traffic Engineering for Quality of Service in the Internet, at Large Scale», accessible via: http://www.ist-tequila.org. Dernière visite le 28 septembre 2010.

[30]Tequila Project, «Final Architecture, Protocol and Algorithm Specification », Livrable Tequila D 3.4 Part B, Octobre 2003.

[31] Thi Mai Trang N., Boukhatem N., Pujolle G., «COPS-SLS Usage for dynamic Policy-Based QdS Management over Heteregenous IP Networks », Proc in IEEE Network, Volume 17, Issue 3, Page(s): 44 - 50, May-June 2003.

[32] Durham D., Boyle J., Cohen R., Herzog S., Rajan R., Sastry A., " RFC 2748 : The COPS (Common Open Policy Service) Protocol", Request for Comments, IETF, Janvier 2000.

[33] Chen J.C. et al., "Design and implementation of Dynamic Service Negotiation Protocol (DSNP)," Elsevier J. Computer Communications, Jun. 2006.

[34] Manner J., Karagiannis G., McDonald A., "NSLP for Quality-of-Service Signaling", draft-ietf-nsis-QdS-nslp-16.txt, IETF, Feb 2008.

[35] Hancock R., Karagiannis G., Loughney J., Van den Bosch S., "RFC 4080 : Next Steps in Signaling (NSIS): Framework", Request for Comments, IETF, June 2000.

[36] Fraunhofer Institute SCAI. wsag4j organizational pom. http://packcs-e0.scai.fraunhofer.de/mss-project/wsag4j/project-info.html. Dernière visite le 28 septembre 2010.

[37] Dimitrakos, T., Martrat, J., Wesner, S. (Eds.). A selection of common capabilities validated in real-life business trials by the BEinGRID consortium. XV, p. 210 Hardcover: Springer. 2010.

[38] The TrustCoM Project, Priority IST-2002-2.3.1.9. <http://www.eu-trustcom.com>. 30 Apr. 2006.

[39] Francis, W. The NextGRID Final Report (Project Final Report). Edinburgh, United Kingdom: The University of Edinburgh. 2008.

[40] Michael S. and Markus S. A Matchmaking Component for the Discovery of Agreement and Negotiation Spaces in Electronic Markets. Group Decision and Negotiation, 11, 2002.

[41] Rana, O., Warnier, M., Quillinan, T. B., and Brazier, F. Monitoring and Reputation Mechanisms for Service Level Agreements. In Proceedings of the 5th international Workshop on Grid Economics and Business Models (Las Palmas de Gran Canaria, Spain, August 26 - 26, 2008). J. Altmann, D. Neumann, and T. Fahringer, Eds. Lecture Notes In Computer Science, vol. 5206. Springer-Verlag, Berlin, Heidelberg, 125-139. DOI= http://dx.doi.org/10.1007/978-3-540-85485-2_10. 2008.

[42] Ludwig H., Keller A., Dan A., Richard P. K., and Franck R. WSLA language specification, version 1.0. Technical report, IBM Corporation, http ://www.research.ibm.com/wsla/, January 2003.

[43] Keller A., Kar G., Ludwig H., Dan A., and Hellerstein J.L. Managing Dynamic Services: A Contract Based Approach to a Conceptual Architecture. In Proc. Of the 8th IEEE/IFIP Network Operations and Management Symposium (NOMS 2002), Florence, Italy, KL02. 2002.

[44] Sahai A., Machiraju V., Sayal M., van Moorsel A., and Casati F. Automated sla monitoring forweb services. In 13th IFIP/IEEE International Workshop on Distributed Systems : Operations andManagement, DSOM, pages 28–41, Montreal, Canada. Springer-Verlag. 2002.

[45] Sahai A., Durante A., andMachiraju V. Towards automated sla management for web services. Technical report, Machiraju V., Sahai A., and vanMoorsel A. Web services management network: An overlay network for federated service management. Technical report, 2002.

