Composition de services Web

Vérification de contraintes de qualité de services dans les composition de services Web

Gueye Bassirou, Niang Ibrahima, Déye Ould

Département de Mathématiques et d'Informatique
Université Cheikh Anta Diop
Dakar
SENEGAL
bassirou.gueye@ucad.edu.sn, iniang@ucad.sn, mohamed.oulddeye@ucad.edu.sn

RÉSUMÉ. Le challenge réel d'une composition de services web, dans un environnement ouvert tel que l'Internet, est d'assurer une haute qualité d'exécution. En effet, la qualité de service offert par les services Web composites (généralement définie dans des contrats de services ou SLA), est devenu la priorité absolue pour tout prestataire et partenaire de services. Précisément, le temps de réponse d'une composition de services web demeure un problème crucial. Or, les travaux existants se limitent tout simplement à la proposition de formules analytiques de temps de réponse de quelques constructeurs du langage BPEL. Il n'existe donc pas d'outils encore moins de modèles permettant aux concepteurs de faire leurs vérification de qualité de service.
Ainsi, cet article propose un Framework pour la vérification de contraintes qualité de service d'une composition de services web. Dans le but d'offrir un outil automatisé qui permet d'estimer efficacement le temps de réponse d'une composition de services, nous utilisons la technique de "parsing" du fichier de composition en question.
Signalons que notre proposition couvre intégralement la dernière et récente spécification du langage WSBPEL-v2.0 qui a été adoptée en avril 2007 comme standard d'OASIS [1].

ABSTRACT. The real challenge of a web services composition in an open environment like the Internet, is to ensure high quality execution. Indeed, the quality of service offered by Web services composite (generally defined in Service Level Agreement (SLA)), became the absolute priority for any service provider and partner. Specifically, the response time of a web services composition remains a crucial problem. However, the related work are limited simply to the proposed analytical formulas for response times of a few manufacturers of BPEL. There is therefore no tools even fewer models that allow designers to make their quality audit service.
Thus, this paper proposes a Framework for the verification of quality of service constraints of a web services composition. In order to provide a model for estimating the effective response time of a service composition, we use the technique of parsing file composition in question.
Note that our proposal fully covers that last and recent language specification WSBPEL v2.0 which was adopted in April 2007 as OASIS Standard [1]

MOTS-CLÉS : composition web services, BPEL, parser, qualité de service, temps de réponse, SLA.

KEYWORDS : Web services composition, BPEL, parser, quality of service, response time, SLA.

1. Introduction

La vision future de l'Internet est de transformer le réseau d'informations en un réseau de services. Cette évolution du monde informatique a entraîné le développement de nouveaux paradigmes d'interaction entre différentes applications. En effet, les services Web sont des technologies émergentes et prometteuses pour le développement, le déploiement et l'intégration d'applications Internet. Ces technologies, basées sur XML, fournissent une infrastructure pour décrire (WSDL [2] : Web Services Description Language), découvrir (UDDI [3] : Universal Description, Discovery and Integration) et invoquer (SOAP [4] : Simple Object Access Protocol) des services. Un des avantages majeurs des services Web par rapport middlewares traditionnels (CORBA, DCOM et XML-RPC) est l'apport de l'interopérabilité sur Internet. Les services Web ont pour vocation de favoriser une architecture orientée service (Service Oriented Architecture, ou SOA [5]), intégrant des systèmes complexes hétérogènes, fortement distribués et pouvant coopérer sans recourir à une intégration spécifique et coûteuse. Ce sont des applications accessibles sur Internet réalisant chacune une tâche spécifique. Toutefois, pour certains types d'applications, il est nécessaire de combiner un ensemble de services web élémentaires en des services Web composites (ou agrégés) dans le but de répondre à des exigences plus complexes et pour ne former, du point de vue de l'utilisateur, qu'un seul service Web.

