Vérification des propriétés non-fonctionnelles pour l'orchestration de services web

Wael Sellami, Hatem Hadj Kacem, Ahmed Hadj Kacem

Vérification des propriétés non-fonctionnelles dans une orchestration de services web

Wael Sellami
Université de Sfax
Unité de recherche ReDCAD
FSEG, Sfax, Tunisie
wael.sellami@gmail.com

Hatem Hadj Kacem
Université de Sfax
Unité de recherche ReDCAD
FSEG, Sfax, Tunisie
hatemhadjkacem@gmail.com

Ahmed Hadj Kacem
Université de Sfax
Unité de recherche ReDCAD,
FSEG, Sfax, Tunisie
ahmed.hadjkacem@fsegs.rnu.tn

RÉSUMÉ

La composition de services est une tâche primordiale dans le développement de systèmes orientés service. L'orchestration se présente comme un ensemble de mécanismes pour la composition d'un nouveau service web formé d'un ensemble de services atteignables. Afin de valider une telle composition, deux classes de propriétés non fonctionnelles doivent être prises en considération à savoir les propriétés génériques et les propriétés spécifiques. Les propriétés génériques peuvent être vérifiées pour tous les services web invoqués dans une orchestration. Les propriétés spécifiques constituent les relations d'interdépendance entre les différentes activités au sein d'un processus d'orchestration. Ces propriétés ne peuvent pas être vérifiées directement sur le processus, l'utilisation donc d'une technique formelle s'avère intéressante. Pour se faire, nous présenterons dans cet article notre approche formelle pour la validation d'une orchestration de services web. L'approche adopte BPEL 2.0 (Business Process Execution Language) comme langage d'orchestration de services web et utilise le model-checker SPIN pour la vérification. La spécification BPEL est traduite en code Promela, le langage de spécification de SPIN, afin de vérifier aussi bien les propriétés génériques que les propriétés spécifiques exprimées en LTL (Linear Temporal Logic). L'outil de transformation de BPEL en Promela est développé en utilisant ANTLR (ANother Tool for Language Recognition). Ce travail a été couronné par le développement de l'outil BpelVT (BPEL Verification Tool) afin de consolider l'approche proposée.

ABSTRACT

Web services are a building block in the sense that they can be composed to form higher level services or applications to achieve business goals. They're called composite when their executions involve interactions with other web services. The web service composition specifies what services need to be invoked, in what order and how to handle exceptional conditions. This composition can be described from both a local or a global perspective by respectively orchestration or choreography. We suggest that the verification of web services orchestration must take into account both generic and specific properties. The generic properties can be checked for any invoked web services when the specific properties are different interdependence relationships between activities within an orchestration process. Since these properties can't be directly verified on the process, so, the use of formal techniques is interesting. Doing so, we will present in this paper our formal approach to validate a web services orchestration. The paper adopts BPEL 2.0 (Business Process Execution Language) as the language to describe the web service orchestration, and uses the SPIN model-checker for the verification engine. The BPEL specification is translated into Promela code, the input language for SPIN model-checker, to verify generic and specific properties expressed with LTL (Linear Temporal Logic). The transformation tool of BPEL to Promela is achieved using ANTLR (ANother Tool for Language Recognition). A tool named BpelVT (BPEL Verification Tool) is developed to support the proposed approach. It provides the BPEL manager, the automated process translation of BPEL to Promela code and model-checking views.

1. INTRODUCTION

Les architectures logicielles font le pont entre la stratégie et le développement des systèmes d'informations [3]. En effet, elles ont toujours constitué un élément de réponse aux exigences de l'évolution des systèmes d'information. Elles sont une réponse efficace aux problématiques que rencontrent les entreprises en terme de réutilisation, d'interopérabilité et de réduction de couplage entre les différents systèmes qui implémentent leurs systèmes d'information.

Une architecture orientée services (SOA pour Services Oriented Architecture) est une architecture logicielle s'appuyant sur un ensemble de services (composants logiciels). Elle a été introduite afin de rendre la conception de logiciels une tâche plus facile, et afin d'assurer la réutilisation de services et d'augmenter l'interopérabilité à grande échelle. En effet, les systèmes d'information sont composés d'un ensemble de fonctions dont chacune est appelée service qui peut interagir avec son environnement à travers l'échange de messages. Cependant, avec l'évolution successive des exigences des entreprises, la valeur réelle d'un service deviendra insatisfaisante, d'où la nécessité d'une nouvelle approche pour la composition de services afin d'apporter une valeur ajoutée. Un service composé assemble divers services et coordonne les interactions au cours de leurs exécutions en vue d'utiliser leurs fonctions et réaliser des tâches plus complexes.

Avec l'avènement de la technologie web, l'intégration des ap-

plications sur internet est devenue une nécessité pour offrir plus de coopération inter-entreprises et plus d'opportunités de collaboration mondiale par l'échange de services. Lorsque ces services sont déployés sur le web, nous parlons alors de services web. Un service web est une technologie modulaire, réutilisable, autonome et indépendante des langages de programmation et de plates-formes. Il permet à des applications de dialoguer à distance via internet avec d'autres applications hétérogènes dans des environnements distribués d'une manière faiblement couplée. Cette composition peut être décrite à partir d'un point de vue local (resp. global), le modèle engendré est appelé orchestration (resp. chorégraphie). Dans l'orchestration, les services web invoqués sont sous le contrôle d'un seul processus central (orchestrateur). Ce processus coordonne l'exécution des différentes opérations sur les services web impliqués dans la composition. Tandis que la chorégraphie désigne un type de composition décentralisée où chaque service est indépendant des autres services.
Cependant, certains conflits peuvent être générés au sein de l'orchestration de services web. Pour surmonter ces problèmes, nous suggérons, pour la validation d'une telle composition, la vérification de deux classes de propriétés non-fonctionnelles : les propriétés génériques et les propriétés spécifiques. La première classe de propriétés est composée des propriétés génériques telles que la vivacité, la sécurité ou la disponibilité. Ces propriétés sont considérées ici comme génériques, puisqu'elles sont vérifiées sur les services invoqués indépendamment des activités qui les composent. La deuxième classe de propriétés qui doit être prise en considération est formée des propriétés relatives aux activités d'un processus de composition. Elles sont considérées ici comme des propriétés spécifiques. À titre d'exemple, nous pouvons considérer le cas d'un processus de paiement en ligne, les deux activités de cryptage, cryptage long et cryptage court, assurent toutes les deux la confidentialité, mais combinées, leur effet serait un message crypté à deux reprises, ce qui génère de graves erreurs au moment du décryptage. Ces deux activités sont reliées par une relation d'interdépendance "choix" et elles ne devraient pas être choisies pour un même processus. Pour le contrôle d'accès aux applications, par exemple, l'activité d'autorisation doit précéder l'activité d'authentification pour décider si l'utilisateur a déjà des droits d'accès ou non. Ces deux activités sont reliées par une relation d'inter-dépendance "précédence".