[46] Web Services Flow Language (WSFL 1.0). Edited by Prof. Dr. Frank Leymann (Distinguished Engineer; Member IBM Academy of Technology, IBM Software Group). May 2001

[47] Ludwig H., Dan A., and Kearney B. Cremona: An Architecture and Library for Creation and Monitoring of WS-Agreements. In International Conference on Service Oriented Computing (ICSOC). ACM Press, 2004.

[48] Michlmayr A., Rosenberg F., Leitner P., Dustdar S.: "Comprehensive QoS Monitoring of Web Services and Event-Based SLA Violation Detection", Proceedings of the 4th International Workshop on Middleware for Service Oriented Computing (MW4SOC'09 @ Middleware'09), Urbana-Champaign, Illinois, USA, December 2009.

[49] Chau, T., Muthusamy, V., Jacobsen, H.-A., Litani, E., Chan, A., and Coulthard, P. Automating SLA Modeling. In Proceedings of the 2008 Conference of the Center for Advanced Studies on Collaborative Research (CASCON'08), ACM, pages 126–143. DOI:10.1145/1463788.1463802. 2008.

[50] Lodi G., Panzieri F., Rossi D., and Turrini E. SLA-Driven Clustering of QoS-Aware Application Servers. IEEE Transactions on Software Engineering, 33(3):186-197, 2007.

[51] Raimondi F., Skene J., and Emmerich W. Efficient online monitoring of web-service SLAs. In Proc. of the 16th ACM SIGSOFT Int. Symposium on Foundations of Software Engineering (SIGSOFT'08/FSE-16), 2008.

[52] Sahai A., Machiraju V., Sayal M., vanMoorsel A. P. A., and Casati F. Automated SLA Monitoring for Web Services. In Proc. of International Workshop on Distributed Systems : Operations and Management (DSOM), volume 2506 of Lecture Notes in Computer Science, pages 28–41. Springer, 2002.

[53] Jin H. and Wu H. Semantic-enabled Specification for Web Services Agreement. International Journal of Web Services Practices, 1(1-2), 2005.

[54] Gruber T., « A translation approach to portable ontology specifications », Knowledge Acquisition, vol. 5, pages 199-220, 1993.

[55] Neches R., Fikes R., Finin T., Gruber T., Patil R., Senator T., Swartout W. R., « Enabling Technology for Knowledge Sharing », Artificial Intelligence Magazine. pages 36-56, 1991.

[56] Définition de wikipedia, http://fr.wikipedia.org/wiki/Ontologie_(informatique). Dernière visite le 6 septembre 2010.

[57] DAML-S: Web Service Description for the Semantic Web. In The First International Semantic Web Conference (ISWC), pages 348–363. 2003.

[58] Fensel D., Hendler J., Lieberrman H. and Wahlster W. (eds) (2002). Spinning the Semantic Web : Bringing the World Wide Web to Its Full Potential, The MIT Press.

[59] Mcilraith S., Son T.C., and Zeng H. Semantic Web Services. IEEE Intelligent Systems. Special Issue on the Semantic Web, 16(2):46–53. 2001.

[60] Charlet J., Bachimont B., Troncy R., Le Web sémantique, vol. 4 of Hors série de la revue Information - Interaction - Intelligence, Cépaduès, chapter 4 : Ontologies pour le Web sémantique, pages 7-20, 2005.

[61] Aussenac-Gilles N., Biébow B., et Szulman S. « Modélisation du domaine par une méthode fondée sur l'analyse de corpus» in Actes des 9e journées francophones d'Ingénierie des Connaissances IC 2000,Toulouse, France, 2000.

[62] Recommendation W3C OWL, 2004. http://www.w3.org/TR/owl-ref/. Dernière visite le 09 juillet 2010.

[63] Connolly D. et al. DAML+OIL Reference Description. http://www.w3.org/TR/daml+oil-reference. Dernière visite le 28 septembre 2010.

[64]World Wide Web Consortium. Resource Description Framework (RDF). http://www.w3.org/RDF/. Dernière visite le 09 juillet 2010.

[65] Horrocks I., Patel-Schneider P. F., Boley H., Tabet S., Grosof B. et Dean M, SWRL: A Semantic Web Rule Language Combining owl and RuleML, may 2004.

