Gestion Intégrée du Changement des Modèles de Processus Métier

Mourad Bouneffa, Adeel Ahmad, Henri Basson

Gestion Intégrée du Changement des Modèles de Processus Métier

Mourad Bouneffa, Adeel Ahmad, Henri Basson

Université du Littoral Côte d'Opale
Laboratoire d'Informatique Signal et Image de la Côte d'Opale
50, rue Ferdinand Buisson - BP 719, 62228 CALAIS CEDEX*, FRANCE*
bouneffa,ahmad,basson@lisic.univ-littoral.fr

RÉSUMÉ. *Ces dernières années, les modèles de processus métier ou BPM, ont été utilisés comme des entités servant dans la spécification des différents processus d'une organisation. Des ateliers existent, permettant même de générer des systèmes informatiques exécutables sur des plates-formes distribuées à partir de modèles de processus métier. Ceci a été facilité par la généralisation de la notion d'architectures orientées service ou SOA. Nous proposons une approche à base de graphes et de systèmes de réécriture de graphe pour à la fois modéliser et simuler la propagation de l'impact du changement des différents constituants d'une application basée sur la mise en oeuvre des modèles de processus métier. Nous adoptons également l'utilisation des ontologies dans le but de représenter la sémantique des relations reliant les différents constituants de ce type d'applications. Cette connaissance sémantique sera alors utilisée par un algorithme permettant de générer des règles de réécriture de graphes implémentant la propagation de l'impact du changement de ce type d'applications.*

ABSTRACT. *For the past few years, the Business Process Models or BPMs, have been largely used to specify the different processes of an organisation. The existing frameworks, may also generate executable information systems, deployed on distributed platforms, from the instantiation of business process models. This has been facilitated by the generalization of the so called Service-Oriented Architecture (SOA). We propose a graph-based approach, along with the graph re-writing system, for the modeling and simulation of the change impact propagation among the different constituents of the instantiated business process models. We use ontologies for the semantic representation of the relations linking the different constituents of such applications. This semantic knowledge is then used by an algorithm that may generate the graph re-writing rules to incorporate the change impact propagation in such applications.*

MOTS-CLÉS : *Modèles de Processus Métier (BPM), Propagation de l'impact du changement, Systèmes de réécriture de graphe, Ontologies.*

KEYWORDS: *Business Process Model (BPM), Change impact propagation, Graph re-writing systems, Ontology.*

---

1. Introduction

Les modèles de processus métier (Business Process Models ou BPMs) ont servi, dans un premier temps, comme formalisme dans le cadre du management des organisations et plus particulièrement dans le cadre de la réingénierie des processus métier (Business Process Reengeneering) (Reijers *et al.*, 2016). La dernière décennie a vu l'extension de l'utilisation de ces formalismes au domaine du développement et du déploiement des applications collaboratives souvent distribuées. Ainsi les BPMs sont passés au stade d'abstractions de haut niveau ayant un lien avec des composants informatiques exécutables les implémentant et les déployant sur des plates-formes souvent distribuées. Ceci a fortement été encouragé par l'émergence des architectures dites orientées service (SOA). Ainsi, on assiste à l'emergence d'approches et d'outils pour le développement de logiciels d'entreprise distribués dans lesquels les BPMs sont transformés en entités exécutables. Dans ces approches, les BPMs sont spécifiés à l'aide de notations standards telles que BPMN (Business Process Model Notation) [1] et XPDL [2]. Ils sont, par la suite, transformés en programmes exécutables déployés sur des plates-formes distribuées en forme d'applications multi-tiers (Java J2EE, .NET, etc.). Ces programmes exécutables sont également souvent construits comme des macro-programmes implémentant ce qui est communément appelé *the programming in the large*. Ces programmes contiennent des invocations de services web (Gottschalk *et al.*, 2002) fournis par diverses applications déployées à l'intérieur et parfois à l'extérieur des frontières du système d'information concerné. BPEL (Business Process Execution Language) (Juric, 2006) est l'un des langages les plus connus en matière de *programming in the large.* Il permet la composition ou plutôt l'orchestration de services web dans le but de réaliser une fonctionnalité donnée pouvant correspondre aux actions formant une activité d'un BPM. Ce type d'approches peut contribuer à l'élimination du gap existant entre les BPMs et leur implémentation ou déploiement en termes de composants IT (Information Technology). Il devient alors possible d'imaginer une gestion intégrée des changements d'un BPM en répercutant les effets de ces changements sur les composants IT correspondant et inversement.

La notion de changement est intrinsèquement liée aux applications basées sur le BPM. Comme nous le montrerons dans la section 2, le cycle de vie de ces applications est une succession d'itérations ayant pour but l'amélioration de critères de qualité et de performance formant le tableau de bord des décideurs aussi bien au niveau fonctionnel que technique. Cette question est donc centrale et a conduit à de nombreux travaux reposant généralement sur la notion de patterns ou patrons de changement des BPMs (Kim *et al.*, 2007). Dans ce papier, nous nous intéressons à un problème particulier qui est l'analyse et à la propagation de l'impact du changement de ce type d'applications. Nous privilégions l'utilisation des règles de réécriture de graphes dans le but de pouvoir automatiser leur génération et donc d'automatiser le processus d'identification de l'impact du changement. Les graphes et plus spécifiquement les graphes

