HAL
open science

The Change Impact Analysis in BPM Based Software Applications: A Graph Rewriting and Ontology Based Approach

Mourad Mohamed Bouneffa, Adeel Ahmad

To cite this version:

Mourad Mohamed Bouneffa, Adeel Ahmad. The Change Impact Analysis in BPM Based Software Applications: A Graph Rewriting and Ontology Based Approach. Slimane Hammoudi; José Cordeiro; Leszek A. Maciaszek; Joaquim Filipe. Enterprise Information Systems, 190, Springer International Publishing, pp.280-295, 2014, Lecture Notes in Business Information Processing (LNBIP,volume 190), 978-3-319-09491-5. 10.1007/978-3-319-09492-2_17 . hal-03108763

HAL Id: hal-03108763

https://hal.science/hal-03108763

Submitted on 13 Jan 2021

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

The change impact analysis in BPM based software applications: a graph rewriting and ontology based approach

Mourad Bouneffa and Adeel Ahmad

Université Lille Nord de France
Laboratoire d'Informatique Signal et Image de la Côte d'Opale
50, rue Ferdinand Buisson BP 719 62228 Calais Cedex France
{bouneffa,ahmad}@lisic.univ-littoral.fr
http://www-lisic.univ-littoral.fr

Abstract. The Business Process Models describe and formalize the operations, constraints and policies of an organization. These models have firstly been used as abstract views of all the processes implied in an organisation. These served as inputs and outputs of the business analysis and re-engineering activities with no explicit relationship with the IT infrastructures which have been implementing business processes. In this paper, we deal with the BPM as higher abstraction level artefacts of software applications implementing the organisation processes. It presents our approach dealing with the change management of such applications. The approach is based on the graph based formalisation of all the software artefacts including the BPM ones. It provides an explicit management of various relationships conducting the change impact. The change operations are then formalized by graph rewriting (or transformation) rules. These rules implement both the change and the change impact propagation. The semantic knowledge concerning the various artefacts and the change operations is represented by an ontology. This ontology is intended to be able to automatically generate some change management rules. We use graph rewriting system (AGG) as a mean to formally specify and validate the result of our approach. The resulting specifications are then implemented using an integrated software change management platform appearing as a set of the Eclipse Workbench plug-ins.

Key words: BPM, Change impact propagation, Graph rewriting rules, Ontology, Process change management

1 Introduction

The Business Process Models (BPMs) and their components have first been used as first class entities of the Business Process Management activities. The BPM generally encapsulate semi-formal specifications describing the activities of an organisation. Which may lead to build an artefact repository serving as a knowledge base used by the various activities of the Business Management, also including the Business Process Re-engineering. Such activities may

be viewed as a part of the job of a business analyst with no explicit relationship to the Information Technology Infrastructures supporting the organisation's information system. During the last deacade, BPMs have also been used as the first class entities of a new software development methodology based on the transformation of the BPMs into executable programs. It led to the emergence of new software development tools and approaches based on the BPM[Weske, 2007, Weske, 2012] concept. In these approaches the BPMs are specified by means of some standard notations like BPMN [Silver, 2009, Allweyer, 2010] and XPDL[Van der Aalst, 2003, Haller et al., 2008]. The BPMs are then transformed into executable programs that are generally deployed as multi-tiered distributed applications using platforms like J2EE, .NET, etc. The executable programs are often built as macro programs implementing the well known concept of *programming in the large*[Emig et al., 2005]. These programs contain invocations of web services[Gottschalk et al., 2002] provided by the various software applications which have been deployed inside or sometimes outside the information system boundaries. The Business Process Execution Language (BPEL)[Juric, 2006] is one of the most known *programming in the large* language. The main motivation of such an approach is to eliminate the gap between the activities involved in Business Analysis and Information Technology making it more easy and rapid to implement business change requirements.