Dans ce contexte, plusieurs solutions ont été proposées [1, 10, 9, 4]. Ces solutions se sont essentiellement intéressées à la vérification des propriétés génériques. À notre connaissance, nous notons qu'aucune des approches présentées précédemment ne s'est intéressée spécifiquement à la vérification des propriétés spécifiques. Dans ce travail, nous avons surmonté ces difficultés en proposant une approche formelle pour la validation de l'orchestration des services web par la vérification des propriétés génériques et spécifiques. Divers langages ont été proposés pour spécifier l'orchestration de services web parmi lesquels nous pouvons citer le langage BPEL. Ce langage est le plus connu et le plus utilisé dans l'orchestration de services web. Un processus BPEL est composé d'un ensemble d'activités qui définit l'ordre d'invocation des services. Cet ordre peut être séquentiel ou parallèle. Nous proposons d'utiliser le model-checker SPIN pour le processus de vérification. SPIN est un outil de vérification générique qui supporte à la fois la conception et la vérification des systèmes de processus asynchrones.
Une autre contribution importante dans ce travail est le processus de transformation de BPEL vers Promela que nous avons développé en utilisant l'outil ANTLR.

La suite de cet article est organisée comme suit : dans la section suivante, nous introduisons quelques concepts préliminaires tels que le concept d'orchestration, SPIN/Promela et le langage BPEL. Dans la section 3, nous synthétisons notre étude des différentes approches existantes permettant la validation de l'orchestration de services web. L'approche que nous proposons est présentée dans la section 4. Dans la section 5, nous donnons une brève description du processus de transformation de BPEL vers Promela que nous avons développé. Dans la section 6, nous présentons, à travers une étude de cas, la mise en œuvre et la validation de notre approche. Finalement, nous concluons en section 7 et introduisons les perspectives de ces travaux.

2. PRÉLIMINAIRES

2.1 Composition des services web

Les services web sont une technologie clé pour supporter et réaliser des processus métiers. Leur composition peut être étudiée selon deux points de vue complémentaires : (*i*) un point de vue global dans lequel des partenaires entrants dans la composition sont considérés ; nous parlerons alors de la chorégraphie, (*ii*) un point de vue local dans lequel seul le processus interne des services est modélisé ; nous parlerons alors de l'orchestration. Dans ce travail, nous traiterons le cas de l'orchestration.

La chorégraphie a été introduite comme une nouvelle vue sur l'interaction des services. C'est une description abstraite des protocoles tout en offrant une coordination collaborative décentralisée. La chorégraphie qui est de nature descriptive décrit le contrat d'interaction entre deux ou plusieurs services. Ce modèle permet de décrire le comportement des services dans la composition. A ce niveau la littérature distingue deux types de chorégraphies à savoir une chorégraphie locale qui décrit le flux d'un point de vue des partenaires participants et une chorégraphie globale qui définit, à partir d'un point de vue neutre, le processus inter-organisationnels. Contrairement à la chorégraphie, l'orchestration désigne un type de composition centralisé où un seul processus central (orchestrateur) détient la coordination et le contrôle de tous les services web participant à la composition. En effet, les services web invoqués ne savent pas s'ils participent ou non dans une composition [8], seul le coordinateur possède l'information sur l'enchaînement d'exécution du processus métier. Afin de composer des services web, divers langages ont été définis dont le plus célèbre est BPEL4WS. En effet, BPEL4WS (ou BPEL tout court) est un langage d'orchestration de services web créé en 2003 par l'association industrielle entre Microsoft, Sun, IBM et BEA [12]. BPEL est défini pour décrire la logique de contrôle requis pour coordonner des services web participant à un flux de processus. En outre, il est créé par la fusion de deux langages qui sont le langage XLANG de Microsoft et le langage WSFL de IBM pour offrir un vocabulaire riche pour la description des processus métiers.

2.2 Le langage BPEL

BPEL a connu historiquement plusieurs versions telles que

Figure 1: Activités de base [2].

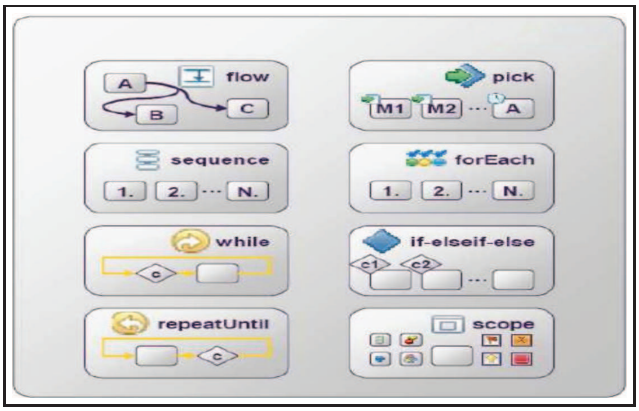

Figure 2: Activités structurées [2].

la version 1.0 en Août 2002, la version 1.1 en Mars 2003 et la version 2.0 standardisée en 2007 par OASIS. C'est un langage utilisé pour décrire l'exécution du processus métier d'une façon synchrone ou asynchrone et de partager les informations communes avec les services web partenaires.
Le but principal de BPEL est de standardiser le processus d'automatisation de la composition des services web au sein des entreprises à travers la définition des activités de base (Figure 1) à savoir invoke, receive et reply et des activités structurées (Figure 2) tels que flow, repeat until et while ainsi que des activités de gestion d'erreurs tels que throw et compensate.

Avec BPEL, il est possible de décrire les processus métiers de deux manières distinctes :

- Les processus métiers exécutables : ce sont des processus qui composent un ensemble de services web existants et précisent l'algorithme exact d'exécution des activités. En effet, les processus exécutables sont importants, car ils présentent les détails exacts du fonctionnement, de remplir l'écart entre les spécifications du processus métier et le code responsable de leur exécution et il présente une solution directe au problème de l'automatisation des processus métiers.
- Les processus métiers abstraits : ce sont des processus qui décrivent l'échange de messages publics entre le processus métier et son environnement (services invoqués) sans spécifier les détails techniques des flux de processus. Les processus métiers abstraits sont généralement définis pour décrire le comportement externe du processus métier avec son environnement (partenaires).