1. http://www.omg.org/spec/BPMN/2.0
2. http://www.xpdl.org

typés et attribués ont, depuis longtemps, servi comme structures de prédilection dans le cadre de la gestion de l'évolution du logiciel. Cela est dû à la nature du logiciel qui est constitué d'une grande collection d'artefacts de différents types (représentables par des nœuds) reliés par divers types de relations. Dans ce cadre, les systèmes de réécriture de graphes ont également été expérimentés depuis de nombreuses années aussi bien dans la problématique de propagation de l'impact du changement (Rajlich, Gosavi, 2004 ; Maweed *et al.*, 2005) qu'à celui du refactoring de modèles (Folli, Mens, 2007 ; Mens, 2005), etc. Ces systèmes fournissent un outil de raisonnement, de spécification et de mise en oeuvre visuelle (donc simplifiée) des différentes tâches rentrant dans le cadre de l'évolution du logiciel et nécessitant une prise en compte des différents liens inter-artefacts. Ils permettent, entre autres, d'éviter le phénomène *d'impedence mismatch* qui est généré par une utilisation de structures hétérogènes pour la manipulation des artefacts logiciels. C'est le cas notamment quand ces artefacts sont stockés dans des bases de données relationnelles, transformés après en faits pour être manipulés par un système d'inférence et en graphes pour être visualisés. Notre choix des systèmes de réécriture de graphes pour la gestion de l'évolution d'applications reposant sur les processus métier s'inscrit dans ce cadre. Notre but est également d'assurer une certaine inter-opérabilité entre les travaux que nous avons déjà menés dans le cadre de l'évolution du logiciel et ceux concernant plus spécifiquement les applications basées sur les processus métier. Nous avons donc expérimenté la faisabilité de l'utilisation des systèmes de réécriture de graphes dans le cadre de l'analyse *a priori* et la propagation de l'impact du changement des applications orientées BPM (Bouneffa, Ahmad, 2013b ; 2013a). Dans le but d'une automatisation de la génération des règles de réécriture de graphes et donc d'une automatisation de la génération du processus d'analyse et de propagation du changement, nous avons été confrontés à une limitation des systèmes de réécriture de graphes. En effet, pour une telle automatisation il était nécessaire d'expliciter des connaissances d'ordre sémantique sur les artefacts et plus particulièrement sur les relations inter-artefacts. Pour cela, nous avons introduit, dans ce papier, une ontologie qui explicite certaines connaissances utiles telles que la manière dont une relation conduit l'impact d'un changement, etc. Bien que la notion d'impact soit étroitement liée à la violation de propriétés de cohérence des processus métier, dans ce papier, nous n'abordons pas de façon approfondie la problématique de vérification des propriétés des processus que nous avons déjà traitée dans (Kherbouche *et al.*, 2013).

La suite du papier est organisée comme suit : la section 2 décrit les principaux constituants des applications centrées BPM. La section 3 décrit le méta modèle associé aux BPMs et BPEL et qui est basé sur l'utilisation de la notion de graphes attribués et typés enrichis par des connaissances sémantiques explicités par une ontologie. En effet, notre approche étant basée sur l'utilisation des systèmes de réécriture de graphes pour la mise en oeuvre et l'analyse de l'impact des changements des applications centrées BPM, il était évident que le méta modèle emprunte le vocabulaire issus de ces systèmes et repose sur l'utilisation des types de graphes. La section 4 décrit les opérations de changement, leur exécution et le processus d'identification et de propagation de l'impact de ces changements à l'aide des règles de réécriture de

graphes. Elle décrit également l'utilisation des ontologies dans le cadre de la mise en oeuvre d'un algorithme automatisant la génération des règles de réécriture de graphes pour la propagation de l'impact du changement. La section 5 illustre l'intégration de notre approche dans le cadre d'un atelier que nous développons depuis de nombreuses années et qui est dédié à la gestion de l'impact des applications distribués. La dernière section conclut le papier en faisant le bilan des résultats obtenus et en y esquissant ses perspectives.

2. Les applications centrées sur le BPM

Le cycle de vie des applications développées et déployées comme une instantiation exécutable de BPMs est une succession d'itérations formées chacune de quatre principales phases : *la modélisation, le déploiement, l'exécution et le monitoring* et *l'amélioration*. La phase de modélisation du BPM en termes de tâches ou activités est un préalable à l'implémentation d'un processus. Elle explicite, entre autres, l'ordre d'exécution des tâches, les acteurs humains et éventuellement les données nécessaires à l'accomplissement de ces tâches. Plusieurs modèles ou notations ont été introduits pour la représentation des processus métier. Les premières notations étaient destinées à produire des modélisations utilisables dans le cadre des projets de réingénierie des processus métier (BPR : Business Process Reingineering). Dans ce papier, nous considérons particulièrement les modèles utilisés dans le cadre de l'automatisation des processus métier. En d'autres termes, nous nous plaçons dans le cadre de l'utilisation des BPMs comme une abstraction de haut niveau destinée à être transformée en une application déployable sur une architecture informatique distribuée. Nous avons choisi le langage qui semble être reconnu comme un standard en la matière, en d'autres termes BPMN (Business Process Model Notation). FIGURE 1 montre un exemple de processus métier exprimé selon la notation BPMN et représentant une partie d'une chaîne de vente. La figure met en évidence deux principaux acteurs de cette chaîne et qui sont le *Client* et le *Processus de gestion de commande*. Au démarrage, un événement de début *(start event)* associé au client permet de considérer l'activité *Place Order* comme la première activité démarrant le processus. Cette activité génère une commande qui est transmise par courrier ou message à l'activité *Check Availability* qui vérifie la disponibilité du produit commandé. Cette dernière est reliée à un nœud de type *GateWay* ou passerelle qui selon le résultat de l'activité réalise un branchement vers l'activité *Check Payment* (dans le cas où le produit commandé est disponible) ou l'activité *Cancel Order* dans le cas contraire. L'activité *Check Payment* est elle même reliée en sortie à un nœud GateWay qui aiguille respectivement vers les activités *Confirm Order* ou *Cancel Order* selon que le paiement soit validé ou pas. Ces deux dernières activités sont reliées chacune à un événement de type *Message* mais également terminal (représenté en gras).