Cependant, la notion de Qualité de Service (ou QoS) qui s'intéresse à la qualité de la relation entre un service et ses clients constitue un enjeu crucial. En fait, bien qu'il existe déjà d'importants travaux autour des compositions de services, qui ont notamment permis l'élaboration du standard BPEL4WS (Business Processus and Execution Languages For Web Services), le problème de la gestion de la QoS dans les compositions de services manque de solutions flexibles, réutilisables et offrant un degré d'abstraction approprié. En effet, l'expressivité du langage BPEL vise uniquement les aspects fonctionnels des compositions. Or, dans le contexte des architectures orientées services où les acteurs se connaissent peu, les enjeux liés à la garantie de la qualité de service sont aussi fondamentaux que ceux liés à l'assemblage fonctionnel des services. Ainsi, à l'instar des traitements métiers, les caractéristiques de qualité de service des services web doivent faire rencontrer l'offre et la demande définies généralement dans des contrats de services (SLA : Service Level Agreement). Plus spécifiquement, un SLA [6] consiste en un document généré à l'issu d'un protocole de négociation (plus ou moins complexe) entre le consommateur et le fournisseur de service, et qui permet de définir les offres et exigences de ces deux rôles. Les spécifications de ce document doivent être vérifiées à l'exécution via un mécanisme de monitoring.

La notion qualité de service fait référence à diverses préoccupations telles que le temps de réponse ou encore la disponibilité. Le temps de réponse constitue est un facteur important dans le domaine des services web car sa prise en charge induit celle de la disponibilité au moment actuel. Effectivement, le fait d'invoquer un service web pouvant se trouvé n'importe où sur internet et obtenir une réponse, sous entend que ce dernier est disponible. Ainsi, la complexité grandissante de la qualité de service, plus particulièrement du temps de réponse et son intérêt, tant pour les fournisseurs que pour les clients, nécessitent le développement d'outils permettant sa gestion, car la dégradation de cette dernière peut engendrer de sérieuses conséquences dont un impact économique important.

Le travail présenté dans cet article se situe justement dans le cadre de développement d'un Framework pour la vérification de contraintes qualité de service d'une composition de services web.

Le reste du document est organisé comme suit. Après un examen de la littérature dans la section 2, la section 3 présente l'architecture de notre système. Section 4 détails les différents constructeurs BPEL mis en jeux pour l'estimation du temps de réponse d'une composition. Dans la section 5, nous présentons les algorithmes développés pour la conception de notre "Framework". La section 6 valide expérimentalement notre modèle. Enfin, la section 7 conclut ce travail.

2. Travaux connexes

Les services web sont un domaine qui suscite l'intérêt de nombreux organismes de recherche académiques et industriels. La majeur partie des travaux dans ce domaine sont concentrés sur les compositions de services web et leurs temps de réponse.

Les auteurs de [7] proposent des formules analytiques de temps de réponse des différents constructeurs du langage BPEL. Notons qu'une bonne vision d'ensemble de leurs travaux fut décris par les auteurs de [8].

Les auteurs de [9] proposent une extension du modèle de Menascé [10] qui modélise un service Web composite comme un ensemble de tâches qui s'exécutent en parallèle. Ce groupe de travail considère que le modèle décrit par Menascé n'est acceptable que si tous les services web participant dans la composition peuvent être exécutés de manière indépendante. Ce qui n'est généralement pas le cas. Ainsi, en prenant comme point de départ les travaux de [10], ces auteurs proposent des formules analytiques de temps de réponse des constructeurs <sequence>, <switch> et <flow> du standard BPEL.

Ces auteurs reviennent dans [11] pour étendre leurs travaux antérieurs proposés dans [9]. En effet, ils considèrent que le fait d'avoir pris en considération différents caractéristiques statistiques pour les services, que le nombre de services web invoqué est aléatoire et que les temps de réponse de ces services sont supposés exponentiels avec différents paramètres, constituent de véritables limitations. Pour pallier les insuffisances de leurs études précédentes, ces auteurs considèrent dans [11] que le nombre de services Web élémentaires invoqué peut être variable mais aussi que les temps de réponse de ces services peuvent être représentés avec les lois exponentielles et Heavy-tailed. Ils justifient l'utilisation de la loi de distribution de Heavy-tailed par le fait que les auteurs de [12] aient montré que les temps de réponse des services élémentaires peuvent être modélisés par une telle distribution.