[66]SWRL: A Semantic Web Rule Language Combining OWL and RuleML. http://www.w3.org/Submission/SWRL/. Dernière visite le 11 juin 2010.

[67] Bu-Sung Lee C. "QdS Measurement Issues with DAML-QdS". IEEE InterChen Zhou, Likang-Tien national Conference on e-Business Engineering (ICEBE'05) pages 395-403. 2005.

[68]OWL-S. An OWL-based Web service ontology. <http://www.w3.org/Submission/2004/07/> Dernière visite le 28 septembre 2010.

[69] Dobson G., Lock R., Sommerville I. "QoSOnt: a QdS Ontology for Service-Centric System", EUROMICRO Conference on Software Engineering and Advanced Applications, Porto, Portugal, Aug. 2005.

[70]MeGuinness D.L., Harmelen F. van. OWL Web Ontology Language Overview. http://www.w3.org/TR/owl-features/. Dernière visite le 28 septembre 2010.

[71] OWL-S. An OWL-based Web service ontology. http://www.w3.org/Submission/2004/07/. Dernière visite le 28 septembre 2010.

[72] Bleul S., Weise T., Geihss K. "An Ontology for Quality-Aware Service Discovery". Special Edition Editorial: Engineering Design and Composition of Service-Oriented Applications, Computer Systems Science & Engineering, Volume 5, Number 21 – 2006.

[73] Tian M., Gramm A., Ritter H. and Schiller J. "Efficient Selection and Monitoring of QoS aware Web services with the WSQoS Framework". IEEE/WIC/ACM International Conference on Web Intelligence (WI'04), Beijing, China, 2004.

[74] Foundation for Intelligent Physical Agents "FIPA Quality of Service Ontology Specification", Geneva, Switzerland, Nov. 2002, http://www.fipa.org/specs/fipa00094/XC00094.html. Dernière visite le 28 septembre 2010.

[75] FIPA specifications. http://www.fipa.org/specs/fipa00027/. Dernière visite le 10 juin 2010.

[76] FIPA specifications. http://www.fipa.org/specs/fipa00035/index.html. Dernière visite le 10 juin 2010.

[77] OSI model From Wikipedia, the free encyclopedia. http://en.wikipedia.org/wiki/OSI_model. Dernière visite le 10 juin 2010.

[78] Kim HM., Sengupta A. and Evermann J. "MOQ: Web Services Ontologies for QdS and General Quality Evaluations", European Conference on Information Systems, Regensburg, Germany. May 2005.

[79] Searle J. Intentionality. Cambridge: Cambridge University Press. 1983.

[80] Kanso H. Vers la reconnaissance des intentions de communications. Application au contenu d publications scientifiques. Thèse de doctorat. Institut de recherche en informatique de Toulouse. Juin 2009.

[81] Tazi S. Evrard F., Intentional Structures of Documents, ACM Hypertext' 01 proceedings, University of Arhus, Arhus, Denmark. August 14-18, 2001.

[82] Bach T.L., Dieng-Kuntz R., Gandon F., « On Ontology Matching Problems - for Building a Corporate Semantic Web in a Multi-Communities Organization ». ICEIS (4): pages 236-243, 2004.

[83] RAHM E. and BERNSTEIN P. A survey of approaches to automatic schema matching. The VLDB Journal, 10(4):334–350, 2001.

[84] Mellal N. Réalisation de l'interopérabilité sémantique des systèmes, basée sur les ontologies et les flux d'information. Polytech'Savoie. Décembre 2007.

[85] Naymoun C, IL-Yeol S. and Hyoil H. A survey on ontology mapping. SIGMOD Rec., 35(3):34–41, September 2006.

[86] Noy N. and Musen M. Prompt: Algorithm and tool for automated ontology merging and alignment. In Procs.AAAI-2000, 2000.

[87] Corcho O. and Gomez-Perez A. Solving Integration Problems of E-Commerce Standards and Initiatives through Ontological Mappings. IJCAI-01 Workshop on Ontologies and Information Sharing, pages 131–140, 2001.