2.3 SPIN/Promela

SPIN [7] est un environnement de spécification et de vérification formelle des systèmes distribués. Il a été développé au début des années 80, aux laboratoires Bell. Cet outil accepte en entrée un programme spécifié avec le langage Promela qui décrit le comportement des différents modules du système (leurs fonctionnements et leurs interactions). Le model-checker SPIN contient un interprétateur permettant de réaliser des simulations aléatoires de l'exécution du système et d'effectuer une vérification exhaustive des propriétés exprimées en LTL. Durant la vérification et la simulation, SPIN vérifie l'absence d'inter-blocages et les terminaisons anormales du code.

Promela est un langage de spécification de SPIN servant à décrire les systèmes distribués tels que les protocoles de communication, les systèmes d'exploitation, les algorithmes concurrents, etc. Ce langage est conçu pour faciliter la construction de modèles de systèmes distribués en soutenant la spécification des structures de contrôle non-déterministe, la création et la communication entre les processus. SPIN permet d'exprimer des propriétés en termes de logique temporelle linéaire (LTL) et de vérifier si le programme satisfait ou non ces propriétés. Une autre caractéristique de SPIN est la simulation visuelle des interactions suivant un chemin d'exécution possible par le système et en présentant la trace d'exécution résultant pour l'utilisateur.

3. TRAVAUX CONNEXES

La vérification des compositions de services web est devenu actuellement un champ à part entière. En effet, de nos jours, plusieurs travaux de recherches accordant une grande importance à la vérification de l'orchestration à travers la définition de différentes propriétés pour mener à une fiable composition de services web. Dans la première partie de cette section, nous revenons sur les travaux existants qui ont abordé la vérification des propriétés non fonctionnelles. La deuxième partie sera dédiée à la présentation de quelques outils de transformation de BPEL vers Promela.

La vérification des propriétés non fonctionnelles est le sujet de quelques travaux de recherche qui ont utilisé différentes techniques d'abstraction et outils de vérification formelle. En effet Hegedus et al. [6] présentent une approche pour la validation de la composition de services web. Pour cela, ils utilisent le model-checker SAL pour la vérification des propriétés de blocage et d'autres propriétés comme les variables non utilisées, la lecture des variables non initialisées et les traces d'exécution. Abousaid et al. [1] proposent une approche basée sur la transformation du processus BPEL en algèbre de processus BP-calcul pour vérifier, par la suite, des propriétés de vivacité, d'équité, de fiabilité, de disponibilité et de sûreté en utilisant le model-checker HAL-Toolkit.

Kovacs et al. [10] présentent un framework pour vérifier le processus BPEL. Ce framework traduit ces processus vers un réseau de flux de données. Ensuite, ils traduisent le modèle généré en code Promela afin de vérifier des propriétés telles que le blocage, l'accessibilité et la lecture des variables non-initialisées en utilisant SPIN.

Dans [13], les auteurs intègrent un algorithme dans l'outil ActiveBPEL pour traduire le processus BPEL vers un modèle d'automate temporisé. Le modèle généré est chargé en entrée par le model-checker UPPAAL pour vérifier les propriétés de blocage, d'accessibilité et aussi des propriétés temporelles.

Kazhamiakin et al. [9] fournissent une approche de vérification des propriétés temporelles pour le processus BPEL. Ils transforment le processus vers un formalisme WSTTS (Web Service Timed State Transition Systems). Ils utilisent ensuite le model-checker NuSMV pour vérifier les comportements temporels.

Dekdouk et al.[4] présentent une approche de vérification formelle, qui utilise la technique de model-checking et la simulation. Pour ce faire, ils utilisent le langage BPEL pour orchestrer les services web et le model-checker SPIN afin de vérifier la propriété de vivacité.

En vue d'interroger le model-checker SPIN, la transformation de BPEL vers la spécification formelle Promela s'avère une tâche primordiale pour vérifier formellement les propriétés désirées. Cette transformation est le sujet de quelques travaux de recherche. Quyet et al. [14] définissent un outil de transformation pour traduire le processus BPEL en un graphe LCFG (Labeled Control Flow Graph). Le modèle généré est traduit, ensuite, vers un code Promela. Dans [15, 4], les auteurs développent un outil de transformation pour traduire directement le processus BPEL en Promela. L'outil présenté dans [11] est composé de deux phases : la première phase, consiste à transformer le processus BPEL vers un automate fini étendu comme modèle intermédiaire. La deuxième phase consiste à traduire cet automate en Promela. La même approche a été adoptée par Fu et al. [5] en transformant dans la première phase le processus BPEL en automata gardé qui sera traduit dans la deuxième phase en Promela.

À travers ce recueil de l'état de l'art, nous remarquons que les travaux qui ont traité la vérification de l'orchestration des services web se sont concentrés uniquement sur la vérification des propriétés génériques telles que la vivacité, la sécurité et l'échange de messages. À notre connaissance, aucun travail de vérification n'a traité le cas des propriétés spécifiques. En effet, ces propriétés visent à étendre la composition de services web en permettant aux utilisateurs de sélectionner le mode d'interaction le plus approprié pour leurs besoins actuels et le cadre dans lequel une interaction spécifique peut se produire ou non.
Notre solution surmonte ces lacunes en permettant la vérification des propriétés génériques et spécifiques. Par ailleurs, nous remarquons que les traducteurs présentés dans [5, 4, 11, 15] ne supportent pas le langage standard BPEL 2.0, mais uniquement BPEL 1.1. Seul le travail de Quyet et al [14] a traduit le processus BPEL 2.0 en Promela. Cependant, ce travail n'est pas robuste du fait qu'il a proposé des règles de transformation ad-hoc qui ne traitent pas toutes les activités présentées dans BPEL 2.0. Par rapport à ce travail, notre outil de transformation peut être considéré comme un outil complet car il traite toutes les activités BPEL, sans passer par un modèle intermédiaire.

