Formalisation des contrats structurels et de QdS d'une composition de services Web

Raoudha Maraoui
Prince Unity
Hammam Sousse, Tunisia
maraoui.raoudha@gmail.com

Amina Ben Belgacem
Faculté des Sciences Monastir
benbelgacem.amina@gmail.com

Mohamed Graiet
High School of Computer Science and Mathematics
Monastir, Tunisia
mohamed.graiet@imag.fr

Eric Cariou
Université de Pau et des pays
France
Eric.Cariou@univ-pa

ABSTRACT

Etablir et s'assurer de la qualité de service (QdS) d'un composant tel qu'un service Web représente un enjeu crucial puisque ceci permet d'établir une relation de confiance entre le fournisseur d'un service et un client en attente d'une certaine fiabilité. Cependant, contrairement aux spécifications bien établies dans le domaine fonctionnel des services Web (telles que WSDL, SOAP ou UDDI), il n'existe pas de standards officiellement reconnus par la communauté en ce qui concerne la description et l'établissement de propriétés de QdS. Pour autant, plusieurs travaux visent à apporter des réponses à cette lacune par le biais des contrats de service (SLA : Service Level Agreement) dont l'objectif est de permettre la réalisation d'un accord entre consommateur et fournisseur de service portant sur le niveau de QdS que doit fournir le service.

L'objectif du travail présenté est de proposer un style architectural pour permettre la vérification formelle des contrats structurels et des contrats de QdS d'une composition de services Web avec l'ADL ACME. Pour se faire, nous modélisons en premier lieu le méta-modèle d'un service Web décrivant les aspects structurels et ceux de QdS. Puis, nous spécifions les propriétés et les contraintes non fonctionnelles avec CQML et WSLA. Enfin, nous formalisons les deux contrats de qualité de service, CQML et WSLA, avec ACME en se référant à ARMANI qui fournit un langage de prédicats assez puissant pour assurer formellement une composition fiable de Services Web.

Keywords

Services Web, composition, ADL, ACME/ARMANI, méta-modèle, contrat de qualité de service, vérification formelle, CQML, WSLA.

1. INTRODUCTION

L'architecture SOA (Service-Oriented Architecture) permet la réutilisation des applications et des services existants. Ainsi de nouveaux services peuvent être créés à partir d'une infrastructure informatique de systèmes déjà existante, en leur offrant une interopérabilité entre applications et technologies hétérogènes. Afin de mettre en oeuvre une architecture SOA avec succès, il ne suffit pas d'avoir les capacités technologiques. Il convient également d'adopter un certain nombre de principes importants au niveau de la conception, du développement et de la gestion. C'est ainsi que la composition de services Web, qui est la plus importante fonctionnalité assurée par une architecture SOA, peut être vue comme un système complexe [22].

La production de nombreux services disponibles et les nouvelles techniques de programmation permettent aujourd'hui la mise en place d'une composition de services qui évolue en fonction du contexte d'exécution. Le défaut de service ou l'usage malveillant d'un service peut alors occasionner des problèmes dans les applications résultantes des compositions. Il est donc particulièrement utile de définir clairement les propriétés respectées par un service et les conditions à respecter pour utiliser et composer ces services. Lors de l'usage d'un service, il y a donc établissement d'un contrat tacite ou explicite entre les fournisseurs et consommateurs de services. Les services et les contextes d'exécution évoluant, les clauses de ces contrats doivent également évoluer. Les contrats peuvent être eux-mêmes à l'initiative des évolutions des applications lorsqu'ils sont violés en modifiant par exemple la composition de services Web. Ceci est d'autant plus important que les architectures orientées services évoluent actuellement vers des approches mixtes entre composants et services. Il est donc nécessaire d'autoriser une intégration des contrats qui prenne en charge les modifications des compositions des services.

Cependant et afin de vérifier la cohérence ou l'absence de contradiction d'une composition de services, une approche contractuelle basée sur des contrats établis entre les divers services est préconisée. Cette approche contractuelle inter-services est perçue comme un prolongement à la conception par contrats (Design by Contracts) célèbre dans le monde OO et supportée par divers langages comme Eiffel [20] et OCL [21].

Dans cet article, nous traitons non seulement l'aspect structurel mais aussi l'aspect non fonctionnel (qualité de service). En effet, plusieurs recherches abordées dans le domaine de services Web ont essayé de pallier le problème d'élaboration de contrats traitant les contraintes structurelles et comportementales. Cependant, il n'y avait pas assez de préoccupation et concentration sur la qualité de service et très peu de recherches s'intéressaient à cette branche permettant la qualification d'un service Web.

L'objectif de ce travail est de proposer un style architectural qui permet de vérifier formellement en plus des contrats structurelles, les contrats de QdS (Qualité de Service) d'une composition de services Web. C'est grâce à l'ADL ACME/ARMANI [8], qui fournit un langage de prédicats assez puissant pour vérifier toute inadéquation, que nous assurons une composition fiable de services Web.

Le reste de ce document est structuré comme suit : dans la deuxième section, nous abordons les contrats des services Web et leur classification. Ensuite dans une troisième section, nous présentons les travaux connexes sur la vérification formelle des contrats des services Web. Puis, nous présentons dans la quatrième section, notre approche de vérification formelle pour la composition de services Web dotés des propriétés de QdS. Dans la cinquième section, nous spécifions les propriétés de QdS en utilisant CQML et WSLA. Dans la sixième section, nous proposons un style architectural dotés des contrats structurels et de QdS tout en mettant en oeuvre notre travail par un exemple d'application de voyage en ligne. Dans la septième section, nous dressons une comparaison entre les deux formalisations de CQML et WSLA afin de déterminer laquelle paraît la plus pertinente. Enfin, nous donnons un bilan sur le travail réalisé tout en mettant l'accent sur les perspectives envisagées.

2. LES CONTRATS DES SERVICES WEB

Nous étudions les différentes approches contractuelles qui ont été proposées dans les approches par objets, puis par composants, et dernièrement dans les architectures orientées services. Nous nous intéressons donc aux formes de contrats et à certaines techniques de spécification et de vérification dans ces trois paradigmes.
Historiquement, les propriétés d'un programme ont pu être exprimées à l'aide d'assertions qui sont parfois appelées aussi contrats. Une assertion dans un programme est une expression booléenne qui doit être satisfaite pour que le code associé soit exécuté correctement. Les assertions viennent des travaux de Floyd [7] et de Hoare [11]. Elles servent essentiellement dans la spécification des programmes, des objets et plus récemment dans la spécification des composants. Les programmeurs utilisent des assertions dans la description ou le raffinement de contraintes de typage définies dans les interfaces de classe. Le travail le plus connu dans ce domaine est probablement celui d'Eiffel [20]. Le langage Eiffel contient des éléments natifs pour exprimer des assertions dans des pré et postconditions et dans des invariants de classes pour décrire des contrats entre l'utilisateur et le développeur d'une classe. Cette forme de contrat définit ainsi des propriétés vérifiables à l'exécution. Chaque spécification peut être interprétée informellement pour déterminer qui est censée la garantir. En cas de violation, cela permet d'établir des responsabilités.
D'autres travaux utilisent une notion de contrat pour définir, un peu de la même manière, des propriétés à vérifier, avec une notion de responsabilité vis-à-vis de la garantie des propriétés. Ainsi, dans notre étude, nous pourrions aborder la grande majorité des techniques de vérification et de validation en essayant de dégager ces caractéristiques ; mais afin d'utiliser un spectre pertinent, nous ne conserverons que les approches qui se donnent une interprétation contractuelle ou qui sont directement en rapport avec les architectures orientées services.