3. Architecture de notre système

Les travaux dans la littérature se limitent tout simplement à la proposition de formules analytiques de temps de réponse de quelques constructeurs BPEL. Il n'existe donc ni d'outils encore moins de modèles pour verifier ou estimer le temps de réponse d'une composition de services web. C'est pour apporter des solutions face à ces limites que nous proposons notre framework dont le cœur est constitué principalement d'un module qui assure la vérification de contraintes QoS d'une compostion de services web. Cependant, offrir une haute qualité d'exécution d'une telle composition n'est pas chose facile. En effet, les services web dits élémentaires, tels qu'ils sont décrits par WSDL, sont conceptuellement limités à des fonctionnalités relativement simples qui sont déclarées comme une collection d'opérations. De plus, les annuaires publics existant n'ont pas encore intégré ce critère temps de réponse dans la représentation des services qu'ils offrent. Ainsi, la

plupart des services web élémentaires n'exposent pas explicitement leurs QoS.
C'est dans ces conditions que nous avons jugé nécessaire d'offrir un autre module qui assure la détermination préalable du temps de réponse des services élémentaires. L'architecture proposée est décrit dans la figure 1.

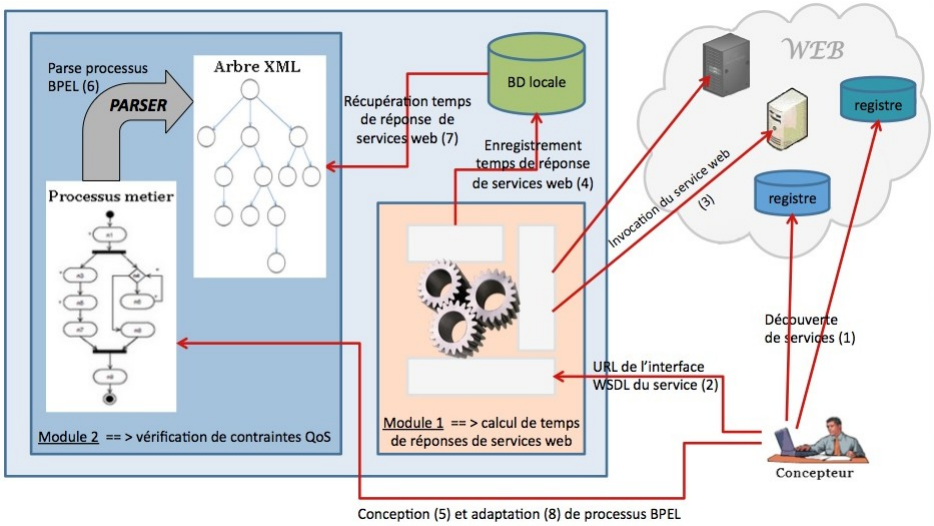

Figure 1. *Architecture de notre framework*

Description de l'architecture :

1) Connexion sur un annuaire pour effectuer une recherche afin de trouver les services susceptibles de participer à une composition. Une fois un service voulu est découvert, on récupère l'adresse de l'interface du fichier de description WSDL.

2) On passe au module 1 l'URL de l'interface du ficher WSDL du service web élémentaire découvert dans l'étape 1.

3) Le module 1 fait le traitement nécessaire afin de determiner le temps de réponse du service. Ce traitement consiste à générer automatiquement, à partir de l'URL du ficher WSDL du service web élémentaire reçu en paramétre, un ensemble de classes (classe "nomservice", classe "nomserviceLocator", classe "nomserviceSoap", classe "nomserviceSoapProxy" et classe "nomserviceSoapStub") servant de base pour le calcul du temps de réponse.
On crée, par la suite, une classe principale utilisant les fonctionnalités des classes précédemment générées. Dans le but d'obtenir un résultat optimal de temps de réponse, on soumet simultanément plusieurs requêtes au serveur hébergeant le service en question. Ainsi, pour chaque requête client cl_i, on obtient un temps de réponse T_i.
Le temps de réponse moyen d'un service web ws_i sera :

$$T[ws_i] = \sum_{i=1}^{k} (\frac{T_i}{cl_i}) \qquad [1]$$

4) Enregistrement du temps de réponse du service web précédemment invoqué.

5) Après assemblage des services web selon un ordre bien définit, on passe au module 2 la contrainte QoS et le processus BPEL décrivant la composition.