In a recent paper [Bouneffa and Ahmad, 2013], we considered the study of this new generation of applications and we demonstrate the feasibility of the implementation of a process to control the change impact which may affect these applications. Our approach is mainly based on the use of attributed typed graphs to represent the business process model and software artefacts and the use of graph rewriting system for a formal specification of the BPM changes.

In this paper we enrich our approach by the use of ontologies to explicitly represent more knowledge concerning the BPM, the software artefacts, and the changes affecting them. We consider the fact that a specific relationship between a BPM artefact and a software one conducts the change impact in certain direction. We represent the knowledge concerning the structure of an enterprise and associate BPM artefacts to its one or more structural units. It may respond to queries like which BPM artefacts are affected by a change concerning the particular structural unit? And which software artefact are affected by this change? We can also specify queries like what are the change operations concerning a specific artefact type? etc. In this work, we focus on the change impact propagation aspect which can be achieved by associating more semantic information to the relationship types.

Our approach is mainly based on an ontology, which is built in an interactive and incremental manner. This ontology concerns both the business analysts and software engineers. For this purpose we have been using a simple tool to build such an ontology. For instance, we used the Protégé tool[1] as an assistance to build OWL[2] ontologies using graphical and interactive user interface. We

[1] Protégé : `http://protege.stanford.edu/`

[2] OWL Web Ontology Language : `http://www.w3.org/TR/owl2-profiles/`

also developed a semantic annotation tool to assist the business analysts and software engineer to annotate the business models and software artefacts with those belonging to ontologies. In fact, all the artefacts are yet stored as graph elements of attributed and typed graphs[Bouneffa and Ahmad, 2013]. The annotation mechanism provide help to add more semantical information to these elements.

In Section 2 we present, the meta-model for the BPM formalization and the notions relevant to BPM and BPM-based software applications. The section 3 specifies a taxonomy of BPM change operations and we formalize these operations with the help of graph rewriting rules. The section 4 presents the change impact analysis and propagation process along with its formalization by the graph rewriting rules. The section 5 explains the use of ontology to associate more semantic information to BPM and software artefact. It also presents a generic algorithm we used to automatically generate graph rewriting rules in order to manage the change impact propagation. The section 6 shows the prototype implementation of our specifications regarding the integrated platform to control the software changes[M.O. et al., 2010]. The section 7 summarizes the contribution with the conclusion and the perspectives of this work.

2 BPM formalization and meta-modeling

Before explaining the formalization and meta-modeling of BPM, we first explain the concept of BPM and especially the life cycle of BPM based software applications. As shown in Fig. 1 the development, deployment, and evolution of BPM based software applications obey a life cycle. The different phases of BPM life cycle are described in the following sections :

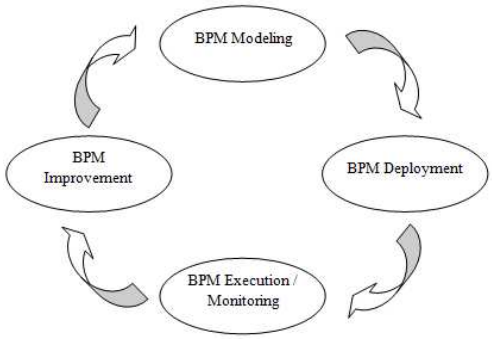

Fig. 1. The life cycle of BPM based applications

2.1 The BPM Modeling

This phase involves BPM Modeling in terms of tasks or activities which are necessary to implement the process, the order of tasks accomplishment, the human