En effet, établir et s'assurer de la QdS d'un composant tel que le service Web représente un enjeu crucial puisque ceci permet d'établir une relation de confiance entre le fournisseur d'un service et un client en attente d'une certaine fiabilité. Cependant, contrairement aux spécifications bien établies dans le domaine fonctionnel des services Web (telles que WSDL, SOAP ou UDDI), il n'existe pas de standards officiellement reconnus par la communauté en ce qui concerne la description et l'établissement de propriétés de QdS. Pour autant, plusieurs travaux, comme WS-Agreement, IBM, WSLA (Web Service Level Agreement), WSOL (Web Service Offerings Language), SLAng) [18] [14] [27] [16], visent à apporter des réponses à cette lacune par le biais des contrats de service (SLA : Service Level Agreement) dont l'objectif est de permettre la réalisation d'un accord entre consommateur et fournisseur de service portant sur le niveau de QdS que doit fournir le service.
Nous présenterons, par la suite, une classification de contrats des services Web que nous adoptons et qui est basée sur le travail de Tosic et Pagurek [26]. Cette classification distingue trois types de contrats : **fonctionnels**, **de qualités** et **d'infrastructures**. A noter que cette décomposition est inspirée des quatre niveaux de spécifications pour les objets et composants distribués décrits par Beugnard *et al.* [2] :

- **Les contrats fonctionnels** : Ils décrivent ce que fait un service Web. Ceci peut être réalisé aussi bien avec une approche ontologique, par exemple avec OWL-S (Semantic Markup for Web Services) ou WSMO [19], ou en combinant plusieurs types de contrats : contrats syntaxiques, contrats comportementaux, contrats de synchronisation, contrats d'orchestration et contrats de chorégraphie.
- **Les contrats d'infrastructures** : Ils peuvent servir à définir les protocoles de communication utilisés pour encapsuler (par exemple SOAP) et transporter (par exemple HTTP) les messages entre services Web ou encore à définir des politiques de sécurité [3].
- **Les contrats de qualité de service** : Ils permettent de différencier entre des services offrant les mêmes fonctionnalités, en se basant sur leurs propriétés non fonctionnelles telles que la performance, la fiabilité, la disponibilité, etc.

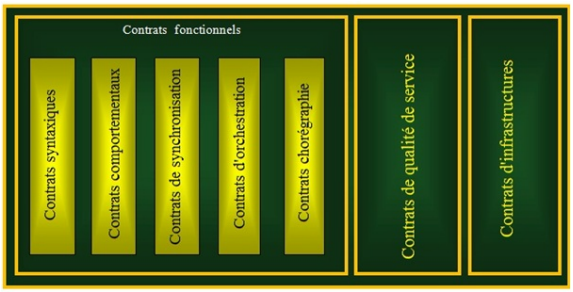

Figure 1: Classification des contrats pour les services Web

La figure 1 montre la classification des contrats pour les services Web. Dans notre travail, nous sommes particulièrement intéressés par les contrats structurels et de QdS des services Web que représentent CQML et WSLA :

- **CQML** C'est un langage lexical de spécification de QdS pour le modèle composant. Il est considéré comme un langage généraliste, compatible avec UML et utilisable à des niveaux d'abstraction différents. Grâce à son recours à l'OCL qui est un langage semi-formel d'expression de contraintes bien adapté aux diagrammes d'UML, en particulier à celui de classes, ce langage offre une bonne précision dans la spécification des QdS. Ainsi, il permet la séparation entre l'aspect qualitatif et l'aspect fonctionnel. CQML repose sur trois concepts clés : la caractéristique de qualité, la qualité et le profil [25].
- **WSLA** Plusieurs langages ont été proposé pour implanter les spécifications de SLAs. Ces langages se focalisent notamment sur l'expression des capacités des fournisseurs de services et des contraintes de qualité de service [6]. Parmi ces langages nous citons WSLA, WSOL, Slang, WS-Agreement. De manière générale, les propriétés de QoS de chaque service et les contraintes qui portent dessus sont décrites dans le cadre d'un SLA (Service Level Agreement) qui décrit le contrat passé en un service

et son utilisateur. Il faut noter qu'aussi bien l'utilisateur que le fournisseur du service peut être le responsable du respect de ces contraintes (bien qu'en général il s'agisse surtout du fournisseur). Dans notre travail, nous nous intéressons à l'étude et la formalisation du contrat WSLA.

3. TRAVAUX CONNEXES SUR LA VÉRIFICATION FORMELLE DES CONTRATS

Vu la complexité croissante des systèmes informatiques et l'évolution fulgurente des technologies Internet, la vérification formelle est devenue un outil indispensable intervenant avant l'implémentation des systèmes. Autrement dit, elle permet de prévoir les erreurs de composition des services Web pouvant intervenir en cas de disfonctionnement ou de mauvaise liaison ou même incompatibilité entre les services Web.

La vérification des propriétés non fonctionnelles est le sujet de quelques travaux de recherche qui ont utilisé différentes techniques d'abstraction et d'outils de vérification formelle. Nous présentons ci-dessous quelques travaux existants pour la vérification des propriétés non fonctionnelles :

- **A. Dekdouk etT.H. El-Basuny** [1] présentent une approche qui utilise la technique de model-checking et la simulation pour vérifier la propriété de vivacité (liveness) d'un processus métier. Ils ont utilisé le langage BPEL pour modéliser la composition des services Web. Ensuite, ils traduisent le processus BPEL vers un code Promela pour vérifier la propriété avec SPIN.
- **R. Kazmiakin, P.K. Pandya, et M. Pistore** [13] proposent une approche pour la modélisation et l'analyse des propriétés temporelles pour la composition des services Web modélisée avec BPEL. En effet, la représentation de l'aspect temps dans BPEL est caractérisée par des lacunes où la durée de l'opération ou l'exigence chronométrage ne peut pas l'être. Il s'agit donc de proposer un formalisme intitulé "Web Service Timed State Transition Systems" (WSTTS) qui est proche des automates temporisés en exprimant la notion de la durée capturée par le comportement temporel d'un service Web composé. Ensuite, ils utilisent le model-checker NuSMV pour vérifier la propriété désirée.
- **J. Koehler, G. Tirenni, et S. Kumaran** [23] définissent une approche dirigée par les modèles pour la vérification de la propriété d'atteignabilité (reachability). Cette approche est fondée sur l'abstraction du processus métier en un automate non-déterministe comme un modèle intermédiaire. Ensuite, ils traduisent l'automate en code SMV, le langage d'entrée du modelchecker NuSMV, afin de vérifier la propriété de reachability exprimée en CTL.

Nous avons choisi l'ADL ACME pour effectuer la vérification formelle pour diverses raisons. En effet, ACME est un langage de description d'architectures établi par la communauté scientifique dans le domaine des architectures logicielles. Il a pour buts principaux de fournir un langage pivot qui prend en compte les caractéristiques communes à l'ensemble des ADL, qui soit compatible avec leurs terminologies et qui propose un langage permettant d'intégrer facilement de nouveaux ADL. En effet, la plupart des langages fournissent des notions similaires comme le composant ou le connecteur. ACME apparaît alors plus comme un langage fédérateur de ce qui existe que comme un langage réellement novateur. Il couvre la spécification des composants, des connecteurs, des configurations et éventuellement des styles architecturaux. Les auteurs d'ACME ont imaginé un langage sur lequel pourraient être converties toutes sortes de description d'architectures écrites suivant d'autres ADL, et depuis lequel une description pourrait être convertie suivant un ADL particulier pour bénéficier de ses outils supports.
Cependant, une extension nommée ACME/ARMANI [9] autorise la spécification de contraintes structurelles. ARMANI est un langage de prédicats puissant. Ce langage couplé à ACME est basé sur des prédicats de logique du premier ordre. Il permet de décrire des propriétés architecturales sous forme d'invariant ou d'heuristique attachées à divers éléments architecturaux. ARMANI est similaire à OCL [21] mais il fournit en plus un ensemble d'opérations de manipulation spécifiques aux architectures logicielles. Ces opérations facilitent la définition de contraintes qui portent sur des concepts comme les interactions entre composants, la conformité de types ou les relations d'héritage. ARMANI est basé sur quatre concepts de base : les fonctions prédéfinies, les opérateurs, les quantificateurs et les fonctions de conception.
L'outil de vérification est directement intégré dans l'environnement ACMEStudio. Ce qui permet à l'outil de fournir des messages d'erreurs, de haut niveau d'abstraction, en relation avec la conception de l'architecture. De cette manière, l'architecte logiciel peut directement comprendre l'erreur et modifier sa conception.

4. APPROCHE PROPOSÉE

Dans le contexte de la vérification formelle pour la composition des services Web, nous utilisons une approche basée sur l'ingénierie dirigée par les modèles IDM, comme décrit par la figure 2. Au niveau M2 se trouvent le méta-modèle d'une composition de services web et sa formalisation sous forme d'un style architectural en ACME/ARMANI. Le niveau M1 décrit les modèles de services que nous visons à vérifier. Ils doivent être par principe conformes à leur méta-modèle respectif[15].