Les contributions présentées dans cet article sont de deux volets. Premièrement, nous proposons une approche de vérification, qui prend en compte à la fois les propriétés génériques et spécifiques. Deuxièmement, nous présentons un outil de transformation de BPEL en Promela à l'aide de l'outil ANTLR. Une transformation des deux d'activités BPEL (Partner Link et Repeat Until) est détaillée dans ce travail.[1]

4. APPROCHE PROPOSÉE

L'approche de validation d'orchestration de services web que nous proposons dans cet article est composée principalement de deux phases (voir Figure 3) :

- **Génération du code Promela (Phase 1):** Au niveau de cette étape, seul le processus BPEL est pris en considération afin de générer le code Promela correspondant sans tenir compte des propriétés non-fonctionnelles à vérifier. La transformation de BPEL en Promela est développée à l'aide de l'outil BTP (BPEL To Promela) que nous le détaillerons dans la section 5.

- **Vérification des propriétés (Phase 2):** les propriétés non-fonctionnelles sélectionnées par le concepteur seront prises en considération par le model-checker SPIN afin de s'assurer si la spécification Promela satisfait ces propriétés ou non.

Comme nous l'avons indiqué précédemment, le processus de vérification tient compte de deux classes de propriétés non-fonctionnelles : génériques et spécifiques.

4.1 Propriétés génériques

Les propriétés génériques sont relatives aux services web invoqués sans tenir compte des activités qui les composent. Parmi ces propriétés, nous pouvons citer la vivacité, la sureté ou la disponibilité :

Vivacité Elle signifie qu'un comportement souhaitable peut se produire dans l'avenir.

Sureté Elle signifie qu'un comportement indésirable ne se produira jamais.

Disponibilité Plusieurs travaux de recherche ont abordé le concept de la disponibilité sous différentes définitions. Dans le domaine de l'embarqué, un système est disponible signifie que ses fonctions sont accessibles aux utilisateurs dans des délais prévus. Dans le cadre des systèmes d'information, la disponibilité consiste à garantir que les ressources et les services du système sont disponibles aux utilisateurs autorisés. Selon [1], la disponibilité est la capacité d'un service web à être disponible à tout moment. La définition que nous adoptons dans ce travail est la suivante : *la disponibilité d'un service est sa capacité à achever une tâche suite à une invocation.* En effet, cette définition combine les propriétés de vivacité et de réactivité pour assurer une terminaison réussie d'un service web composé et garantir une requête de réponse pour chaque invocation.

4.2 Propriétés spécifiques

Les propriétés spécifiques ou d'interdépendances sont des propriétés qui peuvent être vérifiées au niveau des activités d'un service web composé. Elles désignent le type de relations entre ces activités. Elles permettent aux utilisateurs de sélectionner dynamiquement le mode d'interaction le plus approprié pour leurs besoins actuels et le cadre dans lequel une interaction spécifique peut se produire ou non. Ces propriétés peuvent être classées en trois sous-classes :

[1]La transformation des autres activités est disponible à l'adresse : http://www.redcad.org/software.html

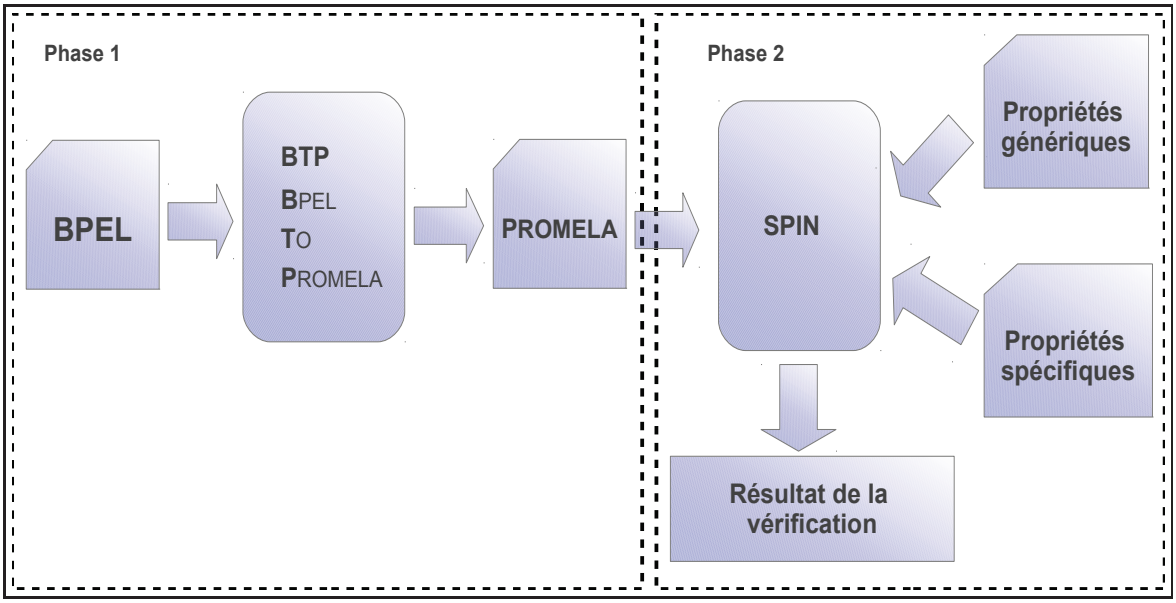

Figure 3: Approche formelle de vérification.

1. **Propriétés d'exclusion**
   Étant donné deux activités, les propriétés d'exclusion désignent qu'une activité ne doit pas être exécutée si une autre a eu lieu. "Choice", "mutex" et "conflict" sont trois alternatives des propriétés d'exclusion.

   - ***Choice*** La propriété d'interdépendance "choice" entre deux activités signifie que les deux activités réalisent essentiellement les mêmes actions. Ainsi, elles ne devraient pas être exécutées ensemble parce qu'elles vont provoquer des problèmes d'exécution.
   - ***Mutex*** La propriété d'interdépendance "mutex" signifie que si une activité a été exécutée, alors l'exécution de l'autre activité ne doit pas avoir lieu sinon elles vont engendrer des erreurs. Cette propriété diffère de la propriété "choice" du fait que les activités n'assurent pas les mêmes actions.
   - ***Conflict*** Cette propriété d'interdépendance entre deux activités données peuvent avoir lieu lorsque l'une affecte une activité d'une manière positive et l'autre affecte la même activité d'une manière négative.

2. **Propriétés d'inclusion**
   Les propriétés d'inclusion sont des relations entre les activités dans lesquelles l'exécution d'une activité nécessite l'exécution de l'autre activité pour avoir plus de valeur ajoutée. "Requirement" et "assistance" sont deux exemples de propriétés d'inclusion.

   - ***Requirement*** La propriété "requirement" entre deux activités signifie que l'utilisation de l'une nécessite l'utilisation de l'autre. (indépendamment de l'ordre d'exécution).
   - ***Assistance*** Cette propriété d'interdépendance entre deux activités signifie qu'une activité peut assister une autre activité, si et seulement si elle a un impact positif sur ses fonctionnalités.