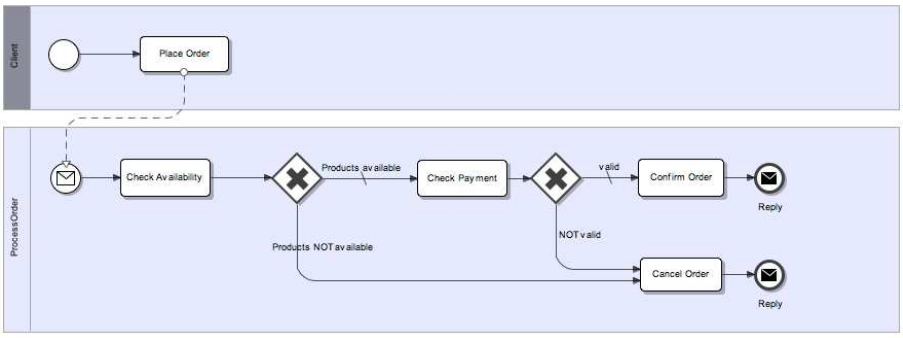

Fig. 2. An example of Business Process Model Notation

actors involved in the performance of these tasks, etc. In recent years, several models or notations have been defined to model the BPM. At first, designing a BPM is an activity within the scope of the information system cartography. It was most often performed as a part of BPR (Business Process Re-engineering) projects [Lee et al., 2011]. Thus, many models and methods have been used such as the OSSAD method[Dumas et al., 1990], etc. Our present work is particularly related to the BPM as a means of specification, development, and deployment of automated processes. We selected the widely considered and used notations in this area, in particular the BPMN [OMG, 2011]. Fig. 2 shows an example of such a process. In this figure, the process is a partial description of the *Sales Chain Management*. We first distinguish two important actors : the *Client* and the *Process Order*. At the beginning, a start event represent the fact that *Place Order* is the first task. This first task performed by the *Client* consists of the generation of an order that is sent as a *message* to the *Check Availability* task, which is linked to a gateway involving the *Check Payment* task. If the products are available or the *Cancel Order*, otherwise. The process ends by end events linked with *Confirm Order* and *Cancel Order* tasks.

2.2 The Development and the Deployment of the BPM

The development and the deployment of a BPM are two separate activities that can be performed manually, automatically, or usually semi-automatically. In principle, these activities can be considered as classical operations for code generation, where the BPM plays the role of a detailed design and the code is represented by an application, deployed most often on a web platform. There exists also software tools to automate the deployment of such applications almost transparently. Some of these tools are Bonita[3], Intalio[4], BizAgi[5] and Barium Live![6], etc. In these tools a web application is generated and is hosted in form

[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/`

of dynamic web pages, a Java servlet engine, or ASP.NET pages, etc. Without going to the full automation and complete transparent development and deployment of BPM based applications, there are intermediate languages playing the role of orchestrators or macro programs involving software components already encapsulated by web services. BPEL is one of the main languages of this type and appears like a sort of standard in the matter. In one perspective of Model Driven Engineering (MDE) [Schmidt, 2006], BPMN can be considered as a Platform Independent Model (PIM) and BPEL as a Platform Specific Model (PSM) consisting of implementing BPMN in an environment using web services as a means of communication and interoperability.

Fig. 3 shows an example of BPEL implementing the BPM shown in Fig. 2. In this figure the BPEL is a kind of web services orchestration. To do this, the BPEL contains calls or invocations to web services like *Check Availability*, *Check Payment* and *Cancel Order* and flow management nodes like *sequence* for a sequential execution of web services invocations or *If* for conditional branches. It is useful to remark that the *programming in the large* concepts are quite similar to the *programming in the small* [DeRemer and Kron, 1975] ones. For our