[88] Klein M. Combining and relating ontologies : an analysis of problems and solutions. Proc. of the IJCAI-01 Workshop on Ontologies and Information Sharing, Seattle, USA, August 4-5, pages 53-62, 2001.

[89] Aleksovski Z., Klein M., Kate W. T., Harmelen F. V., « Matching Unstructured Vocabularies using a Background Ontology », Proceedings of the 15th International Conference on Knowledge Engineering and Knowledge Management, Hong Kong, China, pages 182-197, 2006.

[90] Stuckensschmidt H., Harmelen F. V., Serafini L., Bouquet P. and Giunchiglia F. « Using C-OWL for the Alignment and Merging of Medical Ontologies », Proceedings of the First Int. WS. on Formal Biomedical K. R. , 2004.

[91] Van HageW., Katrenko S. and Schreiber G. « A Method to combine Linguistic Ontology-Mapping Techniques », Proceedings of International Semantic Web Conference, Sardinia, Italy, pages 732-744, 2005.

[92] Shvaiko P. and Euzenat J. « A Survey of Schema-based Matching Approaches », Journal on Data Semantics, pages 146-171. 2005.

[93] Noy N.F. et Musen M.A.: The prompt suite: interactive tools for ontology merging and mapping. Int. J. Hum.-Comput. Stud. 59(6) 983-1024, 2003.

[94] Giunchiglia F., Shvaiko P. and Yatskevich M. S-match: an algorithm and an implementation of semantic matching. In Y. KALFOGLOU, M. SCHORLEMMER, A. SHETH, S. STAAB et M. USCHOLD, réds. , Semantic Interoperability and Integration, number 04391 in Dagstuhl Seminar Proceedings. Internationales Begegnungs und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2005.

[95] Maedche A., Motik B., Silva N. and Volz R. MAFRA - A Mapping Framework for Distributed Ontologies. In Proc. of 13th European Conference on Knowledge Engineering and Knowledge Management (EKAW), Siquenca, Spain, 2002.

[96] Doan A., Madhavan J., Domingos P. et Halevy A. Ontology matching: A machine learning approach. 2003.

[97] Shvaiko P., Giunchiglia F. et Yatskevich M., "Semantic matching with S-Match" in Semantic Web Information management, a model based perspective, Berlin: Springer, p. 183-203, 2010.

[98] Ehrig M. and Staab S. Qom quick ontology mapping. In International Semantic Web Conference (ISWC2004), Japan, November 2004.

[99] Thanh LE B., Dieng-kuntz R. et Gandon F. On ontology matching problems - for building a corporate semantic web in a multi-communities organization. In ICEIS (4), pages 236–243, 2004.

[100] Shvaiko P. and Euzenat J. Tutorial on Schema and Ontology Matching. ESWC'05. Universita Degli Studi Di Trento et INRIA Rhônes Alpes. pages 1-71. 2005.

[101] Thành Lê B. Construction d'un Web sémantique multi-points de vue. École des Mines de Paris à Sophia Antipolis. Octobre 2006.

[102] Kanso H., Soulé-Dupuy C., Tazi S. Representing Author's Intentions of Scientific Documents. Dans: International Conference on Enterprise Information Systems (ICEIS 2007), Funchal, Protugal, Vol. 3, INSTICC Press, pages 489-492, Juin 2007.

[103] Overview: T. Pedersen, and S. Patwardhan, and J. Michelizzi, "WordNet::Similarity - Measuring the Relatedness of Concepts" In: Proceedings of Fifth Annual Meeting of the North American Chapter of the Association for Computational Linguistics (NAACL-04), pages 38-41, Boston, May 2004.

[104] Hirst and St-Onge D. Lexical chains as representations of context for the detection and correction of malapropisms. In C. Fellbaum, editor, WordNet: An electronic lexical database, pages 305–332. MITPress. 1998.

[105] Jess, the Rule Engine for the JavaTM Platform. http://www.jessrules.com/. Dernière visite le 06 septembre 2010.