3. **Propriété de précédence**
   La propriété d'interdépendance "précédence" entre deux activités est un cas particulier de la propriété "requirement" puisqu'elle tient compte de l'ordre d'exécution des activités.

5. MISE EN OE UVRE DE LA TRANSFORMATION

Dans cette section, nous présentons notre outil BTP (**B**PEL **T**o **P**romela). C'est un traducteur à base de grammaire développé à l'aide de l'outil ANTLR. Cet outil ANTLR permet de générer un reconnaisseur du langage défini par une grammaire à travers la lecture des flux d'entrée et la génération des erreurs si un flux d'entrée n'est pas conforme à la syntaxe spécifiée par la grammaire.

Afin de réaliser une analyse lexicale et syntaxique, ANTLR répartit la grammaire en deux parties principales qui sont Lexer et Parser.

- Lexer : c'est l'ensemble des règles de la grammaire qui permet d'analyser la structure des caractères du fichier d'entrée (stream of caracters) et les convertir en des jetons (stream of tokens) pour être analysés plus tard par le parseur. Les lexers sont souvent décrits en majuscule.

- Parser : c'est l'ensemble des règles formées par les jetons produits par les lexers afin de construire un arbre AST (Abstract Syntax Tree) de toute la grammaire. Les parsers sont souvent décrits en minuscule.

Afin d'expliquer les différentes phases de l'outil, deux exemples sont mis en jeu à savoir Partner Link and Repeat Until.

5.1 L'analyse lexicale

Cette étape consiste à déterminer la grammaire du langage BPEL dans un format défini par l'outil ANTLR dans lequel

chaque activité BPEL est transformée en un attribut dans la règle correspondante.

Partner Link Les partner links désignent les services web qui communiquent avec le processus BPEL. Le "Partner Link Type" précise le type de partenaire invoqué, le champ "PartnerRole" précise le rôle du web service sollicité et le champ "MyRole" qui est spécifié uniquement dans le cas d'un appel asynchrone permet de préciser le rôle du BPEL appelant.

```
<bpel:partnerLink name="name_Link"
partnerLinkType= "name_PLT"
myRole="name_MR"
partnerRole= "name_PR">
</bpel:partnerLink>
```

La règle de grammaire correspondante est la suivante :

```
BEGIN_partnerLink name
partlinktyp(myRole)?(partnerRole)?
END_partnerLink
```

"PartnerRole" et "MyRole" sont définis comme des attributs optionnels à l'aide de l'opérateur "?" qui indique que ces attributs peuvent être répétés zéro ou plusieurs fois.

Repeat Until C'est une activité structurée qui permet d'exécuter plusieurs fois de suite une même séquence d'actions. La condition est évaluée après chaque itération.

```
<bpel:RepeatUntil name="repun">
activities
<bpel:condition>
<![CDATA[Expr_cond]]>
</bpel:condition>
</bpel:repeatUntil>
```

La règle de grammaire de l'activité Repeat Until est la suivante :

```
BEGIN_repeatUntil
name (activities)+ condition
END_repeatUntil
```

L'opérateur "+" est utilisé pour indiquer que le bloc "activities" peut être exécuté zéro ou plusieurs fois.

5.2 L'analyse syntaxique

cette étape consiste à compléter la grammaire par des règles de réécriture (rewrite rules). En effet, elle se base sur la définition d'un jeton et sur une règle de réécriture pour chaque règle de la grammaire afin de construire l'arbre AST (Abstract Syntax Tree). Afin de structurer l'arbre AST, chaque règle de réécriture se définit par un nœud racine et des nœuds fils.

Partner Link Pour l'activité partner link, la règle de réécriture est la suivante :

```
partnerlink: ->^(PARTTOKEN name
partlinktyp
(myrole)? (partnerRole)?);
```

Repeat Until La règle de réécriture de l'activité structurée Repeat Until est la suivante :

```
repeatuntil: ->^(REPTOKEN name
(Activities)+ condition);
```

Le jeton PARTTOKEN (resp. REPTOKEN) de l'activité Partner Link (resp. Repeat Until) est inséré dans l'ensemble des jetons Tokens :

```
Tokens{PARTTOKEN; REPTOKEN}
```

5.3 Génération de code

Cette étape consiste à définir le TreeWalker dans laquelle nous définissons le code sortie correspondant à chaque règle de réécriture de la grammaire.

Partner Link Le code Promela correspondant à l'activité Partner Link est le suivant :

```
partnerlink:->^(PARTTOKEN name
partlinktyp
(myrole)? (partnerRole)?)
{chan name_Link_IN = [0] of
{mtype, type_var};
chan name_Link_OUT = [0] of
{mtype, type_var};};
```

En Promela, la communication est assurée à travers des canaux de communication. Pour cela, nous avons défini pour chaque "Partner link" deux canaux : un canal de type OUT pour la communication entre le processus BPEL et le service web invoqué et un canal de type IN qui permet aux services web invoqués de communiquer avec le processus BPEL.

Repeat Until Pour l'activité Repeat Until, le code Promela correspondant est le suivant :

```
if-statement:->^(REPTOKEN name
(activits)+ condition){
do
:: activities ;
:: Expr_Cond-> break ;
od;};
```

En Promela, elle garde la même structure utilisée en BPEL en la traduisant en une boucle "do". Lorsque la condition est vérifiée, nous définissons l'instruction "break" pour arrêter l'exécution et quitter la boucle.

6. IMPLÉMENTATION ET VALIDATION

Pour mettre en œuvre notre approche de vérification formelle dans un contexte pratique, nous proposons un outil de vérification BPELVT (BPEL Verification Tool)[2] afin de valider l'orchestration de services web. Cet outil est composé de trois modules 4 :

Gestionnaire BPEL L'outil prend en entrée une spécification BPEL encodée en XML. Ensuite, ce fichier est analysé pour extraire toutes les activités.

Génération de code Une fois la spécification BPEL est chargée, le code Promela sera généré à l'aide de BTP (BPEL To Promela) développé en utilisant l'outil ANTLR et détaillé dans la section 5.

Processus de vérification A ce niveau, un ensemble de propriétés génériques, spécifiques et composées sont à la disposition de l'utilisateur afin de déclencher le processus de vérification. Deux types de vérification sont possibles :

- *Vérification simple* : dans ce cas un message s'affiche contenant le résultat de la vérification des propriétés sélectionnées.