Le développement et le déploiement des applications centrées BPM sont deux activités séparées pouvant être effectuées de façon manuelle, semi-automatique ou automatique. En principe ces deux activités peuvent être comme des opérations classiques de génération et de déploiement de code où le BPM joue le rôle de conception dé-

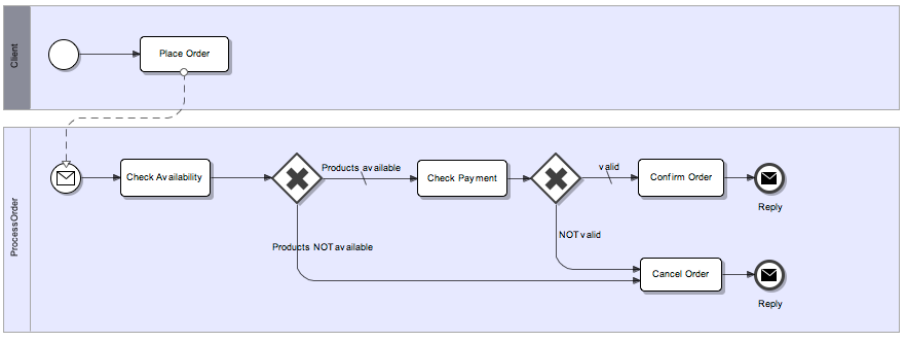

FIGURE *1. Exemple de processus métier en notation BPMN*

taillée et où le code est matérialisé par une application généralement déployée en forme d'application Web. Il existe, depuis quelques années, des outils automatisant le déploiement de ce type d'applications. C'est le cas notamment de Bonita [3], Intalio [4], BizAgi [5] et Barium Live! [6], etc. Ces outils permettent, à partir d'un BPM, la génération d'une application web de façon automatique et quasi transparente. Ces applications sont constituées de pages web dynamiques pouvant être des pages JSP, des ASP.NET, etc. Indépendamment de la question de l'automatisation de la génération de l'application implémentant un BPM, il existe des langages ayant pour but la mise en oeuvre sous la forme d'un programme *in the large*, d'orchestrations de services concourant à la réalisation effective de ce qui est modélisé dans le BPM. C'est le cas notamment du langage BPEL (Business Proces Execution Language) (Juric, 2006). Dans (Bouneffa, Ahmad, 2013a), nous décrivons une implantation de l'exemple de FIGURE 1 à l'aide d'une orchestration de services web BPEL contenant principalement des invocations de services web qui constituent les activités de base et des structures représentant des séquences d'invocations, ou des strutures conditionnelles et itéartives, etc.

L'exécution d'un BPM permet de recueillir certaines informations concernant les temps d'exécution, la consommation de ressources, etc. Ces données présentent un intérêt dans le cadre de l'étude des performances et en général de la qualité des processus mis en oeuvre. D'un autres côté, les experts métier mettent en place un certain nombre d'indicateurs de performance ou KPI (Key Performance Indicators). Cela permet de renseigner sur, par exemple, le temps moyen d'attente avant l'accomplissement d'une tâche donnée, etc. Certaines données concernant les KPI peuvent être obtenues par un

3. Bonita Open Solution url: http://www.bonitasoft.com/
4. Intalio|BPMS : http://www.intalio.com/
5. Bizagi BPM Suite : http://www.bizagi.com/
6. Barium Live! : http://www.bariumlive.com/

profiling de l'exécution de l'application (Ahmad *et al.*, 2009). D'autres informations ne peuvent être fournies que manuellement par un expert métier.

L'amélioration du BPM est un terme générique se référant à la notion d'évolution des processus du BPM. En réalité, l'amélioration attendue est le résultat de changements pouvant affecter chaque constituant d'une application centrée BPM et qui peut aussi bien être un élément du modèle de processus qu'un service web ou tout autre artefact logiciel rentrant dans la composition de l'application. Le but de tels changements peut être la correction d'erreurs ou d'anomalies constatées, un alignement avec des changements affectant le processus métier (changement de réglementation, réingéniérie du processus, etc.), etc. Dans la littérature, le changement est vu du point de vue du processus métier et concerne très rarement la composante logicielle implémentant ces processus.

3. Un méta modèle de BPM et de BPEL à base de graphes

Nous proposons un méta modèle représentant les concepts impliqués dans la définition des BPMs en utilisant la notion de graphes attribués et typés et les ontologies. L'utilisation des graphes typés et attribués est une conséquence de l'adoption dans tous nos travaux de la notion de graphes pour représenter tous les artefacts logiciels et leurs différents liens sémantiques et cela quelle que soit la nature de ces artefacts (programmes, schémas de bases de données, BPMs, etc.). Cela est également rendu nécessaire par le fait que nous préconisons l'utilisation des systèmes de réécriture de graphes pour l'implémentation des règles régissant l'identification et la propagation de l'impact du changement. L'utilisation des ontologies est arrivée comme une conséquence d'un besoin en matière d'explicitation de connaissances sémantiques supplémentaires et nécessaires, notamment, à la mise en œuvre d'un processus de génération automatique de règles de propagation de l'impact. Dans ce papier, nous nous sommes cantonnés à des connaissances concernant la conductivité de l'impact par les différentes relations reliant les artefacts d'une application basée BPM en utilisant le langage OWL [7] et la plate-forme *Protégé* [8]. Nous avons donc profiter du fait que OWL permette une hiérarchisation des relations par le lien ISA (FIGURE 2) pour définir 5 sous-types de relations selon le sens dans lequel ces relations conduisent l'impact à savoir : de la source ou domaine vers la destination ou co-domaine pour les relations *ForwardImpact* et inversement pour les relations *InverseImpact*. Nous avons également considéré le fait que l'impact soit certain ou conditionnel et également le fait qu'il n'y ait pas d'impact. Ces types de relations ne sont pas tous disjoints. Cela veut dire qu'une relation peut être de type *ForwardImpact* et *InverseImpact*. Une relation peut également être de type *ForwardImpact* et *CertainImpact* comme elle peut être *InverseImpact* et *ConditionalImpact*. Par exemple, une relation qu'on appellera *ImplementedBy* entre un BPM et le BPEL l'implémentant est aussi bien *ForwardImpact*