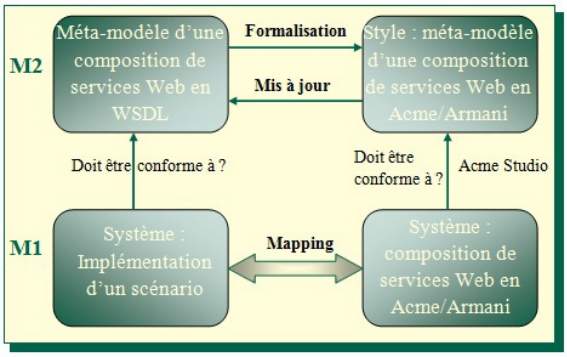

Figure 2: Classification des contrats pour les services Web

Notre travail s'intéresse en particulier à vérifier formellement les contrats de qualité de service. En se basant sur les deux langages dédiés aux contrats de qualité de service (CQML et WSLA), nous allons formaliser ces deux derniers avec l'ACME Studio. Puis, nous implémenterons, au niveau M1, un système de réservation de voyages en ligne qui qoit conforme à son style architectural avec l'extension des propriétés de QdS. Ensuite, grâce à ACME Studio qui fournit le langage de prédicats "ARMANI", nous vérifierons la composition des services Web de notre système en comparant les deux versions (la formalisation en CQML et la formalisation en WSLA). Un système ACME de niveau M1 est en effet dit conforme à son méta-modèle en ACME, s'il satisfait les règles de cohérence décrites dans le niveau M2. La figure 3 montre notre démarche de Vérification formelle des contrats Structurels et de QdS. En effet, on part d'une composition de services Web décrite en WSDL, puis on formalise ses contrats structurels avec ACME pour l'obtension d'une composition de services Web avec ACME à laquelle on ra-

joute des contrats de Qualité de Service. Ensuite, on vérifie une telle composition avec l'outil AcmeStudio.

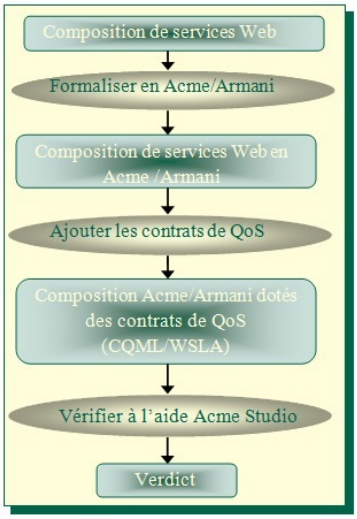

Figure 3: Démarche de Vérification formelle des contrats Structurels et de QdS

5. SPÉCIFICATION DES QDS AVEC CQML ET WSLA

Vu son utilisation fréquente comme exemple d'illustration, nous avons choisi le système composite "réservation de voyage en ligne RVL" comme application cible pour notre étude de cas. Le diagramme de composants de la figure 4 représente la composition des services Web formant le service composite "Réservation de voyage en ligne" et il décrit structurellement la composition du système. Ce diagramme permet de mettre l'accent sur la notion d'interface relative aux services Web. En effet, il permet d'illustrer la manière dont les services Web communiquent à travers ces interfaces. En effet, un client souhaitant partir en voyage a besoin de consulter une agence de voyage en ligne lui permettant de spécifier ses besoins (Composant SBC) pour réserver son vol (Composant RV) et sa chambre d'hôtel (Composant RH). Puis, le client paye ce qu'il doit à travers le Composant SP qui s'en charge d'envoyer les documents (Composant SEDR). Afin de satisfaire ce besoin, plusieurs services interagissent bien qu'ils soient indépendants et probablement hétérogènes. Cette interaction nécessite un mécanisme de composition assez robuste pour assurer un fonctionnement assez efficace.

5.1 Spécification des QdS avec CQML

Nous proposons d'attacher quelques propriétés non fonctionnelles (PNFs), particulièrement de qualité de service, aux services de notre système afin de garantir le bon fonctionnement de ce dernier. Les qualités traitées dans notre système sont la fiabilité, la disponibilité et la performance.

- **Spécification des caractéristiques de qualité**
  La caractéristique de qualité (quality_characteristic) est la construction de base d'une spécification CQML permettant la définition d'un type d'une PNF. Chaque caractéristique possède un nom et un domaine et elle peut être paramétrée (les paramètres admis peuvent être des opérations, des classes ou des interfaces aux sens d'UML). En outre, une caractéristique CQML peut contenir dans sa définition un invariant exprimé à l'aide des prédicats

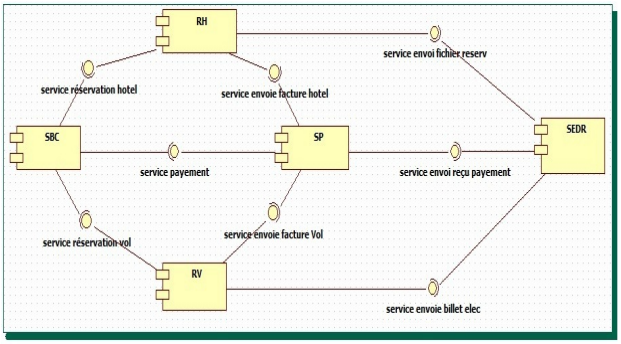

Figure 4: Diagramme de composants du système "RVL"

OCL. Ainsi, elle peut avoir une clause "Values" formulant l'expression de calcul de la valeur de la qualité. Nous spécifions, dans le listing 1, que la propriété non-fonctionnelle "Disponibilité" est définie en pourcentage. Donc, ses valeurs sont de type réel dans l'intervalle [0,100]. Ainsi, plus ces valeurs sont élevées, plus la disponibilité d'un service est améliorée.

```
1 Quality_characteristic Disponibilite
2 {domain: increasing numeric real [0..100] %;
3 }
```

Listing 1: Modélisation de la caractéristique Disponibilité en CQML

- **Spécification des qualités**
  Le concept qualité (quality) permet de spécifier une catégorie de qualité d'un service. Il permet la définition d'une qualité par la restriction d'un ensemble de caractéristiques en utilisant des contraintes sur ces dernières. Chaque qualité en CQML est identifiée par un nom et un ensemble de sous-qualités. En outre, elle peut être paramétrée.

  Le listing 2 illustre bien que nous restreignons la PNF "Disponibilité" afin d'établir deux qualités. La première, appelée "DispoBonne", exige que la PNF relative soit supérieure ou égale à 80%, et la deuxième associe à la "Disponibilité" une qualité supérieure ou égale à 90%.

```
1 Quality DispoBonne
2 {  Fiabilite>= 80;
3 }
4 Quality DispoTresBonne
5 {
6   Fiabilite >= 90;
7 }
```

Listing 2: Modélisation des qualités liées à la disponibilité en CQML

- **Attachement de qualités aux services Web**
  Après la définition des deux concepts permettant la spécification de QdS, il faut attacher à chaque service ses propres qualités requises et/ou offertes. Ceci constitue le troisième concept de CQML, le profil. Afin de distinguer les qualités requises de celles offertes d'un service Web, CQML associe le mot clé "provides" aux qualités offertes et le mot clé "uses" aux qualités requises.

  Parmi les qualités spéficiées dans le profil du service Web de

besoin de client, nous montrons dans le listing 3, que "SBC" doit offrir une disponibilité très bonne et exige en contrepartie une bonne disponiblité.

```
Profile profilSBC for SBC
{   provides DispoTresBonne;
    uses DispoBonne;}
```

Listing 3: Modélisation des profils associés aux services Web en CQML

5.2 Spécification des QdS avec WSLA

Le WSLA (Web Service Level Agreement) forme un contrat pour la spécification, le contrôle et la vérification de violation des "SLA". Il garantit de prendre une action en cas où l'une des parties signataires ne respecte pas les articles et obligations exigés dans ce contrat. Il intervient donc tout au long du cycle de vie du contrat entre le fournisseur du service et le client.
WSLA est un contrat à base de XML et comporte ainsi trois concepts majeurs permettant la modélisation des "SLAs" (ou encore les QdS). Ces derniers sont les "parties", le "serviceDefinition" et les "obligations". Le listing 4 présente la structure générale de WSLA [5].

```
<xsd:complexType name="WSLAType">
  <xsd:sequence>
   <xsd:element ref="wsla:Parties"/>
   <xsd:element name="ServiceDefinition" type="
       wsla:ServiceDefinitionType" maxOccurs="
       unbounded"/>
   <xsd:element name="Obligations" type="wsla:
       ObligationsType"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>
<xsd:element name="SLA" type="wsla:WSLAType">
</xsd:element>
```

Listing 4: Structure générale de WSLA

Dans ce qui se suit, nous modélisons les SLAs à l'aide de notre application de voyage.