[2]Cet outil est disponible à l'adresse : http://www.redcad.org/software.html

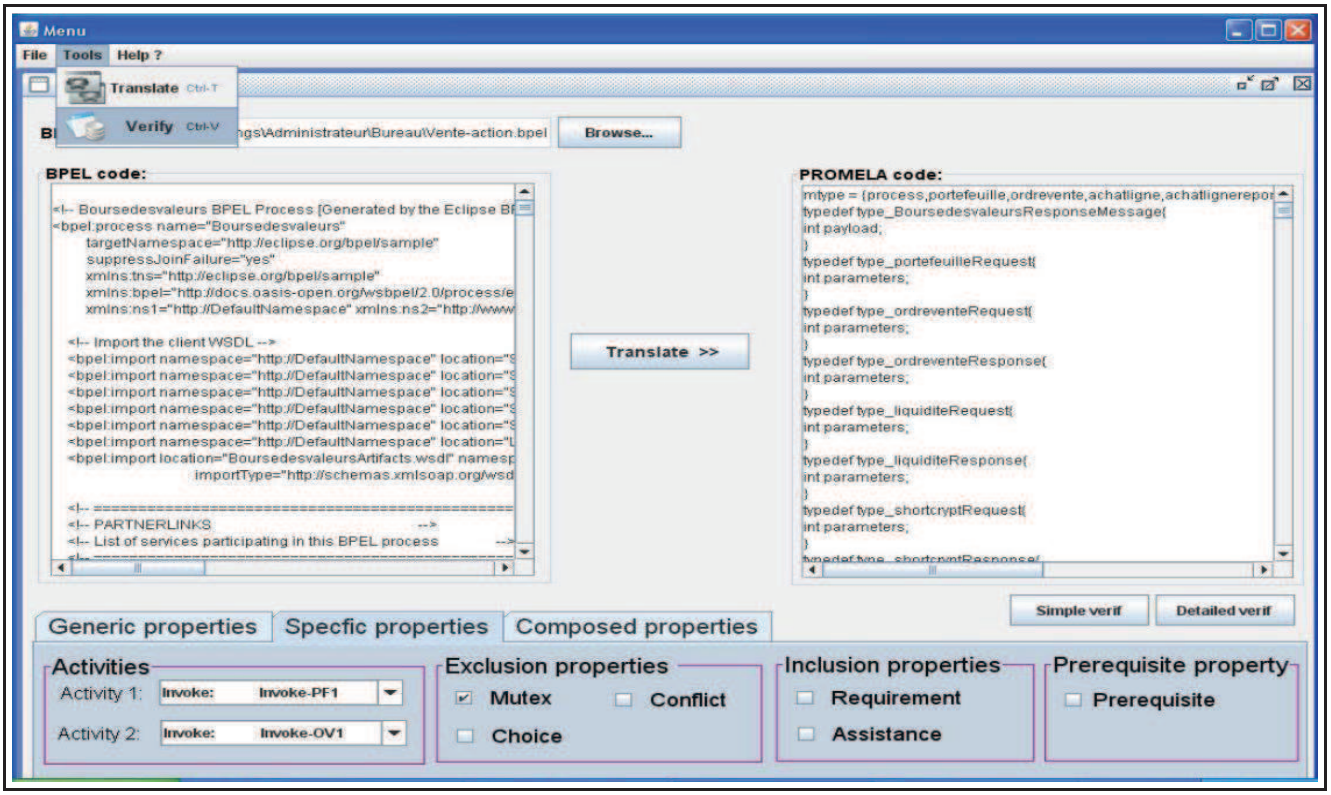

Figure 4: L'outil BpelVT.

- *Vérification complexe* : une fenêtre SPIN s'affiche pour chaque propriété sélectionnée contenant la propriété exprimée en LTL, le résultat de la vérification, le nombre d'états accessibles et l'espace mémoire utilisé.

6.1 Expérimentation

Comme illustration de notre approche de vérification de l'orchestration des services web, nous prenons comme étude de cas la bourse des valeurs (voir Figure 5) qui est un service web composé permettant aux clients de réaliser la vente en ligne des actions de bourse.

Invocation du service L'exécution du processus "Bourse des valeurs" au sein d'un marché financier est initialisée par la réception d'une demande de la part du client contenant le libellé de l'action, le nombre d'actions à vendre, le prix de vente unitaire ainsi que la période de vente. Dès qu'il reçoit le message, l'orchestrateur invoque le service web du portefeuille du client "SW Portefeuilles" pour vérifier le nombre d'actions à vendre. Si le nombre est supérieur à celui demandé, le processus envoie un message de vérification au client. Dans le cas inverse, le processus invoque le service web "SW Ordre de vente" afin d'enregistrer cet offre. Une fois le processus enregistre la transaction de vente, il sollicite le service web de gestion d'achat "SW Gestion d'achats", pour consulter les demandes d'achats des actions dans lesquelles le nombre d'actions à acheter est inférieur ou égale au nombre à vendre. Deux cas se présentent : si le service de gestion d'achats ne retourne aucune demande d'achats, le processus reste en attente d'une demande pendant la période de vente jusqu'à l'annulation de la transaction de vente pour informer le client qu'aucune demande d'achat n'est reçue pendant la période donnée. Sinon, le service web de gestion d'achat envoie ensuite au processus une demande d'achat.

Règlement Afin de procéder au payement, le processus crypte le montant de vente afin d'effectuer un règlement sécurisé. Si le montant de la facture est inférieur à 10 000 $, le processus invoque le service web de cryptage court "Cryptage court SW". Sinon, il invoque le service web de cryptage long "Cryptage long SW" pour crypter les données. Ensuite il sollicite le service web de liquidité "Liquidité SW" afin de régler la transaction en ligne. Lorsque le règlement est terminé, le processus invoque en parallèle deux services web: le premier est "Porte feuilles SW" pour débiter les actions du client, et le deuxième est "Ordre de vente SW" pour valider la transaction de vente. Finalement, le processus retourne les détails de vente au client.

6.2 Vérification des propriétés génériques

Dans cette section, trois propriétés sont considérées à savoir la vivacité, la sûreté et la disponibilité.

Vivacité Il s'agit de vérifier que le processus BPEL termine éventuellement son exécution avec sucées. La formule LTL est la suivante :

```
#define p activity-1==true
#define q activity-2==true

[] (p -> <>q)
```

activity-1 et activity-2 sont respectivement la première et la dernière activité du processus.