7. https://www.w3.org/2004/OWL
8. http://protege.stanford.edu

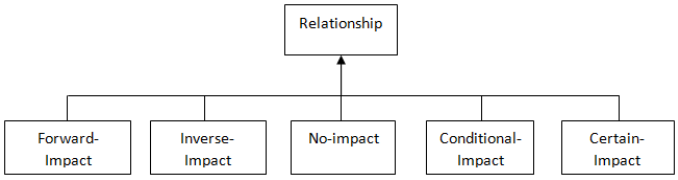

FIGURE 2. *Hiérarchie des classes de relations selon la propagation de l'impact*

que *InverseImpact*. En effet, le changement affectant un BPM peut affecter le BPEL l'implémentant. Il en est de même du changement affectant un BPEL par rapport au BPM correspondant. Cependant cette relation est *ConditionalImpact* puisque le changement d'un BPM ne conduit pas forcément à un changement affectant le BPEL correspondant et inversement.

Le méta modèle à base de graphes typés et attribués a été formalisé dans le cadre de la plate-forme AGG [9] dédiée à la transformation ou réécriture de graphes. Cette méta-modélisation est schématisée par FIGURE 3 qui présente les principaux concepts introduits par la notation BPMN. Le concept de *Processsus* représente les processus métier qui peuvent contenir des objets de flots. Ces derniers peuvent être des *tâches* ou activités, des *sous-processus* ou macro tâches raffinées par des processus, des *passerelles* (gateways), des *événements* ou des artefacts utilisés par les tâches tels que les documents, les données, etc. Un processus est associé à un acteur qu'on appellera *owner* représentant un ou plusieurs utilisateurs l'ayant défini et ayant le droit de le changer. Un processus est implémenté par une application pouvant héberger une ou plusieurs instances d'exécution de ce processus. Chaque instance fait intervenir des acteurs qui interagissent avec les différentes tâches exécutées durant le cycle de vie de l'instance. Le formalisme à base de graphes typés et attribués peut être vu comme un moyen permettant, entre autres, la vérification de la consistance des artefacts d'un BPM qui sont eux même formalisés par des graphes. Parmi les relations intéressantes notamment pour l'analyse de l'impact du changement, on considèrera les relations *input* et *output* associées à des objets de flots. En effet la relation *input* spécifie les objets de type *Tâche*, *Sous Processus* ou *Passerelle* qui constituent les entrées ou objets en amont d'un objet donné alors que la relation *output* spécifie les objets en sortie d'un objet donné. Ainsi, lors du changement d'un objet, on pourra, par le biais de la relation *output* , retrouver les objets en aval au niveau du flot de contrôle et par *input*, identifier les objets en amont de l'objet qui vient d'être modifié.

Le méta-modèle de FIGURE 3 représente la couche processus. Dans notre travail d'analyse de l'impact du changement nous avons également comme objectif la propagation de l'impact au niveau des artefacts d'un processus exprimé en BPMN vers

9. http://user.cs.tu-berlin.de/~gragra/agg/

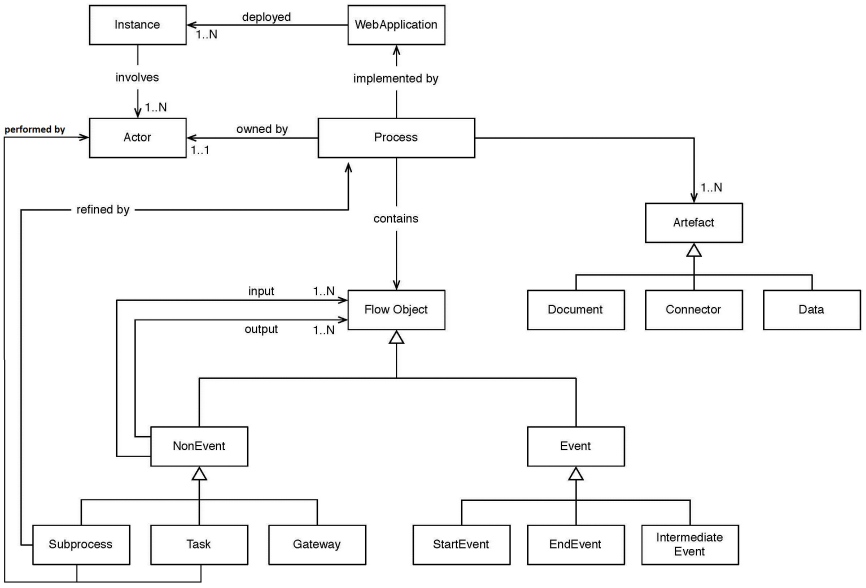

FIGURE *3. Métamodélisation des BPMs à base de graphes typés et attribués*

ceux de la couche service constitués d'orchestrations de services web. Le méta modèle de BPEL représente un processus BPEL comme étant composé d'activités qui peuvent être simples (*Basic Activity*) ou composites (*Structured Activity*). Une activité simple correspond à une invocation d'un service web (*Invoke*), l'attente de la réception d'un message (*Receive*), la sortie du processus (*Exit*) ou une activité vide (*Empty*) consistant à ne rien faire. Une activité composite consiste généralement en une orchestration d'activités simples ou composites et cela par le biais de structure telle que *Sequence* qui consiste à ordonnancer l'exécution de plusieurs activités en séquentiel, *Flow* qui correspond à une exécution parallèle de plusieurs activités, *Switch* qui est une exécution conditionnelle, etc. Dans ce papier, nous ne traitons pas du processus de transformation d'un BPMN en BPEL, mais nous partons du fait que des mappings existent et nous les matérialisons par deux relations appelées *mappedTo* et *ImplementedBy* et que nous exploitons dans le processus d'analyse et de propagation de l'impact qu'on appelle vertical et qui se propage d'une couche plus abstraite d'une application vers sa concrétisation ou implémentation et inversement. Ces relations s'expriment au niveau des graphes représentant les processus par un arc de type *ImplementedBy* dont la source est un artefact BPMN et la destination un artefact de BPEL et un lien inverse de type *mappedTo*. Un mapping peut exister entre un processus BPMN et un processus BPEL, entre une tâche BPMN et une activité BPEL, etc. En effet, plusieurs stratégies ont été proposées dans la littérature ayant conduit à différentes approches et algorithmes de transformation de BPMN en BPEL et *vice versa* (Ouvans *et al.*, 2006 ;