- **Spécification des "parties"** Cette partie du contrat WSLA, appelée "Parties", identifie les différents acteurs intervenant dans la spécification et la surveillance du niveau de service. Elle permet de définir deux types de "parties" : un acteur peut être soit "signatoryParty", soit "supportingParty". Chaque "Party" est identifiée par ses coordonnées et son action.
  Le listing 5 représente la structure générale de la partie "Parties" du contrat WSLA :

```
<xsd:complexType name="PartyType" abstract="true
    ">
  <xsd:sequence>
   <xsd:element name="Contact" type="wsla:
       ContactInformationType" minOccurs="0"/>
   <xsd:element name="Action"  type="wsla:
       ActionDescriptionType" minOccurs="0"
       maxOccurs="unbounded"/>
  </xsd:sequence>
  <xsd:attribute name="name" type="xsd:string"/>
</xsd:complexType>
<xsd:complexType name="SignatoryPartyType">
  <xsd:complexContent>
   <xsd:extension base="wsla:PartyType"> </xsd:
       extension>
  </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="SupportingPartyType">
    <xsd:complexContent>
      <xsd:extension base="wsla:PartyType">
       <xsd:sequence>
        <xsd:element name="Sponsor" type="xsd:
            string" maxOccurs="2"/>
       </xsd:sequence>
     <xsd:attribute name="role" type="wsla:
         RoleType"/>
      </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
<xsd:complexType name="PartiesType">
  <xsd:sequence>
    <xsd:element name="ServiceProvider" type="
        wsla:SignatoryPartyType"/>
    <xsd:element name="ServiceConsumer" type="
        wsla:SignatoryPartyType"/>
   <xsd:element name="SupportingParty" type="
       wsla:SupportingPartyType" minOccurs="0"
       maxOccurs="unbounded"/>
  </xsd:sequence>
</xsd:complexType>
<xsd:element name="Parties" type="wsla:
    PartiesType"/>
```

Listing 5: Structure générale de "Parties" du WSLA

Afin de bien comprendre la structure générale des "Parties" du WSLA, nous proposons une illustration relative à notre système de "RVL". En effet, le listing 6 présente le "provider" du service, appelé "RH", le "consumer", nommé "SBC" et la partie tierce appelée "suppSBCH" qui est sponsorisée par les deux WS précédemment définis. Ainsi, nous spécifions pour chaque partie impliquée son action.

```
<Parties>
  <ServiceProvider name="SH">
     <Action xsi:type="
         WSDLSOAPActionDescriptionType" name="
         Notification" partyName="suppSBCH">
       <WSDLFile>Notification.wsdl</WSDLFile>
       <SOAPBindingName>SOAPNotificationBinding
           </SOAPBindingName>
       <SOAPOperationName>Notify</
           SOAPOperationName>
     </Action>
  </ServiceProvider>
  <ServiceCustomer name="SBC">
    <Contact>
      <Street> Rue Paul Santy 69008</street>
      <City> Lyon, France</City>
      <Mail>WS.SBC@gmail.com</Mail>
    </Contact>
   </ServiceCustomer>
 <SupportingParty name="suppSBCH" role="
     ManagementService">
    <Contact>
      <Street> 19 Skyline Drive</street>
      <City> Hawthorne, NY 10532, USA/City>
      <Mail>supp.SBCH@hotmail.fr</Mail>
   </Contact>
 <Action xsi:type="WSDLSOAPActionDescriptionType
     " name="Notification" partyName="SH">
     <WSDLFile>Notification.wsdl</WSDLFile>
     <SOAPBindingName>SOAPNotificationBinding</
         SOAPBindingName>
     <SOAPOperationName>Notify</
```

```
        SOAPOperationName>
  </Action>
        <Sponsor>SBC</Sponsor>
        <Sponsor>SH</Sponsor>
    </ServiceCustomer>
```

Listing 6: Exemple de "parties" du WSLA

- **Spécification de "serviceDefinition"** C'est au niveau de "serviceDefinition" que la spécification des "SLAParameters" intervient. Bien que "SLAParameter" constitue le concept fondamental de la définition du service, cela n'empêche pas que cette partie définisse les opérations qui attachent les SLAs au service désigné. La définition de service permet de spécifier un groupe d'opérations en attribuant, pour un ensemble d'opérations, un "SLAParameter" commun.

  Le listing 7 définit un certain nombre de planifications de notre système "RVL", des "Schedules" (par exemple "DaySchedule"). Ensuite, nous définissons une "opération" intitulée "ReservHotel" en spécifiant ses différents "SLAParameters" (par exemple "Disponibilité") et leurs "metrics" (respectivement "CalculDispo"). La description de l'opération est accompagnée par le nom du fichier référence WSDL, de "SOAPBinding" et l'opération SOAP sur laquelle les "SLAParameters" sont décrits.

```
<ServiceDefinition name="WSLA_SBC&SH">
    <Schedule name="DaySchedule">
       <Period>
          <Start>2014-02-14</Start>
          <End>2014-02-14</End>
       </Period>
       <Interval>
          <Hours>24</Hours>
       </Interval>
    </Schedule>
    <Operation>
        <SLAParameter name="Disponibilit " type
            ="float" unit="Percentage">
            <Metric>CalculDipo</Metric>
            <Communication>
              <Source>suppSBCH</Source>
              <Pull>SBC</Pull>
              <Push>SBC</Push>
            </Communication>
        </SLAParameter>
 <Metric name="CalculDispo" type="float" unit="
     percentage">
   <Source>suppSBCH</Source>
        <Function xsi:type="times" resultType="
            float">
            <Operand>
               <Function xsi:type="divide"
                   resultType="float">
            <Operand>
 <Metric>SumRequ tesEffectu es </LongScalar>
          </Operand>
         <Operand>
           <Metric>totRequ tes </LongScalar>
             </Operand>
             </Function>
         </Operand>
             <Operand>
               <LongScalar>100</LongScalar>
          </Operand>
         </Function>
</Metric>
        <WSDLFile>WSDLHotel.wsdl</WSDLFile>
        <SOAPBindingName>ReservHotel</
            SOAPBindingName>
        <SOAPOperationName>confirmReservHotel</
            SOAPOperationName>
    </Operation>
</ServiceDefinition>
```

Listing 7: Exemple de "serviceDefinition" du WSLA

- **Spécification des "obligations"** La clause "obligations" du contrat WSLA permet de mettre des contraintes sur les qualités de service afin de maintenir le même niveau de qualité ou de satisfaire les pré-conditions appliquées sur les "SLAParameters", déjà définies dans la partie de "serviceDefinition". Les obligations peuvent être regroupées pour différentes raisons, notamment pour faciliter la création des modèles de groupe d'obligations, pour associer des pénalités financières avec des groupes de garanties, etc. Nous proposons une illustration des obligations relative à notre système de "RVL". En effet, le listing 8 définit premièrement un "ServiceLevelObjective" ("SDispo") obligeant "RH" de maintenir une "Disponibilité" supérieure ou égal à 80%. Ainsi, cet extrait du contrat WSLA permet de définir une action de garantie obligeant "SBC" à notifier les autres parties qu'une violation sur "SDispo" a été détectée.

```
<Obligations>
    <ServiceLevelObjective name="SDispo">
        <Obliged>SH</Obliged>
        <Validity>
            <Start>2014-02-11</Start>
            <End>2014-02-11</End>
        </Validity>
        <Schedule>DaySchedule</Schedule>
        <Expression>
            <Predicate xsi:type="GreaterEqual">
                <SLAParameter>Disponibilit </
                    SLAParameter>
                <Value>0.8</Value> <!-- 80\%-->
            </Predicate>
        </Expression>
    </ServiceLvelObjective>
    <ActionGuarantee name="AGDispo">
        <Obliged>SBC</Obliged>
        <Expression>
            <Predicate xsi:type="Violation">
                <ServiceLevelObjective>SDispo</
                    ServiceLevelObjective>
            </Predicate>
        </Expression>
        <EvaluationEvent>NewValue</
            EvaluationEvent>
        <QualifiedAction>
            <Party>SBC,SH</Party>
            <Action actionName="notification"
                xsi:type="Notification">
                <NotificationType>Violation</
                    NotificationType>
                <CausingGuarantee>SDispo</
                    CausingGuarantee>
                <SLAParameter>Disponibilit </
                    SLAParameter>
            </Action>
        </QualifiedAction>
    </ServiceLvelObjective>
</Obligations>
```