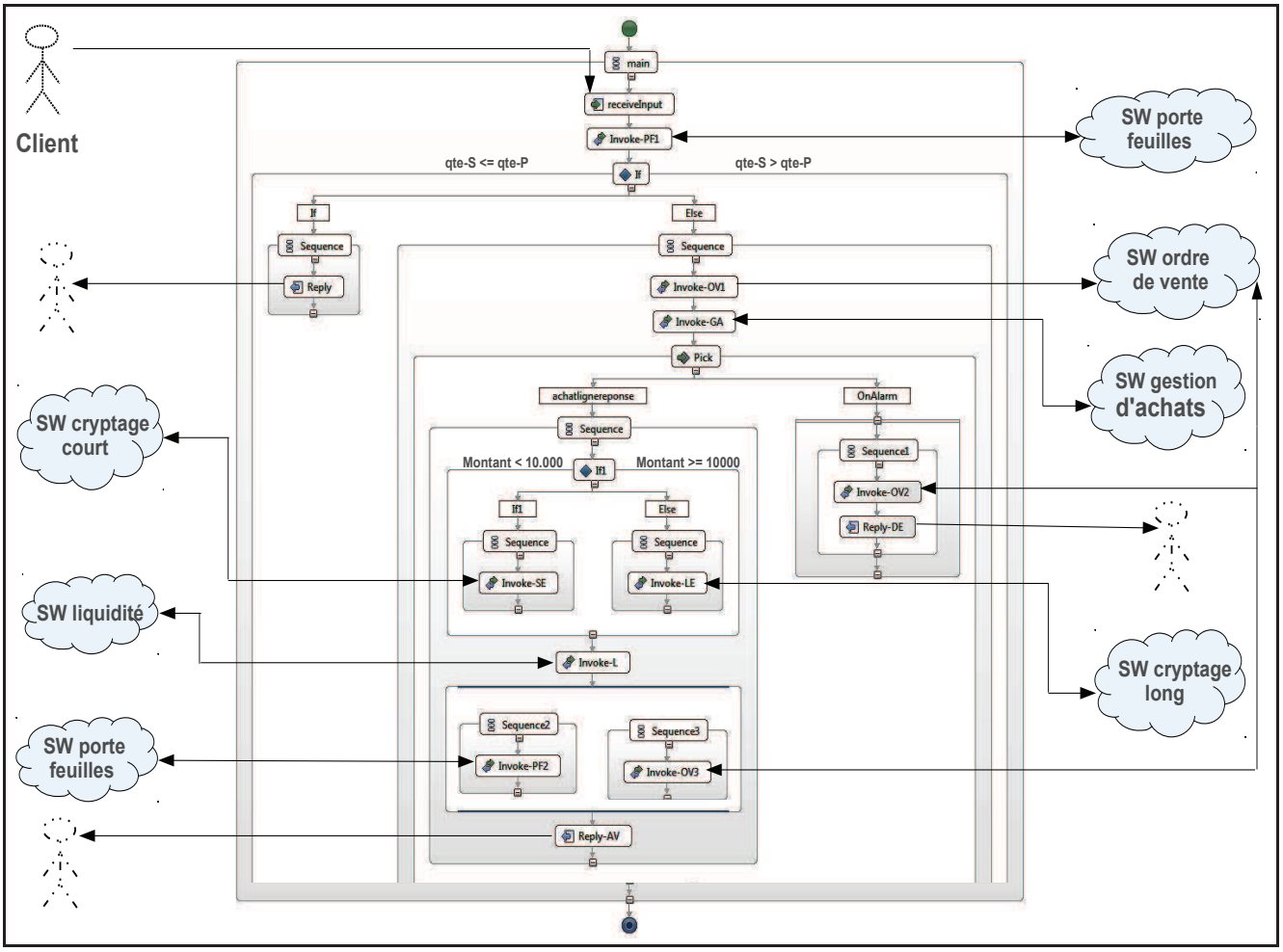

Figure 5: Processus BPEL pour la vente d'actions

Sûreté Chaque processus BPEL dispose toujours au moins d'une activité d'invocation de service web. La formule LTL est la suivante :

```
#define p invoke-activity==true

[] (p)
```

Disponibilité Toujours, chaque processus BPEL termine son exécution avec succès en garantissant qu'une réception d'un message est suivi, éventuellement, par une réponse. La formule LTL est la suivante :

```
#define p activity-1 == false
#define q activity-2 == true
#define x activity-3 == false
#define y activity-4 == true

[] ((p -> <>q) && (x -> <>y))
```

activity-1 et activity-2 sont respectivement la première et la dernière activité tandis que activity-3 et activity-4 sont respectivement invoke et reply.

6.3 Vérification des propriétés spécifiques

Afin que le processus BPEL relatif à notre étude de cas fonctionne à merveille, un ensemble d'exigences doivent être vérifiées. Ces exigences sont résumées dans les propriétés suivantes :

6.3.1 *Propriété d'exclusion : Mutex*

L'activité de confirmation d'achat ("Reply-AV") et celle d'annulation ("Reply-CL") ne doivent pas avoir lieu toutes les deux dans un processus BPEL. La formule LTL correspondante est la suivante :

```
#define p Reply-AV == true
#define q Reply-CL == true

[] !(p && q)
```

L'exécution de l'activité d'annulation signifie que l'action d'achat n'a pas eu lieu pour une raison ou une autre.

6.3.2 *Propriété d'inclusion : Requirement*

L'activité de confirmation d'achat (" invoke-OV3") ne doit avoir lieu que si l'action de payement (" invoke-L") est achevée avec succès. La formule LTL correspondante est la suivante:

```
#define p invoke-OV3 == true
#define q invoke-L == true

<>(p && q)) || <> (p)
```

6.3.3 *Propriété de précédence*

Une fois l'action d'achat (" invoke"-GA) est achevée avec succès, un message de confirmation (" Reply-DE") doit être soumis au consommateur. La formule LTL correspondante est la suivante :

```
#define p invoke–GA == true
#define q Reply–DE == true

!p W q
```

6.4 Composition de propriétés spécifiques

La composition de propriétés se réfère à la création d'une nouvelle propriété composée en rassemblant les propriétés spécifiques. En effet, la vérification d'une seule propriété est parfois insatisfaisante pour s'assurer de la bonne évolution globale du service web composé. Une composition de propriétés spécifiques est donc nécessaire pour construire de nouvelles propriétés avec de nouvelles fonctionnalités par leurs exécutions dans un ordre bien défini afin de satisfaire un besoin bien déterminé. Nous pouvons, par exemple, définir la propriété composée comme suit: une relation de disjonction entre la propriété de Précédence (entre les activités "invoke-GA" et "Reply-DE") et la conjonction de deux propriétés Mutex (entre les activités "Reply-AV" et "Reply-CL") et Assistance (entre les activités "Invoke-SE" et "Invoke-L").

```
Prerequisite || (Mutex && Assistance)
```

Cette propriété composée fournit d'autres scénarios consistants pour la vérification formelle en fournissant plus de flexibilité et d'expressivité en terme de combinaison de propriétés spécifiques à l'aide des opérateurs logiques.