[106]The Perl Programming Language. http://www.perl.org/. Dernière visite le 28 août 2010.

[107] Wu and M. Palmer. Verb semantics and lexical selection. In 32nd Annual Meeting of the Association for Computational Linguistics, pages 133–138, Las Cruces, New Mexico. 1994.

[108] Leacock and M. Chodorow. Combining local context and WordNet similarity for word sense identification. In C. Fellbaum, editor, WordNet: An electronic lexical database, pages 265–283. MIT Press. 1998.

[109] Pedersen T., Patwardhan S., & Michelizzi J. WordNet::Similarity - Measuring the Relatedness of Concepts. In: Proceedings of Fifth Annual Meeting of the North American Chapter of the Association for Computational Linguistics (pages 38-41). Boston: Association for Computational Linguistics. 2004.

[110] Resnik, P. Using information content to evaluate semantic similarity in a taxonomy. In Proceedings of the 14th international Joint Conference on Artificial intelligence - Volume 1 (Montreal, Quebec, Canada, August 20 - 25, 1995). C. S. Mellish, Ed. International Joint Conference On Artificial Intelligence. Morgan Kaufmann Publishers, San Francisco, CA, 448-453. 1995.

[111] Jiang and D. Conrath. Semantic similarity based on corpus statistics and lexical taxonomy. In Proceedings on International Conference on Research in Computational Linguistics, pages 19–33, Taiwan. 1997.

[112] Lin. An information-theoretic definition of similarity. In Proceedings of the International Conference on Machine Learning, Madison, August. 1998.

[113] Resnik. Using information content to evaluate semantic similarity in a taxonomy. In Proceedings of the 14th International Joint Conference on Artificial Intelligence, pages 448–453, Montreal, August. 1995.

[114] Hirst and D. St-Onge. Lexical chains as representations of context for the detection and correction of malapropisms. In C. Fellbaum, editor, WordNet: An electronic lexical database, pages 305–332. MITPress. 1998.

[115] Banerjee S. and Pedersen T.. An adapted Lesk algorithm for word sense disambiguation using Word- Net. In Proceedings of the Third International Conference on Intelligent Text Processing and Computational Linguistics, pages 136–145, Mexico City, February 2002.

[116] Patwardhan. Incorporating dictionary and corpus information into a context vector measure of semantic relatedness. Master's thesis, University of Minnesota, Duluth, August 2003.

[117] Steffen, B., Thomas, W., & Kurt, G. An Ontology for Quality-Aware Service Discovery. International Journal of Computer Systems Science & Engineering, Special Edition Editorial: Engineering Design and Composition of Service-Oriented Applications, Computer Systems Science & Engineering, 21(5). 2006.

[118] Fichier OWL contenant la déclaration des builtin personnalisés disponible à http://www.redcad.org/members/kaouthar.fakhfakh/negotiation/slaOntActions.owl. Dernière visite le 6 septembre 2010.

[119] Fichier OWL représentant le modèle de contrat de QdS "SLAOnt" disponible à http://www.redcad.org/members/kaouthar.fakhfakh/negotiation/SLAont.owl. Dernière visite le 6 septembre 2010.

[120]Do H. H., Melnik S., Rahm E. Comparison of Schema Matching Evaluations. In Revised Papers From the Node 2002 Web and Database-Related Workshops on Web, Web-Services, and Database Systems (October 07 - 10, 2002). A. B. Chaudhri, M. Jeckle, E. Rahm, and R. Unland, Eds. Lecture Notes In Computer Science, vol. 2593. Springer-Verlag, London, pages 221-237. 2003.

[121] C. J. Van Rijsbergen. Information retrieval. Butterworths, London, 2 edition, 1979.

[122] Fichier OWL représentant les offres d'un fournisseur d'une bibliothéque numérique de films. Disponible à http://www.redcad.org/members/kaouthar.fakhfakh/negotiation/providerOnto.owl. Dernière visite le 6 septembre 2010.