Mazanek, Hanus, 2011 ; Doux *et al.*, 2009). Ce qui est notable dans ces différents travaux est que ces transformations ne sont pas forcément évidentes et cela est du au fait que le BPMN est orienté graphe alors que le BPEL est orienté bloc et structuration de blocs.

4. Exécution des opérations de changement et analyse de l'impact du changement par un système de réécriture de graphes

La méta-modélisation de la section précédente nous permet de dresser une taxinomie des opérations de changement en considérant le critère de la granularité des constituants à modifier et la couche de l'application à laquelle ce constituant appartient. Nous avons ainsi considéré la couche BPM et la couche BPEL ou service et nous avons considéré chaque type de nœuds et d'arcs. Nous déterminons ainsi des opérations atomiques du changement telles que la création et la suppression d'une tâche d'un BPM, d'une activité d'un BPEL, etc. Nous définissons également des opérations composites telles que la fusion de deux activités, le remplacement d'une activité de base par une activité composite, etc. Une description plus détaillée de ces opérations a été effectuée dans (Bouneffa, Ahmad, 2013a). Bien entendu, nous sommes conscient qu'il serait plus judicieux de reprendre les taxinomies déjà existantes et particulièrement celles découlant de la définition de patterns de changement et du refactoring des processus métier. Cependant la limitation de la taille du papier nous contraint à réduire voir simplifier la taxinomie. L'objectif principal du papier étant plus la présentation du mécanisme de propagation de l'impact.

L'exécution des opérations de changement est formalisée par des règles de réécriture de graphes. Une règle de réécriture de graphe est en réalité une règle de production $LHS \longrightarrow RHS$ dans laquelle les deux termes *LHS (Left Hand Side)* et *RHS (Right Hand Side)* sont deux graphes. L'application d'une telle règle sur un graphe G dit graphe *Hôte*, consistera à remplacer les sous-graphes de G correspondant à *LHS* par *RHS*. En d'autres termes, cette règle consiste à trouver un morphisme m permettant d'établir une correspondance entre *LHS* et une partie g de G ($m(g)=LHS$) puis remplacer g par *RHS* dans G. *LHS* est appelée la pré condition de la règle et *RHS* la post condition. Il existe également des expressions qu'on appellera *NAC* ou pré conditions négatives qui stipulent que la règle ne peut être appliquée s'il existe dans le graphe hôte G, un sous graphe qui correspond à *NAC*. Ces règles sont souvent exprimées de façon visuelle. C'est notamment le cas du système *AGG* que nous utilisons. Ainsi, FIGURE 4 montre une règle correspondant à l'insertion d'une nouvelle tâche. Cette règle énonce comme pré condition à la création d'une tâche, l'existence d'un nœud de type *Process*. En d'autres termes, une tâche ne peut être créée que si elle est reliée à un processus existant. La pré condition négative énonce le fait que le processus apparaissant dans la pré condition ne doit pas contenir une tâche du même nom. Les numéros préfixant les deux nœuds de types processus, celui apparaissant dans la *LHS* et celui apparaissant dans la *NAC*, permettent de faire la liaison entre les deux processus et de stipuler qu'ils désignent le même processus.

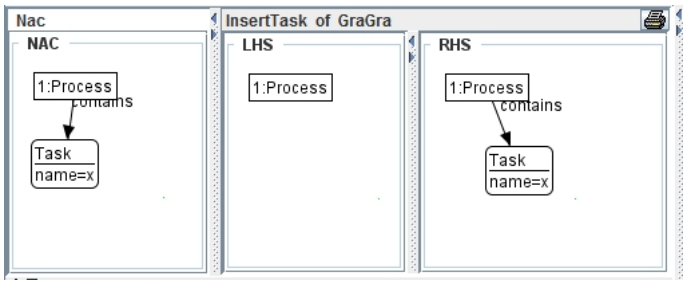

FIGURE *4. Règle de réécriture de graphe implémentant la création d'une tâche*

4.1. L'analyse de l'impact du changement

Dans le but d'analyser l'impact d'une opération de changement, nous considérons deux principales fonctionnalités : la génération et la propagation de l'impact. La génération de l'impact consiste à détecter les effets de bord d'un changement alors que la propagation de cet impact consiste à déterminer les artefacts qui sont directement ou indirectement liés aux artefacts qui viennent d'être changés et pour lesquels un impact a été généré. Pour ce faire, nous nous basons sur l'utilisation des règles de réécriture de graphes.