6) Le module 2 permet de faire l'estimation du temps de réponse d'une composition de service web ; mais aussi la vérification de contraintes de qualité de service. Pour parvenir à ceux-ci, nous avons utilisé la technique de «parsing» du fichier de composition en question. Les détails du traitement sont décris dans la section 4.

7) Récupération du temps de réponse du service participant dans la composition.

8) Adaptation de l'assemblage au cas où il y'a violation de contraintes QoS et que l'on trouve la nécessité de changer un service (des services) par un autre (d'autres).

4. Constructeurs clés

Le processus BPEL est constitué d'activités ou de constructeurs (simples et structurées) liés par un flot de contrôle. A la suite d'une étude des constructeurs de la spécification BPEL 2.0, nous avons identifiés certains d'entre eux que nous appelons dans ce travail des constructeurs clés. Ces constructeurs clés sont :

– les constructeurs susceptibles de définir un temps d'attente et ou d'exécution : <receive>, <invoke>, <reply>, <wait>, <onAlarm> et <repeatEvery> ;

– et les constructeurs structurés : <sequence>, <scope>, <flow>, <if> avec ses sous constructeurs <elseif> et <else>, <repeatUntil>, <while>, <forEach>, <pick> avec ses sous constructeurs <onMessage> et <onAlarm>, <eventHandlers> avec ses sous constructeurs <onEvent>, <onAlarm> et optionnellement <repeatEvery>.

5. Framework pour la verification de contraintes de qualité de service

L'étude des travaux existant à montrer que les concepteurs manquent d'outils pour effectuer leur vérification de QoS. C'est ainsi que nous proposons un outil automatisé pour la vérification de contraintes QoS d'une composition de services web. L'approche utilisée consiste à "parser" le fichier de composition en question. Ainsi, le module reçoit en entrée une contrainte de qualité de service et un processus BPEL décrivant la composition de services. La première étape consisterait à créer l'arbre XML du processus. Il s'en suit la phase de récupération des nœuds fils directs de la racine :

– Le nœud <partnerLinks> qui définit la liste des partenaires (services web) qui vont participer à la composition.

– Le nœud <sequence> qui constitu le constructeur principal du processus puisqu'il définit l'ordre d'exécution du processus BPEL.

Après avoir initialisé le compteur du temps de réponse, on parcourt l'arbre XML afin de vérifier si l'élément courant correspond à un des constructeurs clés au quel cas on fera appel au gestionnaire concerné. A chaque fois qu'un gestionnaire est appelé, ce dernier effectue son traitement spécifique et retourne le temps de réponse obtenu. Ce temps sera incrémenté au compteur temps de réponse ; et à chaque fois que cette opération est réalisée, on vérifie si la contrainte est toujours respectée. Ainsi :

– S'il y'a respect de contrainte QoS, on continu le traitement.

– S'il y'a violation de contrainte QoS, on le signale au concepteur. Ainsi, ce dernier aura la possibilité d'ajuster la contrainte spécifiée et/ou la composition en question (retour à l'étape d'assemblage).

L'organigramme de la figure 2 décrit notre algorithme de parsing d'un processus BPEL.