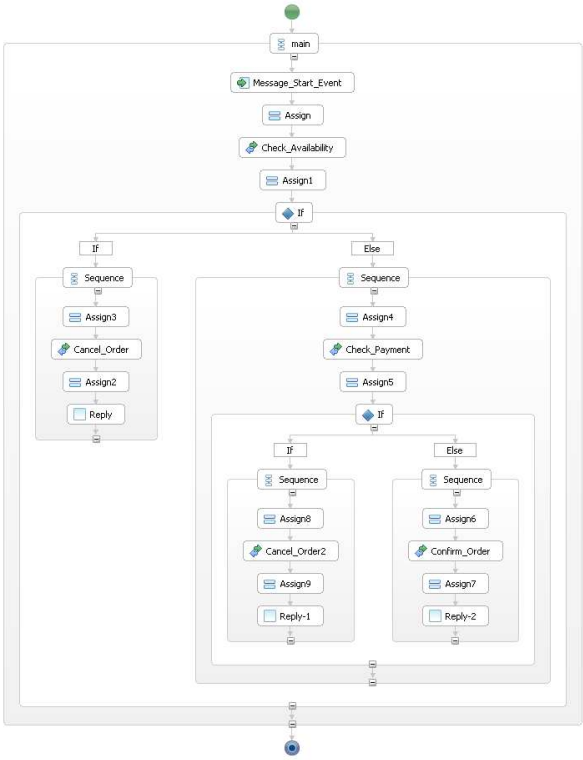

Fig. 3. An example of Business Process Execution Language

example, the tasks or activities of the BPMs are implemented by web service calls while *gateways* are implemented by *If* nodes.

2.3 The execution and monitoring of the BPM

The execution of the application is generally assured by a web platform which is usually on multi-tiered architecture etc. The BPM execution provides generally some interesting data such as response time, resource consumption, etc. These data are necessary for the purpose of analyzing the process quality. Indeed, business managers define performance indicators (Key Performance Indicator[Parmenter, 2007]) for each process, to measure the performance of activities implemented by processes. Some of these informations can be obtained by dynamic analysis by means of program profiling techniques[Ahmad et al., 2008b, Ahmad and Basson, 2009, Ahmad et al., 2009]. Other information are exclusively provided by human experts and will be explicitly taken into account by some organizational process. They are generally the derived data from activities within the framework of customer satisfaction, etc.

2.4 The BPM improvement

The process improvement is a generic term that refers primarily to the evolution of BPM processes. In reality, the improvement is expected but what is actually done, is an evolution of a process embodied by the change affecting the BPM processes. The goal is to fix certain performance anomalies or simply to complete the automation of processes, a part of which is manual, etc. In the literature, the improvement is seen only on the process side and it ignores the software implementation problems. In our case, we consider both aspects, analyzing in particular the BPM change impacts on the software and *vice versa*.

2.5 A meta-model of graph-based BPM

We propose a meta-model to represent the concepts involved in the definition of BPMs. This meta-model has been formalized by a typed graph that we implement particularly in the context of the AGG[7]. The result of this modeling is shown schematically in the Fig. 4. This figure represents the main concepts emerging from the BPMN. The process concept represents the processes that contain what is called *flow objects*. These objects can be *tasks* or *activities*, *sub-processes* or *macro-tasks*, refined by the *processes*, *events* or *gateways*. A process has an actor which is called the *owner* which can be one user, or more, who has defined the process and is authorized to change this process. The process is implemented by an application which is deployed and which can host the execution of multiple instances or cases of this process. Every instance involves actors that are the users interacting with the various tasks performed during the instance life cycle. The typed attributed graph formalism may be viewed as a mean to