6.5 Évaluation

Les résultats qui découlent de la vérification des propriétés génériques, spécifiques et composées sont récapitulés dans le tableau 1. Pour chaque propriété, nous donnons le résultat

Table 1: Résultat de la vérification

Propriété	Résultat	Etat accessibles	Mémoire (Mb)
Vivacité	Vrai	5238	7.176
Sûreté	Vrai	6892	8.836
Disponibilité	Vrai	10563	12.352
Mutex	Vrai	5238	7.176
Requirement	Vrai	5302	7.273
Précédence	Vrai	490	2.195
Propriété composée	Vrai	6571	8.641

de la vérification, le nombre d'états explorés et la taille de la mémoire utilisée pour la vérification.
Pour les propriétés génériques, SPIN explore entre 5000 et 10000 états accessibles en utilisant entre 7.000 et 12.000 Mb de mémoire. Pour vérifier des propriétés spécifiques, SPIN explore entre 400 et 6000 états accessibles en utilisant entre 2.000 et 8.000 Mb de mémoire. En passant d'un type de propriété à un autre, la taille de la mémoire ainsi que le nombre d'états explorés varient.

7. CONCLUSION ET PERSPECTIVES

Dans cet article, nous avons présenté notre approche pour la validation d'une orchestration de services web par la vérification des propriétés non-fonctionnelles. Comparée aux autres approches, l'originalité de notre proposition est de permettre la prise en compte de deux types de propriétés non-fonctionnelles au moment de la validation à savoir les propriétés génériques et les propriétés spécifiques. Les propriétés génériques peuvent être vérifiées sur les services qui participent à une orchestration. Les propriétés spécifiques sont les relations d'interdépendance entre les activités d'un processus BPEL. Une composition de nouvelles propriétés est possible à partir des propriétés spécifiques. La vérification de ces propriétés est assurée à l'aide du model-chercker SPIN pour cela nous étions amener à développer un outil de transformation du BPEL vers Promela, le langage de modélisation de SPIN. C'est un outil à base de grammaire et développé à l'aide de l'outil ANTLR.

Ce travail ouvre plusieurs perspectives. Nous envisageons, dans un premier temps, d'intégrer le retour en arrière (backtracking) du code Promela au modèle BPEL après une étape de raffinement ce qui permet d'aider à la localisation des erreurs en cas de violation d'une (ou plusieurs) propriété(s). Ensuite, nous étendons notre approche pour qu'elle supporte d'autres langages de spécification et ne se limite pas à BPEL comme langage d'orchestration et à SPIN/Promela comme model-checker. A moyen terme, nous souhaitons étudier l'intérêt de notre approche pour permettre la validation de la chorégraphie. Nous envisageons également d'étendre les propriétés non-fonctionnelles en ajoutant d'autres propriétés spécifiques à ce type de composition. Enfin, il serait nécessaire d'étudier la composition de services dans le could computing pour voir la possibilité de bénéficier de notre approche.

8. REFERENCES

[1] F. Abouzaid and J. Mullins. Model-checking web services orchestrations using bp-calculus. *Electronic Notes in Theoretical Computer Science*, 255:3–21, 2009.

[2] I. Ait-Sadoune and Y. Ait-Ameur. From bpel to event-b. In *International Workshop on Integration of Model-based Methods and Tools IM FMT'09 at IFM'09 Conference, Düsseldorf Germany*, Fevruary 2009.

[3] S. Chardigny. *Extraction d'une architecture logicielle à base de composants depuis un système orienté objet : Une approche par exploration.* PhD thesis, Laboratoire d'Informatique de Nantes Atlantique, Université de Nantes, 2009.

[4] A. Dekdouk and T. El-Basuny. On v&v of web service oriented architectures. In *Proceedings of the Second ACM/SIGART/SIGMIS International Conference on Intelligent Computing and Information Systems, Ain Shams University, Egypt*, pages 536–542, March 2005.

[5] X. Fu, T. Bultan, and J. Su. Wsat: A tool for formal analysis of web services. In *Proceedings of 16th Int. Conf. on Computer Aided Verification (CAV*, pages 510–514. Springer, 2004.

[6] A. Hegedüs, I. Ráth, and D. Varró. From bpel to sal and back: a tool demo on back-annotation with viatra2. *8th International Conference on Software Engineering and Formal Methods*, 2010.

[7] G. Holzmann. The spin model checker: Primer and reference manual. Addison-Wesley, Boston, Massachusetts, 2003.

[8] M. Juric. A hands-on introduction to bpel. Oracle Technology Network, 2010.

[9] R. Kazhamiakin, P. K. Pandya, and M. Pistore. Representation, verification, and computation of timed properties in web service compositions. *ICWS*, 2006.

[10] M. Kovacs and L. Gonczy. Simulation and formal analysis of workflow models. *Electronic Notes in Theoretical Computer Science (ENTCS)*, 211, 2008.

[11] S. Nakajima. Model-checking behavioral specification of bpel applications. In *Proceedings of the International Workshop on Web Languages and Formal Methods*, volume 151, pages 89–105. Electronic Notes in Theoretical Computer Science, July 2005.

[12] Oasis. Web service business process execution language version 2.0 specification, oasis standard, April 2007.

[13] Y. Qian, Y. Xu, Z. Wang, G. Pu, H. Zhu, and C. Cai. Tool support for bpel verification in activebpel engine. *Software Engineering Conference*, pages 90–100, 2007.

[14] T. Quyet, Q. Thi, and D. Hoang. A method of verifying web service composition. In *Proceedings of the 2010 Symposium on Information and Communication Technology*, SoICT '10, pages 155–162, New York, NY, USA, 2010. ACM.

[15] A. Salah, G. Tremblay, and A. Chami. Behavioral interface conformance checking for ws-bpel processes. In *International MCETECH Conference on e-Technologies*, pages 253–257, 23-25 Jan 2008.