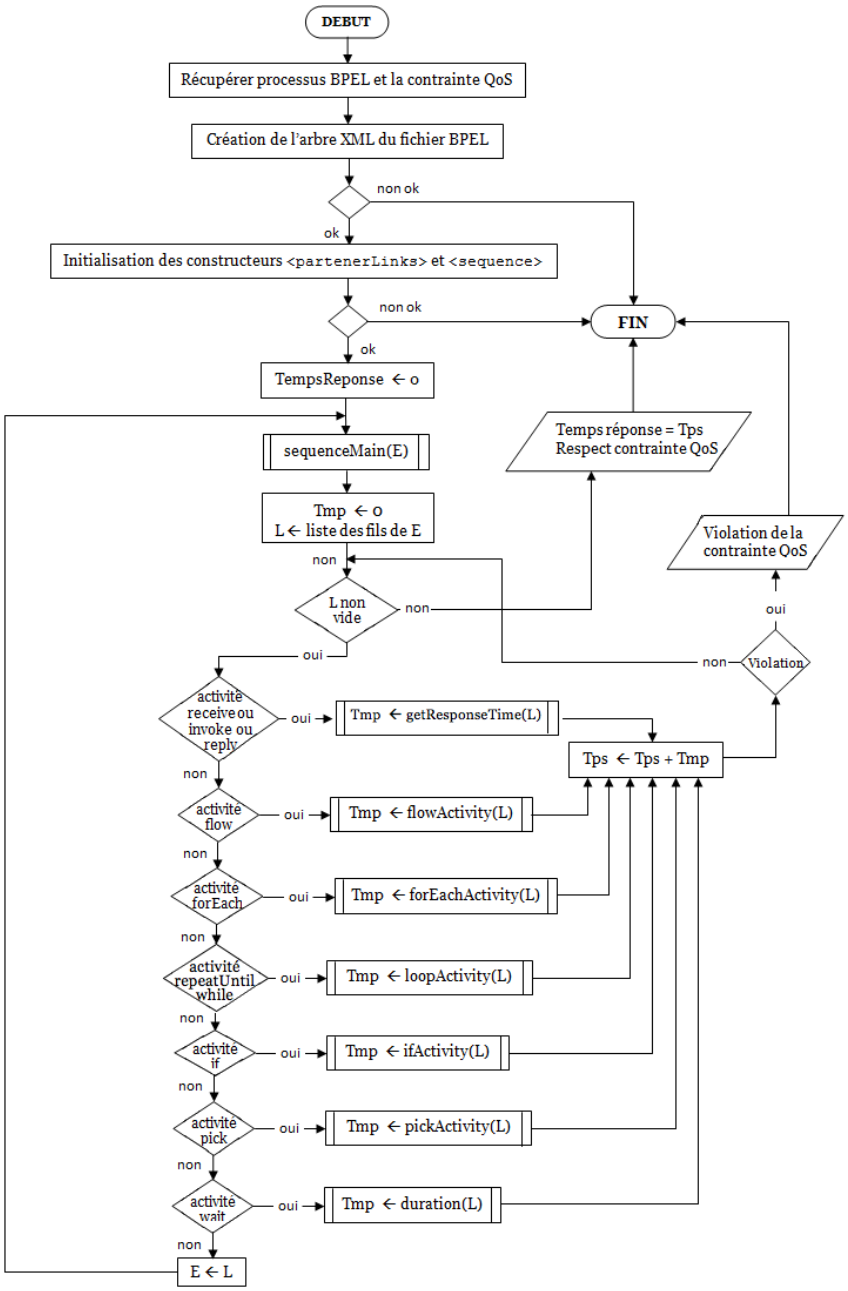

Figure 2. *Algorithme général*

Dans ce qui suit, nous décrivons les gestionnaires (méthodes) définis dans l'algorithme. Pour cela, nous adoptons le formalisme suivant :

$\sum_{i=1}^{n} P[c_i] = 1$; avec $p[c_i]$ la probabilité de passer sur la branche c_i

T[a] : temps de réponse de l'activité a
T_{wait} : temps d'attente d'un délai
T_{body} : temps d'exécution d'une itération de la boucle
k : nombre d'itérations d'une boucle
T_{scopeX} : temps d'exécution du scope "X" d'une activité
$T_{repeatEvery}$: temps d'exécution d'une activité repeatEvery

5.1. Le gestionnaire *getResponseTime*

Ce gestionnaire est invoqué si l'élément courant est un constructeur <receive> ou <invoke> ou <reply> dont le traitement consiste à attendre l'arrivée d'un message d'une source externe (pour le cas <receive>) ou invoquer un service web (pour le cas <invoke>) ou envoyer un message à une source externe (pour le cas <reply>).
Ce gestionnaire fait la correspondance entre l'attribut `partnerLink` définit par le constructeur appelant et l'attribut `name` définit dans le constructeur <partnerLink> afin de connaitre le service web à invoquer. Une autre correspondance sera faite entre l'attribut `name` et celui `partnerLinkType` afin de connaitre l'attribut `opération` du service. Ceci nous permettra ainsi de faire un accès base pour récupérer le temps de réponse du service web.