[7] `http://user.cs.tu-berlin.de/~gragra/agg/`

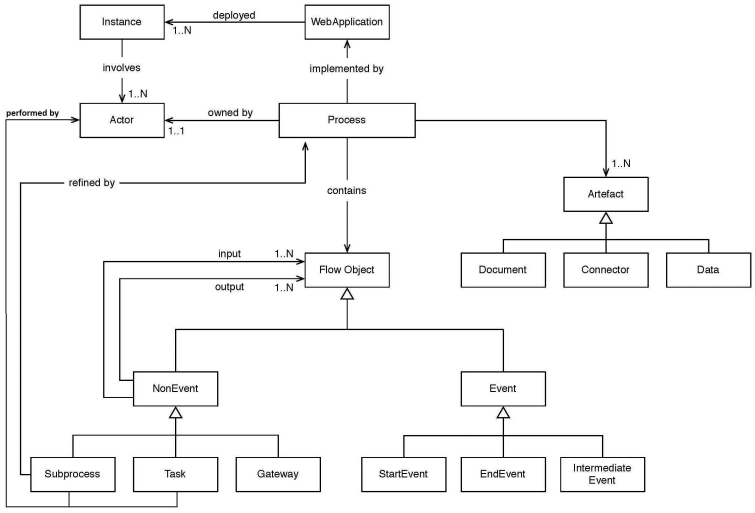

Fig. 4. A UML based representation of BPM meta-model

check the BPM artefacts consistency. In fact these artefacts are also formalized by graphs.

3 The taxonomy of change operations

In the development and the deployment of BPM-oriented applications, the change seems inevitable. It enrolls in fact, in the usual life cycle of this kind of applications. We define it as a taxonomy of change operations that may affect one of the important components of these applications to know the BPM. We consider two kind of changes : the atomic change operations and the composite (or complex) change operations.

3.1 The atomic change operations

The formalization of a BPM as a typed attributed graph allows us to compile a list of atomic change operations. It is important to notify that, "each change operation corresponds to an insertion, deletion or modification of a node or an edge of this graph". We thus obtain the following change operations :

- Insert or Delete or Modify a process.
- Insert or Delete or Modify a task.
- Insert or Delete or Modify an Event.
- Insert or Delete or Modify a Gateway
- Insert or Delete an edge or link (between two flow objects)
- *Et cetera.*

Each defined atomic change operation is then formalized by graph rewriting rules. A graph rewriting rule is in fact a production rule where the left and right sides of the rule are graphs. In other words, a production rule that transform a part of the graph which match or corresponds to the Left Hand Side (LHS) of the production by another subgraph represented by the Right Hand Side (RHS) of the production. There are also preconditions called negative or NAC, which specify the need for non existence of certain sub-graph for the rule to run.

Visually such a rule can be outlined as in the Fig. 5. This figure depicts the partial creation or insertion of a new task. It shows the three components of a rule called *InsertTask* which represents the insertion of a task in a process. The LHS of the rule states that there must be a node in the graph of process type. It would then match a process node of the graph with that of the rule. This matching can be done manually by the user or automatically by the graph rewriting system following many methods that are out of the scope of this paper. The RHS of the rule shows the creation of a task called x connected to the process by the relationship *"contains"*. The NAC of the rule prohibits the creation of a task named x if there is already a task in the process with the same name.

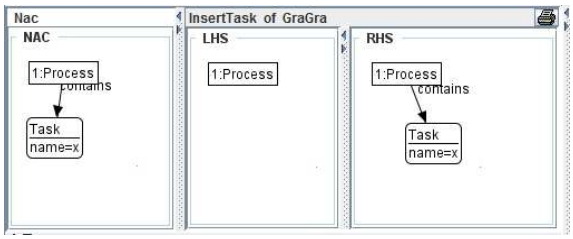

Fig. 5. A rule to insert a task

3.2 The composite change operations

The composite change operations can be expressed in the form of compositions of atomic operations. We did not set a precise or exhaustive taxonomy of these operations but we consider the most significant and frequent ones. It is also possible to define new composite change operations. A comosite operation may be the merger of two tasks, the decomposition of a task into two tasks, the merger of two processes or the breaking down of a task into a subprocess.

4 The change impact analysis

This section deals with the BPM change impact management. A simulation of the change impact generation is presented formerly, using graph rewriting rules and then in later part impact propagation is elaborated to the various artifacts through the different type of relationships.