Listing 8: Exemple d' "obligations" du WSLA

6. FORMALISATION DE LA COMPOSITION DES SERVICES WEB

6.1 Proposition d'un style architectural pour les services Web En ACME/ARMANI

En étudiant les systèmes déployés, on constate qu'un certain nombre d'architectures ne se bornent pas à utiliser un seul style. C'est le cas pour notre style WSC (Web Service Composition) qui fonctionne comme le style client/serveur avec des rôles symétriques et tirant quelques spécificités du style pipe/filter. Le style WSC proposé comporte deux composants : les services "clients" et les services "serveur". Tous jouent le rôle d'un service ayant certaines fonctionnalités [24]. Les services "clients" ainsi que les services "serveurs" peuvent communiquer uniquement avec les connecteurs. Ils utilisent SOAP comme protocole de communication par le fait qu'il s'adapte aux échanges de données structurées indépendamment des langages de programmation ou des systèmes d'exploitation. Chaque composant peut jouer le rôle d'un client ou serveur. Le client doit avoir un port de sortie pour envoyer un message et recevoir une réponse sur ce même port. Ce style n'est pas spécifique à un domaine, il est plutôt générique et ceci afin d'augmenter le niveau de réutilisation et de s'adapter à n'importe quel domaine. En fait cet avantage revient à l'ADL ACME qui permet à ces utilisateurs de formaliser leurs propres styles.

```
Family WSC = {
// ******************Formalisation WSDL
    ****************************
// Classe documentation
property type doc = set {string};
// Classe element
property type elmt = record [name:string; typbase
    :string;];
// Classe part
property type part =record [name:string; typ:
    string; elt: elmt;];
// Classe message
property type message =record [name: string;
    parts: sequence <part>;];
// Classe operation
property type op= record [name: string; typmsg :
    sequence<record [typm: enum {input, output,
    fault};msg: message;]>;];
// Classe port type
property type interface =record [name:string; ops
    : sequence<op>;];
// Types des protocoles
property type TypeTransportProtocols = enum {
    HTTP1_0, HTTP1_1};
property type TypeSoapVersions = enum {SOAP1_1,
    SOAP1_2};
// Les protocoles impliques dans  binding
property type protocol= Record [Transport :
    TypeTransportProtocols; Encoding : enum {
    SOAP1_1, SOAP1_2};];
// Classe binding
property type liaison = record [name:string; typ:
     interface; protocols: set{protocol};];
```

Listing 9: Formalisation de WSDL en ACME/ARMANI

Le listing 9 montre un extrait de notre style WSC. Nous nous sommes inspirés pour ce faire du méta-modèle de WSDL du référentiel OASIS [4] pour formaliser les différents concepts de WSDL en ACME/ARMANI :

- **Classe "Service" :** est formalisée en ACME par un *"component"*.
- **Classe "Port" :** correspond à un *"port Type"* en ACME. Nous définissons deux types de ports : un port type Pclient et un port type Pserveur.
- **Les autre classes :** pour le reste des méta-classes du méta-modèle de WSDL, nous les formalisons en des *"property type"* en ACME.

- **Classe "Binding" :** Dans le méta-modèle de WSDL, un "binding" n'appartient qu'à un seul "port". Donc nous le formalisons en ACME avec un *"property type"* nommé "liaison" composée par un nom, une interface, et un ensemble de protocoles utilisés.
- **Classse "Port Type" :** Permet de décrire l'interface d'un service Web. Ainsi, elle est formalisé par une propriété type appelée "interface". Cette dernière se réfère aux opérations introduites par le service. Donc la classe "Operation" est aussi formalisée en une propriété appelée "op".
- **Classe "Documentation" :** elle est représentée par une propriété appelée "doc".
- **Classe "Message" :** est formalisée en ACME par la propriété "message" qui consiste en un enregistrement composé d'un champ nom et un autre qui forme un ensemble de "part"(s).
- **Classe "Part" :** traduite en ACME en une propriété nommée "part". À son tour, cette propriété fait appel à la propriété "elmt" qui formalise la classe "Element".

6.2 Formalisation des contrats structurels en ACME/ARMANI

Le langage ARMANI, couplé à ACME, permet de décrire des propriétés architecturales sous forme d'invariants ou d'heuristiques attachés à divers éléments architecturaux.
ACME/ARMANI fournit des règles de type *"invariant"* qui permettent de signaler des erreurs en cas de violation des contraintes qu'ils définissent. Ainsi, nous implémentons dans notre formalisation les contrats structurels permettant de respecter la notion des services Web, leur architecture et leur composition. En effet, nous avons introduit, différents *"invariant"*(s) traitant des règles de nos formalisations.
Par exemple, dans notre style architectural [10] :

- Un service Web doit avoir au moins un "port".
- Chaque rôle d'un connecteur défini doit être relié à un "port" d'un service Web.
- Un port attaché à un service Web Serveur doit être de type service

Le listing 10 suivant représente l'ensemble des contrats cités précédemment.

```
component type CService= {
  rule rule1= Invariant size(self.ports)>=1;}
connector type cnxServices=
{ role serveur=
 {rule rule2= Invariant size(self.AttachedPorts)
     ==1;
 rule rule3= Invariant forall p1:port in self.
     AttachedPorts|declaresType(p1,PServeur);}
 role client=
 {rule rule3= Invariant size(self.AttachedPorts)
     ==1;
 rule rule4= Invariant forall p2:port in self.
     AttachedPorts|declaresType(p2,PClient);}
  rule binaire= Invariant size(self.roles)==2;}
  rule rule5 = Invariant forall conn : connector
      in self.connectors | forall r : role in
      conn.roles| exists comp : component in self
      .components | exists p : port in comp.ports
       | attached(p,r);
```

Listing 10: Formalisation des contrat structurels en ACME/ARMANI

6.3 Formalisation des contrats de QdS CQML en ACME/ARMANI

6.3.1 Formalisation du concept "caractéristique de qualité"

La caractéristique de qualité est la construction de base de toute spécification non fonctionnelle. On peut la formaliser par une propriété ACME/ARMANI.
Cette propriété est de type enregistrement composé de cinq principaux champs : un "nom" qui représente le nom significatif de la caractéristique de qualité, des "paramètres", une valeur exprimée par une formule de calcul, un "invariant" définissant une contrainte sur la caractéristique et un champ "domaine". Ce dernier contient en lui même un champ "Dom" qui peut être soit un type numérique, soit un type non numérique (ordinaire, par exemple, "bien","trèsBien", etc.). Par conséquent, la traduction, en ACME, du concept "caractéristique de qualité" de CQML se divise en deux : "caractéristiqueNumérique" et "caractéristiqueOrdinaire" (Listing 11).

```
// Formalisation du concept
    caracteristiqueNumerique en ACME/ARMANI
property type CaracteristiqueNumerique = record
 [Nom : string;
 Parametres: sequence <record [Nom_Par : string;
     Type_Par: string;]>;
 Domaine : record
  [ direction: enum {increasing, decreasing};
  dom: enum{numeric_real, numeric_integer};
  unite: string; ];
 Valeur : string ;
 Invar: string ; ];
// Formalisation du concept
    caracteristiqueOrdinaire en ACME/ARMANI
property type CaracteristiqueOrdinaire = record
 [Nom : string;
 Parametres: sequence <record [Nom_Par : string;
     Type_Par: string;]>;
 Domaine : record
  [ direction: enum {increasing, decreasing,null
      };
  dom: set{ string };
  unite: string ; ];
 Valeur: string ;
 Invar: string ; ];
```