5.2. Le gestionnaire *flowActivity*

Ce gestionnaire est invoqué si l'élément courant est un constructeur <flow>. Le traitement consiste à exécuter simultanément un ensemble d'activités. L'expression du temps de réponse sera :

$$T[a] = \max(T[a_1]....T[a_k]) \quad [2]$$

On parcourt le fichier, et si l'élément courant est un constructeur :

– <receive> ou <invoke> ou <reply>, on fait appel au gestionnaire ***getResponseTime*** ;

– <sequence> (spécifiant une exécution séquentielle), on fait appel au gestionnaire ***sequenceActivity*** ;

5.3. Le gestionnaire *forEachActivity*

Ce gestionnaire est invoqué si l'élément courant est un constructeur <forEach>. Le traitement consiste à exécuter l'ensemble des activités contenues dans le sous constructeur <scope> exactement k fois. L'expression du temps de réponse sera :

$$T[a] = k * T_{body} \quad [3]$$

Pour déterminer k, on récupère d'abord la valeur de l'attribut `parallel`.

– Si cet attribut vaut "no", alors on récupère les valeurs définies dans les balises <*startCounterValue*> et <*finalCounterValue*> que l'on nommera respectivement k1 et k2 par exemple. Dans ce cas k = k2 - k1 + 1.

– Si cet attribut vaut "yes", alors les k itérations de la boucle seront exécutées de manière parallèle. Dans ce cas, k = 1.
Pour déterminer T_{body}, on fera appel au gestionnaire ***sequenceActivity*** en lui passant en paramètre le sous constructeur <scope>.

5.4. Le gestionnaire *loopActivity*

Ce gestionnaire est invoqué si l'élément courant est un constructeur <while> dont le traitement consiste à exécuter l'ensemble des activités contenues dans sa boucle tant que la condition spécifiée reste vraie ; ou un constructeur <repeatUntil> dont le traitement consiste à exécuter l'ensemble des activités contenues dans sa boucle jusqu'à ce que la condition spécifiée soit fausse. Dans les deux, on ne peut pas connaitre à priori quand est ce que la condition va changer. L'expression du temps de réponse est :

$$T[a] = k * T_{body} \quad [4]$$

Discussion sur la valeur de k.

– Dans certains travaux [13], [14], les auteurs ont préférés donner une valeur globale de temps d'exécution de la boucle. L'avantage est le fait qu'ils n'auront pas à dépiler la boucle pour le calcul du temps d'exécution d'une itération. Cependant, l'inconvénient est qu'avec cette valeur estimative aléatoire, le taux d'imprécision peut être très élevé.

– Dans d'autres travaux [15], [16], les auteurs ont choisis de donner une valeur fixe de k. L'avantage ici est que la précision portera seulement sur la valeur fixée de k et non sur le temps d'exécution du corps de la boucle. Cependant, l'inconvénient est que cette valeur figée est trop restrictive car les boucles n'ont pas les mêmes contextes d'exécution.

– Les autres travaux notés dans la littérature n'ont pas préciser comment ils traitent les boucles.

Ainsi, nous écartons les propositions faites par le premier groupe. Nous rejoignons le second groupe mais en proposant une valeur flexible de k. Par exemple pour le cas d'une authentification, le concepteur peut décider de prendre une valeur de k = 3.
Pour aider le concepteur dans la prise de décision, nous avons proposé un algorithme qui en "parsant" le fichier BPEL, lorsqu'on rencontre un constructeur <while> ou <repeatUntil>, on fait appel au gestionnaire ***sequenceActivity*** pour la détermination du temps d'exécution du corps de la boucle. Ensuite, en fonction de ce temps, on détermine le nombre maximum d'itérations possibles qui ne violent pas la contrainte spécifiée. Ainsi, en fonction de ce nombre, le concepteur pourra choisir une valeur de k convenable.