[123] Fichier OWL représentant l'intention d'un client pour télécharger des films. Disponible sur http://www.redcad.org/members/kaouthar.fakhfakh/negotiation/clientOnto.owl. Dernière visite le 6 septembre 2010.

[124] Fichier OWL représentant l'instance de FlightSLA. Disponible à http://www.redcad.org/members/kaouthar.fakhfakh/negotiation/FlightSLA.owl. dernière visite le 6 septembre 2010.

[125] Alipio, P., Lima, S., Carvalho, P. XML service level specification and validation. 10th IEEE Symposium on Digital Object Identifier: Computers and Communications. ISCC, pages 975 - 980. 2005.

PUBLICATIONS

Revues Scientifiques :

Kaouthar FAKHFAKH, Tarak CHAARI, Said TAZI, Khalil DRIRA and Mohamed JMAIEL. «ODACE SLA: Ontology Driven Approach for automatic Establishment of Service Level Agreements». In the International Journal of Systems and Service-Oriented Engineering, 1(3), 1-20, July-September 2010.

Kaouthar FAKHFAKH, Tarak CHAARI, Said TAZI, Mohamed JMAIEL et Ikbel Guidara. « Modélisation et alignement sémantique des intentions des clients avec les offres des fournisseurs». Dans la revue des nouvelles technologies de l'information RNTI L5 CAL 2010. Mars 2010.

Kaouthar FAKHFAKH, Tarak CHAARI, Said TAZI, Khalil DRIRA and Mohamed JMAIEL. «Semantic Enabled Framework for SLA Monitoring». In The International Journal On Advances in Software, IARIA, Page 36-46, May 2009.

Conférences internationales avec comité de lecture

Kaouthar FAKHFAKH, Tarak CHAARI, Saïd TAZI, Khalil DRIRA and Mohamed JMAIEL. « Implementing and testing a semantic-driven approach towards a better comprehension between service consumers and providers». In the Sixth IEEE International Symposium on Frontiers of Information Systems and Network Applications (FINA) held in conjunction with IEEE AINA 2010. 20-23 April 2010, Perth, Australia». (IEEE Xplore)

Kaouthar FAKHFAKH, Tarak CHAARI, Saïd TAZI, Khalil DRIRA and Mohamed JMAIEL. «Enhancing Client Intentions Analysis for Service Level Agreements Establishment Assistance». In the First IEEE International Conference on Models and Ontology-based Design of Protocols, Architectures and Services MOPAS 2010. Athens/Glyfada, Greece. June 13-19, 2010

Kaouthar FAKHFAKH, Tarak CHAARI, Ikbel GUIDARA and Saïd TAZI. « ODACE SLA : An efficient tool for the automatic SLA Establishment between service clients and providers». A demonstration paper in the 10th Annual International Conference on New Technologies of Distributed Systems - Notere 2010. Tozeur, Tunisia. 31 May - 2 June 2010

Kaouthar FAKHFAKH, Tarak CHAARI, Saïd TAZI, Khalil DRIRA and Mohamed JMAIEL. « WordNet based approach towards a better comprehension between providers and clients». In The International Conference on Software Engineering, CONSEG - 09. December 17–19, 2009 at Chennai, India (IEEE Computer Society)

Kaouthar FAKHFAKH, Tarak CHAARI, Said TAZI, Khalil DRIRA and Mohamed JMAIEL. «A comprehensive ontology-based approach for SLA Obligations monitoring». In The 2008 Second IEEE International Conference on Advanced Engineering Computing and Applications in Sciences. ADVCOMP 2008. September 29 - October 4, Valencia, Spain. (IEEE Xplore) Best paper award

Conférences nationales avec comité de lecture

Kaouthar FAKHFAKH, Tarak CHAARI, Saïd TAZI, Khalil DRIRA et Mohamed JMAIEL. « Service Level Agreements Modeling and Monitoring Using Ontologies ». 2èmes Journées Francophones sur les Ontologies. JFO 2008. 1 – 2 Décembre 2008, Lyon, France. Actes édités avec un ISBN ACM. 6 p.