Listing 11: Formalisation du concept "Quality_characteristic" en ACME/ARMANI

Le listing 12 présente la formalisation de la caractéristique "Disponibilité" en ACME/ARMANI. Cette caractéristique est de type quantitatif (le champ "Dom" est de type numérique). Ainsi, elle permet de calculer le nombre des requêtes traitées par rapport au nombre total des requêtes reçues. La variance de l'ensemble des valeurs de la "disponibilité" est croissante et exprimée en pourcentage. Le champ "paramètres" peut être renforcé par des contraintes ARMANI. En outre, le champ "Invar", qui décrit un invariant, peut présenter une approche de traduction OCL vers ARMANI.

```
property Dispo: CaracteristiqueNumerique=
  [Nom = "Disponibilite";
     Parametres=<[Nom_Par=""; Type_Par="";]>;
     Domaine = [direction = increasing;
         dom =  numeric_real;
        unite ="\%"; ];
    Valeur   = "Disponibilite = Nombre de
        requ tes  r ussites/Nombre total de
        requ tes" ;
    Invar    ="valeur positive ( > 0)"  ;];
```

Listing 12: Formalisation de la caractéristique "Disponibilite" en ACME/ARMANI

De la même manière, nous avons formalisé le concept de la "Qualité" et de "Profil". Cependant, d'autres contrats plus complexe peuvent être pris en compte dans cette formalisation comme par exemple l'invariant suivant qui présente une contrainte *"CQuality"* mise sur la formalisation de CQML. En effet, elle consiste à vérifier que, pour un service Web "S1" ayant un profil "prof1" tel qu'il exige un certain nombre de qualités, tout service "S2" connecté directement à "S1" doit avoir un profil "prof2" fournissant des qualités satisfaisant le niveau des qualités requises par "S1".

```
rule CQuality = invariant forall S1: CService in
    self.components| // pour chaque "S1",
 exists prof1: property in S1.properties|
     declaresType (prof1,Profile)-> // il existe
     un profil
 forall S2:CService in self.components|connected(
     S2,S1)-> // chaque "S2" connect     "S1"
 exists prof2:property in S2.properties|
     declaresType(prof2,Profile)-> // ayant un
     profil
    // pour chaque QdSEx de prof1, il doit
        exister QdSOf dans prof2
 forall QdSEx:property in prof1.properties |
     exists QdSOf: property in prof2.properties|
  declaresType(QdSEx,Qualite)and declaresType(
      QdSOf,Qualite)->
    //pour chaque pnf1 dans les qualit s QdSEx,
        il doit exister pnf2 dans QdSOf
 forall pnf1: property in QdSEx.properties|exists
      pnf2: property in QdSOf.properties|
  declaresType(pnf1,PNFNum)and declaresType(pnf2,
      PNFNum)->
    // dans ce qui se suit, on v rifique que
        pour chaque CNpnf1, il existe CNpnf2
    // telque  pnf1 doit satisfaire les valeurs
        exig es par pnf2
 forall CNpnf1: property in pnf1.properties|
     exists CNpnf2: property in pnf2.properties|
  declaresType(CNpnf1,CaracteristiqueNumerique)
      and declaresType(CNpnf2,
      CaracteristiqueNumerique)->
  CNpnf1.Nom== CNpnf2.Nom and pnf1.Operateur ==
      pnf2.Operateur and
   ((pnf1.Operateur == SuperieurOuEgal-> pnf2.
       Valeur >= pnf1.Valeur)or
   (pnf1.Operateur == Superieur-> pnf2.Valeur >
       pnf1.Valeur)or
   (pnf1.Operateur == InferieurOuEgal-> pnf2.
       Valeur <= pnf1.Valeur)or
   (pnf1.Operateur == Inferieur-> pnf2.Valeur <
       pnf1.Valeur));
```

Listing 13: Formalisation de régle "CQuality" en ACME/ARMANI

6.4 Formalisation des contrats de QdS WSLA en ACME/ARMANI

Le WSLA présente un contrat d'accord entre deux services Web qui seront pénalisés en cas de violation de ces derniers. Dans ce qui

suit, nous procédons à la formalisation des concepts de WSLA en ACME. L'approche par contrat formalisé à cette fin se manifeste au niveau du connecteur.

Afin de clarifier notre travail de formalisation de WSLA en ACME, nous divisons cette section en trois sous-sections. Chacune d'elles décrit la formalisation d'un concept fondamental du WSLA.

6.4.1 *Formalisation du concept "Parties"*

Ce concept présente les acteurs impliqués dans le contrat WSLA. Ainsi, afin de bien garder l'aspect structurel de ce concept, nous le formalisons par une propriété ACME, appelée "parties". Cette dernière contient les parties intervenant dans le contrat WSLA tel que dans le listing 13. Elle comporte ainsi un champ "provider" représentant le fournisseur du service, un champ "consumer" faisant référence au client du service et un dernier champ "supp" définissant la partie tierce du contrat.
Le "provider" et le "consumer" sont de type "signatoryP" et le champ "supp" est une séquence de type "supportingP". Ces deux types font appel à la propriété "party" qui définit pour chaque partie son nom, son type, ses coordonnées et son action.

```
// Classe Party
 property type party = record [name:string;
    typarty:enum{signatoryParty, supportingParty
        };
    contact: contactType;
    action: sequence <actionType>;];

 // SignatoryParty
 property type signatoryP = set {party};

// SupportingParty
property type supportingP = record [rol: enum{
    MeasurementService,ManagementService,
    conditionEvaluationService};

  p: set{party};
  sponsors: sequence <string>;];

// Classe Parties
property type parties = record [provider:
    signatoryP;
                                consumer:
                                    signatoryP;
                                supp: sequence <
                                    supportingP
                                    >;];
```

Listing 14: Formalisation du concept "parties" en ACME/ARMANI

Le Listing 14 montre l'instanciation du concept "parties" relatif au contrat WSLA établi entre les deux services web "SBC" et "RH". En fait, l'instanciation de la propriété "parties" se trouve directement sous le connecteur.

```
// Classe parties
property parties: parties =
[// Specification du fournisseur de service et
    ses coordonnees
  provider= {[name ="SH";
  typarty=signatoryParty;
  contact=[street ="PO BOX 218";
  city="Yorktown NY 10598, USA";
  mail="WSHotel@gmail.com";];
  action=<[name="Notification";
  partyName="suppSBCH";
  WSDLFile="Notification.wsdl";
  SOAPBindingName="SOAPNotificationBinding";
  SOAPOperationName="Notify";]
]};
// Specification du client de service et ses
    coordonnees
 consumer= {[name ="SBC";
 typarty=signatoryParty;
 contact=[street ="rue Paul Santy 69008";
 city="Lyon, France";
 mail="WS.SBC@gmail.com";];
 action=<>;]};
// Specification de la partie tierce de service
    et ses coordonnees
 supp=<[rol=ManagementService;
 p={[name ="suppSBCH";
 typarty=supportingParty;
 contact=[street ="19 Skyline Drive";
 city="Hawthorne, NY 10532, USA";
 mail="supp.SBCH@hotmail.fr";]
 action=<[name="Notification";
 partyName="SH";
WSDLFile="Notification.wsdl";
SOAPBindingName="SOAPNotificationBinding";
SOAPOperationName="Notify";]
>;
]};sponsor=<"SBC","SH">;]>;
 ]; // Fin specification "parties" du WSLA
```

Listing 15: Exemple d'instanciation de "parties" en ACME/ARMANI

De la même manière, nous formalisons le concept de "ServiceDefinition"