5.5. Le gestionnaire *ifActivity*

Ce gestionnaire est invoqué si l'élément courant est un constructeur <if>. Le choix exclusif est un point dans le processus où une branche <if>, <elseif> ou <else> est choisie parmi plusieurs. Ainsi, l'expression du temps de réponse est :

$$T[a] = \sum_{i=1}^{k} (T[a_i] * p[c_i]) \quad [5]$$

Le choix de la branche est fait à l'aide d'une décision prise au moment de l'exécution, et il est basé sur une donnée du processus. On ne connait pas ainsi à priori la branche qui sera exécutée ; donc le temps de réponse sera :

$$T[a] = \max(T[a_1]....T[a_k]) \quad [6]$$

5.6. Le gestionnaire *pickActivity*

Ce gestionnaire est invoqué si l'élément courant est un constructeur <pick>. Le traitement consiste à attendre l'occurrence d'un message <onMessage> parmi une liste de

messages possibles ou du signal marquant la fin d'un temporisateur défini dans <onAlarm>. Ce constructeur est donc constitué de plusieurs branches évènement/activité et une des branches va être sélectionner lorsque l'évènement correspondant survient. Ainsi,

$$T[a] = \sum_{i=1}^{k} (p[c_i] * (T_{wait} + T[a_i])) \quad [7]$$

Puisqu'on ne connait pas à priori quand est ce qu'un événement va être déclencher, alors le temps de réponse sera :

$$T[a] = T_{wait} + \max(T[a_1]....T[a_k]) \quad [8]$$

5.7. Le gestionnaire *duration*

Ce gestionnaire est appelé soit par un constructeur <wait>, soit par un sous constructeur <onAlarm>. L'attribut `for` spécifie la durée totale du délai avant la reprise du processus (pour le cas de <wait>) ou le déclenchement de l'alarme (pour le cas de <onAlarm>). La date et l'heure de fin sont précisées en affectant une valeur à l'attribut `until`.

5.8. Le gestionnaire *sequenceActivity*

C'est ce gestionnaire qui traite toutes les activités qui s'exécutent séquentiellement.

Remarque : Le traitement du constructeur <eventHandlers> consiste à attendre l'occurrence d'un événement <onEvent> parmi une liste d'événements possibles ou du signal marquant la fin d'un temporisateur définit dans <onAlarm>. Si aucun événement ne se déclenche jusqu'à l'expiration du deadline, alors les activités associées à <onAlarm> seront exécutées. Dans ce cas, si le sous constructeur optionnel <repeatEvery> est défini, alors le traitement sera répété tant que le scope parent de l'activité <eventHandlers> reste active. L'expression du temps de réponse sera :

$$T[a] = \sum_{i=1}^{k} (p[c_i] * (T_{wait} + k * (T_{scopeAlarm} + T_{repeatEvery}))) \quad [9]$$

où : $k = [\frac{T_{scopeParent} - T_{wait}}{T_{scopeAlarm} + T_{repeatEvery}}]$
Ainsi, dans le cas non périodique c'est-à-dire $T_{repeatEvery}$ = 0, alors k = 1.

Notons cependant que les activités eventHandlers sont invoquées concurremment à l'activité normale du processus. C'est pour cela que ce constructeur n'est pas pris en compte lors de la mise en œuvre puisque son temps n'impactera pas sur le temps de réponse d'une composition.

6. Implémentation

Nous avons implémenté notre framework en utilisant le langage de programmation java (jdk version 1.6) et l'environnement d'édition Eclipse (version 3.4.2). Pour rendre cet environnement praticable, nous avons utilisé les plugins `WTP`(Web Tools Platform) version 3.2.0, `EMF`(Eclipse Modeling Framework), `GEF`(Graphical Editing Framework), `GMF`(Graphical Modeling Framework) et BPEL Visual Designer.

Les autres outils sont : Tomcat (version 6.0.20), Axis2, PostgreSQL (version 8.4), Apache ODE (Orchestration Directeur Engine) version 1.3.4 et le parser JDOM (version 1.6.1) qui est une API open source Java dont le but est de manipuler un document XML. Il utilise DOM[1] pour manipuler les éléments d'un Document Object Model spécifique (créé grâce à un constructeur basé sur SAX) et collections SAX[2] pour parser les fichiers XML.

7. Conclusion

Les services Web constituent une formidable évolution technologique, permettant d'assurer l'interopérabilité des applications sur internet. Le développement et l'adoption de technologies associées aux services web permettent aux entreprises d'implanter de nouvelles applications en composant des services existants.