4.1.1. Génération de l'impact du changement

Comme nous l'avons déjà montré, chaque opération de changement est matérialisée par une règle de réécriture de graphes constituée de trois éléments : une pré condition (*LHS*), une post condition (*RHS*) et une pré condition négative (*NAC*). En général, la pré condition négative sert à éviter l'exécution d'une opération qui peut provoquer un effet de bord indésirable ou impact. En effet, une *NAC* empêche l'exécution d'une opération si cette exécution provoque la violation de différentes règles de cohérence matérialisée par les *NAC*. C'est le cas notamment dans FIGURE 4 où la *NAC* empêche la création d'une tâche portant le même nom qu'une tâche déjà existante. Dans notre approche, la génération de l'impact consiste à définir des opérations de réécriture ne contenant pas de *NAC*. En d'autres termes, on permet l'exécution de changements pouvant provoquer des incohérences. Cependant, nous ajoutons aux postconditions (*RHS*), un nœud représentant l'impact et des arcs le reliant aux nœuds qu'il affecte. En plus, ce nœud est attribué par une chaîne de caractères explicitant la cause de l'impact. FIGURE 5 représente l'opération de suppression d'une tâche reliée à deux autres tâches par un arc de type *outputs* (ces deux tâches se situent à la sortie de la tâche qui vient d'être supprimée). Dans ce cas, la pré condition négative aurait interdit cette suppression. Au lieu de cette interdiction, nous avons défini une *RHS* qui réalise la suppression et contient un nœud de type *Impact* qui est relié aux nœuds représentant les tâches affectées par cette suppression. Ce nœud contient un attribut appelé *explanation* dont la valeur est une chaîne de caractères représentant l'explication de l'ori-

gine de l'impact. Une opération de suppression peut conduire à l'écriture de plusieurs règles de générations d'impact. En d'autres termes, il peut y avoir plusieurs noeuds de type *Impact* correspondant chacun à la violation d'une règle de cohérence du BPM ou du BPEL. Cela peut être fastidieux et surtout répétitif pour le gestionnaire du changement à qui incombe l'écriture de ces règles. Nous avons donc proposé (voir section 4.2), un algorithme permettant d'automatiser la génération des règles de génération et de propagation de l'impact du changement.

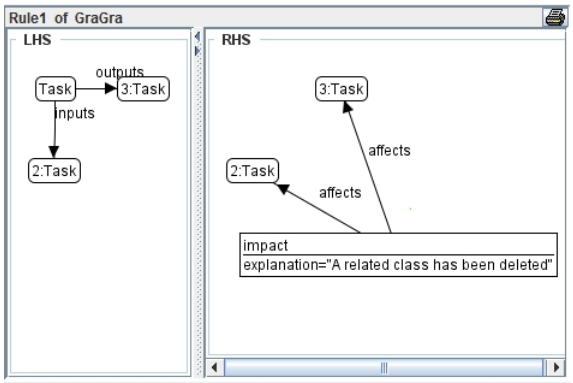

FIGURE *5. Génération d'un impact suite à la suppression d'une tâche*

4.1.2. Propagation de l'impact du changement

La propagation de l'impact d'un changement est un processus permettant d'identifier tous les nœuds indirectement affectés par l'impact de ce changement. Cette propagation est conduite par les différents liens ou relations reliant les nœuds en question. En effet, selon le type de relation et de changement, il est possible que l'impact soit propagé dans un sens ou un autre (de l'origine vers la destination de la relation ou de la destination vers l'origine) ou pas du tout. FIGURE 6 montre la propagation de l'impact affectant une tâche sur l'auteur exécutant cette tâche et cela par le biais de la relation *performs* qui relie un auteur aux tâches qu'il exécute. Nous avons défini un type d'arc appelé *inducedBy* matérialisant une relation de causalité entre impacts et permettant de visualiser la propagation ou effet de vague d'un impact.

Nous considérons deux principaux types de propagation d'impact du changement :

– La propagation horizontale consistant à propager l'impact à travers une relation reliant deux entités appartenant à la même phase du cycle de vie de l'application. En ce qui nous concerne cela revient à propager l'impact entre des constituants appartenant à la phase de modélisation du BPM ou à ceux appartenant à des artefacts appartenant à la couche service (BPEL).

– La propagation verticale consistant à propager l'impact d'un artefact appartenant à une phase du cycle de vie vers un autre artefact appartenant à une phase située en amont ou en aval. C'est le cas notamment de la propagation de l'impact entre une tâche du BPM et l'activité ou les activités BPEL qui participent à son implémentation (utilisation de la relation *ImplementedBy*) et inversement (utilisation de la relation *mappedTo*).

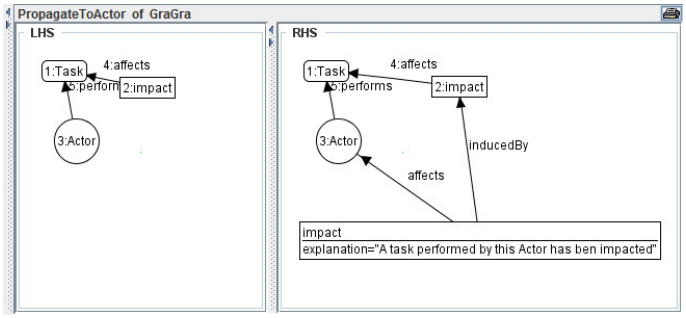

FIGURE *6. Une règle de propagation de l'impact du changement*

Nous considérons également une deuxième classification de la propagation de l'impact selon un autre critère qui est la stratégie d'exécution du processus. Ainsi, nous considérons les trois types de stratégie d'exécution suivante :

– La propagation totale qui consiste à simuler un changement, identifier l'impact de ce changement et le propager à tous les artefacts concernés à travers les différentes relations conductrices de l'impact.

– La propagation sélective qui consiste à propager l'impact à travers un nombre restreint de types de nœuds ou de relations.

– La propagation de type Change-And-Fix (Rajlich, Gosavi, 2004) qui consiste à simuler un changement et identifier l'impact aux seuls voisins directement liés au nœud qui vient d'être affecté par le changement.