6.4.2 *Formalisation du concept "ServiceDefinition"*

Cette partie du contrat est caractérisée principalement par les "SLAParameters" qui définissent les qualités de service et les "Metric" qui permettent de calculer les "SLAParameters". Dans cette partie, nous formalisons le concept "ServiceDefinition", tout en gardant à l'esprit sa structure standard en XML. Ainsi, dans l'éditeur "Family" de ACME Studio (cet éditeur permet de formaliser les méta-modèles que nous implémentons), nous définissons un type propriété ("property type") nommé "serviceDefinition". Ses trois champs font appel directement ou indirectement au type "serviceObj" qui contient principalement la définition des "SLAParameters" et les "Metric"(s) comme le montre le Listing 16.

```
    // serviceObj
    property type serviceObj =
     record[name:string;
           s:sequence<schedule>;
           t:sequence<trigger>;
           c:sequence<constant>;
           mmd:sequence<MetricMacroDefinition>;
           mme:sequence<MetricMacroExpansion>;
           SLA_P:sequence<SLAParameter>;
           m:sequence<Metric>;];
  // operationType
  property type operationType =
      record[sObj:serviceObj;
          WSDLFile:string;
          SOAPBindingName:string;
         SOAPOperationName:string;];
```

```
  // operationGroupType
  property type operationGroupType =
  record[sObj: serviceObj;
         operations:sequence <operationType>;];
// classe ServiceDefinition
property type serviceDefinition =
    record [name:string;
         serviceObject:serviceObj;
         operation:sequence<operationType>;
         operationGroup:sequence<
             operationGroupType>;];
```

Listing 16: Formalisation du concept "serviceDefinition" en ACME/ARMANI

6.4.3 Formalisation du concept "Obligations"

La dernière partie du contrat WSLA regroupe les contraintes appliquées sur la qualité de service décrites dans la partie "serviceDefinition". En effet, chaque partie ("provider" ou "consumer") ne respectant pas les clauses de "obligations" sera pénalisée.
Toujours faisant recours à la description type du langage WSLA exprimé en XML, nous formalisons aussi les contraintes par une propriété nommée "obligations" qui forme un enregistrement contenant deux champs : "obligationObject" et "obligationGroup". Le premier champ est de type "obligationObjectType" alors que le deuxième est une séquence de ce dernier.
La propriété "obligationObjectType" contient les "serviceLevelObjective" qui sont une séquence de contraintes permettant de maintenir un certain niveau de qualité de service, et les "actionGuarantee" qui permettent de définir l'action à faire en cas de détection d'une violation de contraintes de niveau de qualité (Listing 17).

```
    // AGType (ActionGuaranteeType)
    property type AGType =
  record[nameAG:string;
  obliged: string;
  expression: expressionType;
  schedul:string;
  evaluationEvent:enum{newValue, noValue};
  qualifiedAction:sequence<qualifiedActionType>;
  executionModality:enum{Always,
          OnEnteringCondition,
          OnEnteringAndOnLeavingCondition,
          OnEveryevaluation}];

// SLOType
  property type SLOType =
  record [nameSLO:String;
  obliged: string;
validity:periodType;
 expression: expressionType;
  schedul:string;
  evaluationEvent:enum{newValue,noValue};];
// obligationObjectType
  property type obliagtionObjectType =
      record [servicelevelObjective: sequence<
          SLOType>;
     actionGuarantee: sequence <AGType>;];

// concept "obligations"
property type obligations =
   record [obligationObject:obliagtionObjectType;
  obligationGroup: sequence <obliagtionObjectType
      > ;];
```

Listing 17: Formalisation du concept "obligations" en ACME/ARMANI

Nous définissons également d'autres contrats d'implémentation de WSLA sous le connecteur. En effet, "r1, "r2" et "r3" permettent de vérifier que, dans une composition de services Web, chaque connecteur doit contenir les trois propriétés formalisant les concepts de WSLA (*"parties", "serviceDefinition", "obligations"*comme le montre le Listing 18).

```
// rule exigeant que chaque connecteur contient
    une propriete de type "parties"
rule r1=invariant forall c : cnxServices in self.
    connectors|
        exists p : property in c.properties |
        declaresType(p,parties);

// rule exigeant que chaque connecteur contient
    une propriete de type "serviceDefinition"
rule r2=invariant forall c : cnxServices in self.
    connectors|
        exists p : property in c.properties |
        declaresType(p,serviceDefinition);

// rule exigeant que chaque connecteur contient
    une propriete de type "obligations"
rule r3=invariant forall c : cnxServices in self.
    connectors|
        exists p : property in c.properties |
        declaresType(p,obligations);
```

Listing 18: Formalisation des régles WSLA en ACME/ARMANI

7. COMPARAISON ENTRE LES DEUX FORMALISATIONS DES CONTRATS DE QDS

L'apparition des langages paramétrant les QdS ont permis de garantir des compositions de services Web fiable et de maintenir un certain niveau de qualité de service en attachant des contraintes sur cette dernière. Dans notre travail, nous avons pris deux contrats de QdS. Le premier, appelé CQML, présente un contrat de QdS orienté composants que nous avons implémenté avec les services Web en ACME. WSLA, un contrat de "SLA" orienté service Web, constitue le deuxième contrat que nous avons traité et formalisé en ACME.

Le premier facteur de différenciation entre les deux formalisations de deux contrats précédemment présentés, réside dès le départ dans l'orientation du contrat. En effet, CQML est dédié à traiter les QdS de l'approche de composants et il ne forme pas un contrat réel mais plutôt il décrit la façon dont nous définissons et vérifions les QdS à l'aide des concepts qu'il fournit. WSLA forme lui un contrat établi entre le fournisseur du service et le client et se focalise sur les QdS des services Web.
La formalisation de CQML est faite sur les différents services Web. En effet, chaque service Web est accompagné par un "profil" contenant les QdS requises et offertes par ce service Web. Par conséquent, c'est une description interne et limitée au sein du service Web ce qui provoque fréquemment la redondance de spécifications des QdS d'un service Web à un autre. De plus la vérification émise concernant la maintenance d'un niveau de qualité de service offert par rapport au niveau de service requis est rectifiée par un simple invariant en ACME, permettant de vérifier que les QdS offertes par un service Web satisfont les besoins requis en terme de QdS de la part du service Web client.

Comparée à CQML, la formalisation de WSLA est considérée robuste et rigoureuse en permettant de bien présenter la notion de contrat de QdS entre deux services Web. Premièrement, WSLA intervient au niveau du connecteur en ACME, ce qui corrige le défaut de la redondance des spécifications de QdS en CQML. En effet, la description des QdS et leur surveillance implémentées sur le connecteur sont exposées au service Web fournissant le service parallèlement au service Web demandant ce dernier. De plus, le contrat de QdS WSLA intervient tout au long du cycle de vie des "SLA" négociés entre deux services Web. Autrement dit, sa formalisation en ACME, bien qu'elle paraisse statique, permet de prévoir le fonctionnement et l'exécution fiable de ce contrat, c'est-à-dire si la formalisation de WSLA en ACME est pertinente, fiable et compatible avec sa structure en XML. Cela permettra de garantir que cette formalisation est valable tout au long de cycle de vie du contrat dès l'étape de négociation jusqu'à l'étape de sa terminaison. Ainsi, la formalisation de WSLA sera très utile si nous utilisons les versions favorisant l'aspect dynamique d'ACME. En effet, les contraintes spécifiées avec le langage ARMANI permettent uniquement la description des architectures statiques. Donc, ACME a été étendu par une nouvelle version nommée Dynamic ACME pour supporter la description des architectures dynamiques [12].

La formalisation du CQML paraît statique et dépendante du système mis en oeuvre : les spécifications décrites dans le système de CQML sont fortement relatives au cas concrètement implémenté. WSLA résout ce problème en permettant dans sa structure de créer des modèles (templates) de mesure et de surveillance des QdS ce qui fait que ces modèles peuvent être réutilisées sans besoin de définir un autre contrat WSLA. Par conséquent, ces contrats de service peuvent s'adapter à des cas multiples et être réutilisés plusieurs fois par un simple mécanisme de référence.

En outre, la formalisation du WSLA offre non seulement la description et la vérification des QdS mais décrit aussi les actions à invoquer en cas de violation d'un certain niveau de qualité de service spécifié. De plus, la formalisation de WSLA est cohérente et en forte relation avec la formalisation de WSDL. En effet, WSLA est rigoureux car il permet d'attacher les QdS non pas directement au Service Web mais aux différents opérations et services de ce service Web. Contrairement, à la formalistion de CQML qui, généralement, liste les QdS exigées et offertes par un service Web.