Afin de permettre une haute qualité d'exécution d'une composition de services, nous avons proposé un outil automatisé de vérification de contraintes QoS de cette dernière. L'objectif recherché à travers l'utilisation de notre framework, est de permettre aux concepteurs de pouvoir évaluer la qualité de service de leurs compositions afin de mieux répondre aux besoins de leurs clients. De plus, de nombreux services existants sur Internet sont exposés à des contrats ou accords de niveaux de services (SLA : Service Level Agreement) liant les fournisseurs aux clients de services. Ainsi, cet outil peut aider le concepteur ou le fournisseur dans la prise de décision lors de tels accords mais aussi il peut lui permettre d'assurer sa part du contrat (réduire, voire éviter les violations de contraintes QoS).

8. Bibliographie

[1] « Web services business process execution language version 2.0 », *OASIS standard, April 2007.*

[2] R. CHINNICI, J.-J. MOREAU, R. ARTHUR, S. WEERAWARANA, « Web Services Description Language (WSDL) Version 2.0 », *W3C Recommendation 26, June 2007.*

[3] L. CLEMENT, SYSTINET, A. HATELY, C. VON RIEGEN, T. ROGERS, « Computer Associates. UDDI v.3.0.2. OASIS Specification », *October 2004.*

[4] M. GUDGIN, M. HADLEY, J.-J. MOREAU, H. FRYSTYK NIELSEN, « Simple Object Access Protocol (SOAP) Version 1.2. W3C », *July 2001.*

[5] F. CURBERA, M. DUFTLER, R. KHALAF, W. NAGY, N. MUKHI, S. WEERAWARANA, « Unraveling the Web Services Web : An Introduction to SOAP, WSDL, and UDDI », *IEEE Internet Computing*, vol. 6, n° 2, pp. 86-93, 2006.

[6] L. JIE JIN, V. MACHIRAJU, A. SAHAI, « Analysis of service-level agreement for web services », *Technical Report HPL-2002-180, 2002.*

[7] D. RUD, M. KUNZ, A. SCHMIETENDORF, R. DUMKE, « Performance Analysis in WS-BPEL-Based Infrastructures », *Department of Distributed Systems, Berlin School of Economics, Germany. In 23rd UK Performance Engineering Workshop, 2007.*

[8] F. VAN BREUGEL, M. KOSHKINA, « Models and verification of BPEL », *September 2006.*

[9] S. HADDAD, L. MOKDAD, S. YOUCEF, « Response-time analysis of composite Web services », *In Proceedings, IEEE Computer Society, 23-25 July 2008.*

[10] D.A. MENASCÉ, « Response-time analysis of composite Web services », *IEEE Internet Computing*, vol. 8, n° 1, pages 90-92, 2004.

1. www.w3.org/DOM/
2. www.megginson.com/SAX/index.html

[11] S. HADDAD, L. MOKDAD, S. YOUCEF, « Response time of BPEL4WS constructors », *In Proceedings of the 15th IEEE Symposium on Computers and Communications (ISCCÕ10). pages (695-700), 2010.*

[12] U. VALLAMSETTY, K. KANT, P. MOHAPATRA, « Characterization of e-commerce traffic », *Electronic Commerce Research*, vol. 3, n° 1-2, pp. 167192, 2003.

[13] L. ZENG, B. BENATALLAH, A. H. H. NGU, M. DUMAS, J. KALAGNANAM, H. CHANG, « QoS-aware middleware for web services composition », *IEEE Transactions on Software Engineering. pages (311-327), 2004.*

[14] G. CANFORA, M. DI PENTA, R. ESPOSITO, L. VILLANI, « A framework for QoS-aware binding and re-binding of composite web services », *Journal of Systems and Software, 81(10) :1754-1769, 2008.*

[15] J. CARDOSO, A. P. SHETH, J. A. MILLER, J. ARNOLD, K. J. KOCHUT, « Modeling quality of service for workflows and web service processes », *Web Semantics Journal : Science, Services and Agents on the World Wide Web Journal 1 (3), 281-308, 2004.*

[16] F. BALIGAND, N. RIVIERRE, T. LEDOUX, « A declarative approach for qos- aware web service compositions », *Fifth International Conference on Service-Oriented Computing (ICSOC)*, vol. 4749 of LNCS, pages 422-428. Springer, 2007.