La mise en œuvre de ces trois stratégies est simplifiée par l'utilisation d'un système de réécriture de graphes. En effet, ces systèmes proposent plusieurs manières d'exécuter les règles qui se distinguent par la façon de réaliser les morphismes ou mappings entre une *LHS* d'une règle et les parties du graphe *Hôte*. Dans le cas d'une propagation totale, on optera pour l'option d'exécution automatique qui laisse au système le soin de trouver lui même tous les morphismes possibles et donc toutes les règles exécutables. Dans le cas d'une exécution sélective, un regroupement des différentes règles par ensembles homogènes est préalable. Il suffira alors de déclencher le ou les ensembles de règles désirées. Dans le cas du Change-And-fix il faudra choisir l'option qui permet à l'utilisateur de choisir lui même les morphismes à chaque pas d'exécution. En effet, on peut demander au système de détecter les règles applicables à chaque pas d'exécution et choisir celle qu'on veut appliquer.

Listing 1 – Algorithme impactGeneration(artefact:a change:c)

```
1  R is a set of relationships having a as source or destination artefact
2  forAll Ri in R {
3      if Ri.forwardImpact then
4          if Ri.CertainImpact then
5              forAll d in Ri.destination {
6                  Ri.generateImpact('certainImpact', d)
7                  }
8          else
9              forAll d in Ri.destination {
10                 Ri.generateImpact('conditionalImpact', d)
11                 }
12         endif
13     endif
14
15     if Ri.inverseImpact then
16         if Ri.CertainImpact then
17             forAll s in Ri.source {
18                 Ri.generateImpact('certainImpact', s)
19                 }
20         else
21             forAll s in Ri.source {
22                 Ri.generateImpact('conditional', s)
23                 }
24         endif
25     endif
26 }
```

4.2. Automatisation de la génération des règles de propagation de l'impact

L'utilisation des règles de réécriture de graphes est un moyen de mise en œuvre flexible des processus d'analyse et propagation de l'impact du changement des applications centrées BPM. Ces règles peuvent servir comme un moyen de validation des processus de mise en oeuvre du changement. Les règles sont essentiellement basées sur une formalisation des différents constituants ou artefacts d'une application en forme de graphes et leur application est basée sur des morphismes essentiellement syntaxiques. Il est donc nécessaire de définir manuellement toutes les règles nécessaires à la propagation de l'impact en considérant chaque type de relations. Nous avons estimé qu'il serait nécessaire de simplifier ce processus en permettant son automatisation. Cela consiste à mettre en place un algorithme qui a pour entrée les artefacts à changer et le changement désiré et qui génère automatiquement des règles de propagation de l'impact à travers les différents types de relations. Il n'est malheureusement pas possible de mettre en oeuvre ce type d'algorithme sur des bases essentiellement syntaxiques. Cela montre donc la nécessité d'une explicitation de la sémantique par des langages et outils issus du web sémantique comme OWL. En utilisant l'explicitation d'une partie de la sémantique des relations par rapport à la conduction de l'impact, nous avons défini un algorithme permettant la génération de règles de propagation de l'impact (Listing 1). L'algorithme fait appel à la fonction *generateImpact(ImpactType:String, AffectedNode: Node)* dans laquelle *AffectedNode* est la source ou la destination d'une relation *Ri*. Cette fonction génère une règle dans laquelle la *LHS* est un sous graphe contenant *AffectedNode* avec l'arc correspondant à

Ri et toutes les extrémités de cet arc. La *RHS* contient *AffectedNode* non directement liée par l'arc correspondant à *Ri* et un nœud de type *ImpactType* est créé et est lié à *AffectedNode*.

5. Prototype d'implémentation

Le prototype que nous avons développé pour la l'implémentation de notre approche est constitué de deux parties. La première partie concerne l'utilisation du système de réécriture de graphe (*AGG*) et de l'outil *Protégé* pour l'élaboration de l'ontologie. Cela nous a permis de mettre en œuvre un prototype flexible permettant une formalisation exécutable des concepts que nous introduisons. Cet outil sert essentiellement dans un environnement de recherche et les utilisateurs sont des personnes capables de maîtriser les concepts issus de la réécriture de graphes et du web sémantique (notamment le langage OWL). Partant de la spécification opérationnelle de notre approche à l'aide des outils *AGG* et *Protégé*, nous avons intégré les processus d'analyse d'impact dans un atelier intégré appelé *Architect* (Ahmad *et al.*, 2013) que nous avions déjà développé et utilisé dans le cadre de l'analyse d'impact du changement des applications distribuées. Cet atelier se présente comme un ensemble de plugins Eclipse permettant des fonctionnalités incluant le parsing de code sources multi-langages (Java, C++, fichiers de configuration spring et J2EE, schémas de bases de données, etc.); le stockage des différents artefacts dans une base de données en forme de graphes au

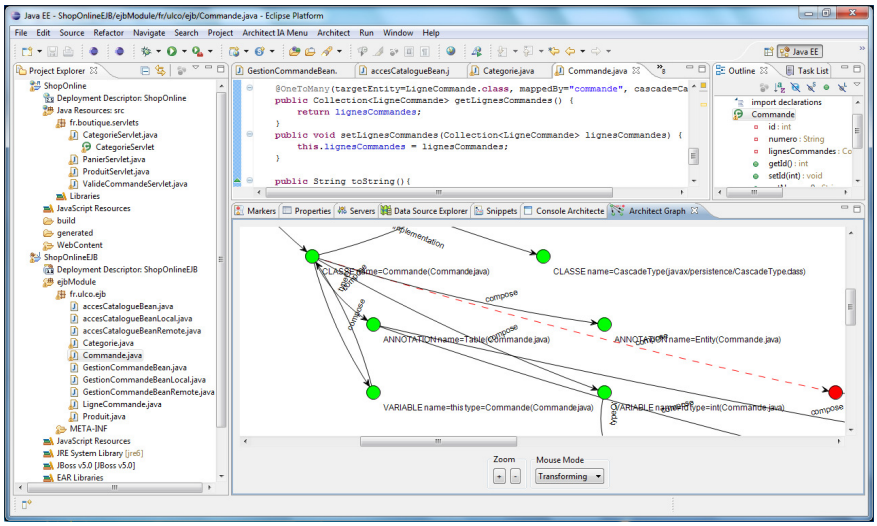

FIGURE 7. *Scénario de la propagation de l'impact dans l'atelier* Architect

format *GXL* [10]; la définition de règles d'analyse et de propagation de l'impact du changement en utilisant le moteur de règle *Drools* [11] et la visualisation et la manipulation à base de graphes des différentes artefacts ainsi que la visualisation de l'impact du changement en utilisant la librairie *Java Universal Network/Graph (JUNG)* [12].