Nous concluons que la formalisation de WSLA est plus favorable, par rapport à celle de CQML vu qu'elle respecte, d'abord, la notion des contrats qui est fortement liée à l'approche des services Web. Ensuite, tenant compte que WSLA est dynamique et possède un cycle de vie, cela aidera bien aux travaux utilisant l'ACME Studio et supportant l'aspect dynamique d'ACME. De plus, l'approche services Web est basée sur des langages de communication, de recherche et de description des services Web basés sur XML qui forme un langage robuste, et flexible et largement utilisé dans les technologies Internet. Ainsi, notre formalisation de WSLA essaie de rapprocher son implémentation en ACME à la structure générale du WSLA décrite en XML. Par conséquent, nous pouvons dire que cette simulation permet de qualifier notre formalisation comme une base fiable aux travaux de la vérification formelle avec les versions d'ACME supportant l'aspect dynamique.

8. CONCLUSION ET PERSPECTIVES

Dans cet article, nous nous sommes intéressés à la formalisation de contrats de QdS pour la composition des services Web. Il s'agit d'un sujet ayant l'intérêt continuellement grandissant vu le besoin insistant à la combinaison des services Web afin de composer un système fiable garantissant un certain niveau de qualité de service. Ainsi, notre travail se décompose en deux parties principales. Une partie modélisation de la composition des services Web avec CQML et WSLA et une partie formalisation dont l'objectif était d'illustrer et mettre en évidence, sur un exemple pertinent (le système de composition des services Web pour la réservation de voyages en ligne), l'intérêt de vérifier formellement les contrats de qualité avec l'outil ACME Studio supportant l'ADL ACME/ARMANI.

À l'issue de ce travail, nous pouvons affirmer que, dans le cadre de contrat de QdS, une formalisation rigoureuse, basée sur la définition et la vérification des QdS des services Web, est très utile voire indispensable pour le développement et l'évaluation objective de la composition des services Web.

Quant aux perspectives de ce travail, nous envisageons les prolongements suivants :

– Dans la continuité directe du travail proposé, notre formalisation de WSLA forme une base fiable pour être réutiliser sous les versions de l'outil ACME Studio supportant l'aspect dynamique de l'ADL ACME/ARMANI. En effet, la version *Dynamic ACME* est une extension d'ACME supportant la reconfiguration dynamique des architectures logicielles. Aussi, une reconfiguration à base de *"Plastik"* introduit des extensions dans ACME notamment pour déconnecter et retirer des éléments de l'architecture, les opérations d'ajout et de connexion étant déjà présentes dans le langage initial pour la description des configurations statiques [17].

– Offrir une vérification formelle de QdS sur l'ensemble des services Web (c'est-à-dire sur le service composite) en définissant une qualité de service moyenne applicable sur le système entier.

9. REFERENCES

[1] Behzad Akbarpour, Abdelkader Dekdouk, and Sofiène Tahar. Formalization of cadence spw fixed-point arithmetic in hol. In *IFM*, pages 185–204, 2002.

[2] A. Beugnard, J.-M. Jézéquel, N. Plouzeau, and D. Watkins. Making components contract aware. *IEEE Computer*, 13(7), July 1999.

[3] Adel BOUKHADRA. La composition dynamique des services web sémantiques a base d'alignement des ontologies owl-s. Master's thesis, Ecole National Supèrieur d'Informatique, 2011.

[4] Peter F Brown and Rebekah Metz Booz Allen Hamilton. Reference model for service oriented architecture 1.0, 2006.

[5] Christophe Dumez, Mohamed Bakhouya, Jaafar Gaber, Maxime Wack, and Pascal Lorenz. Model-driven approach supporting formal verification for web service composition protocols. *J. Network and Computer Applications*, 36(4) :1102–1115, 2013.

[6] Kaouthar FAKHFAKH. *Approche sémantique basée sur les intentions pour la modèlisation, la négociation et la surveillance des contrats de qualité de service*. PhD thesis, Université de Toulouse, Ecole Nationale d'Ing 'enieurs de Sfax, 2011.

[7] Robert W. Floyd. Assigning meanings to programs. In J. T. Schwartz, editor, *Mathematical Aspects of Computer Science*, volume 19 of *Proceedings of Symposia in Applied Mathematics*, pages 19–32, Providence, Rhode Island, 1967. American Mathematical Society.

[8] David Garlan, Robert Monroe, and David Wile. Acme : an architecture description interchange language. In *CASCON First Decade High Impact Papers*, CASCON '10, pages 159–173, Riverton, NJ, USA, 2010. IBM Corp.

[9] David Garlan, Robert T. Monroe, and David Wile. *Acme : Architectural Description of Component-Based Systems*, chapter Acme : Architectural Description of Component-Based Systems, pages 47–68. Cambridge University Press, 2000.

[10] Mohamed Graiet, Raoudha Maraoui, Mourad Kmimech, Mohamed Tahar Bhiri, and Walid Gaaloul. Towards an approach of formal verification of mediation protocol based on web services of mde type. *IJWIS*, 8(1), 2012.

[11] C. A. R. Hoare. An axiomatic basis for computer programming. *Commun. ACM*, 12(10) :576–580, 1969.

[12] Hatem Hadj Kacem, Imen Loulou, and Ahmed Hadj Kacem. A formal approach to model and verify the behaviour of publish/subscribe architectural style. *IJITCC*, 2(3) :234–252, 2012.

[13] Raman Kazhamiakin, Paritosh K. Pandya, and Marco Pistore. Timed modelling and analysis in web service compositions. In *ARES*, pages 840–846, 2006.

[14] Alexander Keller and Heiko Ludwig. The wsla framework : Specifying and monitoring service level agreements for web services. *J. Network Syst. Manage.*, 11(1) :57–81, 2003.

[15] Mourad Kmimech. *Vérification d'assemblages de composants logiciels : Application aux modèles de composants UML2.0 et Ugatze*. PhD thesis, Université de paul et des Pays de l'Adour, 2010.

[16] D. Davide Lamanna, James Skene, and Wolfgang Emmerich. Slang : A language for defining service level agreements. In *Proceedings of the The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems*, FTDCS '03, pages 100–, Washington, DC, USA, 2003. IEEE Computer Society.

[17] Marc Léger, Thomas Ledoux, and Thierry Coupaye. Reliable dynamic reconfigurations in a reflective component model. In *CBSE*, pages 74–92, 2010.

[18] Heiko Ludwig, Asit Dan, and Robert Kearney. Cremona : An architecture and library for creation and monitoring of ws-agreents. In *Proceedings of the 2Nd International Conference on Service Oriented Computing*, ICSOC '04, pages 65–74, New York, NY, USA, 2004. ACM.

[19] David Martin, Mark Burstein, Jerry Hobbs, Ora Lassila, Drew McDermott, Sheila McIlraith, Srini Narayanan, Massimo Paolucci, Bijan Parsia, Terry Payne, Evren Sirin, Naveen Srinivasan, and Katia Sycara. Owl-s : Semantic markup for web services. Internet [http ://www.w3.org/Submission/2004/SUBM-OWL-S-20041122/], 2004.

[20] Bertrand Meyer. Applying "design by contract". *Computer*, 25(10) :40–51, October 1992.

[21] OMG. Object constraint language (ocl) specification, version 2.2, 2010.

[22] Mike P. Papazoglou and Willem-Jan Heuvel. Service oriented architectures : Approaches, technologies and research issues. *The VLDB Journal*, 16(3) :389–415, July 2007.

[23] ENST ENST-BRETAGNE FRANCE TELECOM RD INRIA LIFL SOFTEAM) Projet ACCORD (CNAM, EDF RD. Etat de l'art sur les langages de description d'architecture (adls).

[24] Maraoui Raoudha. Formalisation du protocole de mation pour la composition des services web avec ladl acme/armani. Technical report, Faculté des Sciences de Monastir, 2010.

[25] Sumant Tambe, Akshay Dabholkar, and Aniruddha Gokhale. Cqml : Aspect-oriented modeling for modularizing and weaving qos concerns in component-based systems. *Engineering of Computer-Based Systems, IEEE International Conference on the*, 0 :11–20, 2009.

[26] Vladimir Tosic, Bernard Pagurek, and Kruti Patel. Wsol - a language for the formal specification of classes of service for web services. In *ICWS*, pages 375–381, 2003.

[27] Vladimir Tosic, Kruti Patel, and Bernard Pagurek. Wsol - web service offerings language. In *Revised Papers from the International Workshop on Web Services, E-Business, and the Semantic Web*, CAiSE '02/ WES '02, pages 57–67, London, UK, UK, 2002. Springer-Verlag.