Dans *Architect* (FIGURE 7), les règles de réécriture de graphes ont été transcrites en règles *Drools*. Nous avons donc transcrit chaque règle en explicitant sa pré condition et sa post condition en termes constitués d'objets Java représentant les noeuds et les arcs des sous graphes associés aux pré et post conditions.

6. Conclusion

Nous avons présenté une approche basée sur une méta-modélisation à base de graphes typés et attribués des constituants d'une application centrée BPM et sur les systèmes de règles de réécriture de graphe pour une mise en œuvre flexible de l'analyse et la propagation de l'impact du changement. Nous avons considéré la propagation horizontale de l'impact entre constituants appartenant à la même phase du cycle de vie d'une application centrée BPM et la propagation verticale entre des constituants appartenant à des phases différentes de ce cycle de vie. L'utilisation d'un langage d'ontologies nous a également permis d'expliciter la sémantique des relations inter-constituants en matière de conduction de l'impact du changement. Cela nous a permis de mettre en œuvre un algorithme générique permettant la génération de règles de propagation de l'impact. Les idées émergeant de notre approche ont été validées à l'aide d'un système de réécriture de graphes mais nous avons également mis en œuvre ces mêmes idées dans le cadre d'un atelier intégré que nous développons depuis de nombreuses années et qui est dédié à l'analyse et la propagation de l'impact dans le cadre des applications distribuées incluant, notamment, les applications web multi-tiers.

Nous continuons actuellement notre travail d'explicitation de la sémantique en tentant d'établir des liens de traçabilité entre les besoins des utilisateurs (experts métier) et les BPMs. Le but étant de permettre à un expert d'exprimer les changements au niveau des besoins exprimés en utilisant le vocabulaire de son domaine et d'analyser les impacts sur l'application. Nous avons commencé la mise en oeuvre de cette approche dans le cadre des applications d'optimisation de la chaîne logistique (Hendi *et al.*, 2016).

Bibliographie

Ahmad A., Basson H., Deruelle L., Bouneffa M. (2009, May). A knowledge-based framework for software evolution control. In *Actes du xxviième congrès inforsid*, p. 111-126. Toulouse, France, IRIT Press (www.irit.fr).

10. http://www.gupro.de/GXL/
11. http://www.jboss.org/drools/
12. http://jung.sourceforge.net/

Ahmad A., Bounefa M., Basson H. (2013). *Multi-modélisation de l'évolution du logiciel distribué et hétérogène (french edition)*. Allemagne, Éditions Universitaires Européenes.

Bouneffa M., Ahmad A. (2013a). The change impact analysis in bpm based software applications: A graph rewriting and ontology based approach. In *Enterprise information systems*, p. 280–295. Springer.

Bouneffa M., Ahmad A. (2013b, July). Change management of bpm-based software applications. In *15th international conference on enterprise information systems (iceis 2013)*, p. 37-45. Angers, France, Springer.

Doux G., Jouault F., Bézivin J. (2009). Transforming bpmn process models to bpel process definitions with atl. In *5th international workshop on graph-based tools.*

Folli A., Mens T. (2007). Refactoring of UML models using AGG. *ECEASST*, vol. 8.

Gottschalk K., Graham S., Kreger H., Snell J. (2002, April). Introduction to web services architecture. *IBM Syst. J.*, vol. 41, p. 170–177.

Hendi H. I., Bouneffa M., Ahmad A., Fonlupt C. (2016). Ontology based reasoning for solving passenger train optimization problem. In *Proceedings of aic-mitc 2016. to appear.* IEEE Computational Intelligence Society.

Juric M. B. (2006). *Business process execution language for web services bpel and bpel4ws 2nd edition.* Packt Publishing.

Kherbouche O. M., Ahmad A., Basson H. (2013). Using model checking to control the structural errors in BPMN models. In *IEEE 7th international conference on research challenges in information science, RCIS 2013, paris, france, may 29-31, 2013*, p. 1–12.

Kim D., Kim M., Kim H. (2007). Dynamic business process management based on process change patterns. In *Convergence information technology, 2007. international conference on*, p. 1154–1161.

Maweed Y., Bouneffa M., Basson H., Sack P. O. (2005). Vers une implémentation flexible des activités de maintenance et d'évolution du logiciel. In *Actes du xxiiième congrès inforsid, grenoble, france, 24-27 mai, 2005*, p. 201–216.

Mazanek S., Hanus M. (2011). Constructing a bidirectional transformation between bpmn and bpel with a functional logic programming language. *Journal of Visual Languages & Computing*, vol. 22, n° 1, p. 66 - 89. (Special Issue on Visual Languages and Logic)

Mens T. (2005). On the use of graph transformations for model refactoring. In *Generative and transformational techniques in software engineering, international summer school, GTTSE 2005, braga, portugal, july 4-8, 2005. revised papers*, p. 219–257.

Ouvans C., Dumas M., Hofstede A. ter, Aalst W. van der. (2006, Sept). From bpmn process models to bpel web services. In *Web services, 2006. icws '06. international conference on*, p. 285-292.

Rajlich V., Gosavi P. (2004). Incremental Change in Object-Oriented Programming. *IEEE Softw.*, vol. 21, n° 4, p. 62–69.

Reijers H., Vanderfeesten I., Aalst W. van der. (2016). The effectiveness of workflow management systems: A longitudinal study. *International Journal of Information Management*, vol. 36, n° 1, p. 126 - 141.