4.1 The change impact generation

All change operations are defined by the preconditions, which in the case of a graph rewriting rules consist of the LHS (positive preconditions) and the NAC (negative preconditions) and the post-conditions symbolized by the RHS. We therefore consider the impact of a change M given the result of its execution (symbolized as the RHS), in the case of a violation of the precondition. In the case of graph rewriting rules, this is translated into the creation of a node of *Impact* type, connected to the nodes affected by the impact and containing an attribute called *explanation* which contains a narrative of the impact (see Fig. 6). We simulate the creation of the impact by setting rules without NAC and therefore tolerate the enforcement of the rule with the result, in addition to that provided by the original rule, appending a node of impact type linked to the nodes it affects. In Fig. 6, we define a rule to delete a task that provokes the creation of an *Impact* node affecting the tasks related to the task that has been deleted.

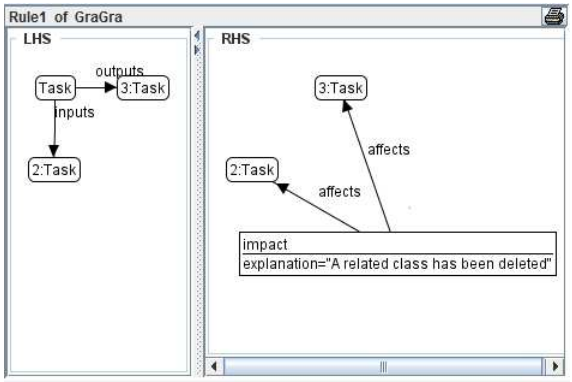

Fig. 6. A rule for change impact analysis of a task deletion

4.2 The change impact propagation

The change impact propagation is a process of propagating the impact to all nodes indirectly affected by the impact. This propagation is done through a link or relationship between nodes. Thus, some relationships are identified as *change impact conductor* [Ahmad et al., 2008a] and propagate the impact in one way or another. For example, the Fig. 7 shows the propagation of the impact affecting a task to the author of this task with the associated explanation.

We distinguish here two types of change impact propagations:

– The horizontal change impact propagation is to propagate the change impact between artifacts belonging to the same phase of the development life cycle

of an application. This is the case of the rule as shown in Fig. 7. It shows the impact propagation between tasks and actors.
– The vertical change impact propagation corresponds to the change impact flow between artifacts belonging to different phases of the development life cycle of an application. This is the case of the change impact propagation between a task and a web service that implements a part of this task and *vice versa*.

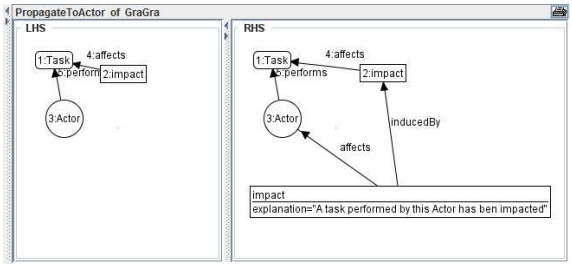

Fig. 7. A rule for change impact propagation

To show how we deal the vertical change impact propagation, we first define a kind of mapping relationships that are useful for the traceability purpose. These relationships are :

– The *mapedTo* relationship between a BPMN process and a BPEL process. In fact, we defined a meta-model of BPEL processes like we have done with the BPMN but the BPEL process contains objects like web services, etc.
– The *ImplementedBy* relationship between a task of the BPM and a web service of the BPEL.

We know that this set of mapping relationships is a restrictive one since it is generally possible that a task may be implemented by more than one web service, it may also be implemented by a process. In another hand a BPMN process may be implemented by a set of BPEL, ones.

The mapping relationships are then used by the graph rewriting rules generating or propagating the impact. So, the Fig. 8 shows the impact generated by the composite change consisting of the merging of two tasks. The question here is what to do with web services implementing these tasks?

On the other hand, we consider three kinds of change impact propagation processes.

– The total change impact propagation simulates the change operation and then execute all possible rules of its impact propagation.
– The selective change impact propagation only propagates the change impacts induced by a subset of nodes relationships.
– The Propagation of type *changes-and-fix* [Rajlich and Gosavi, 2004, Rajlich, 1997] which is to simulate a change, directly addresses the impact of this operation (in terms of direct neighbours). This treatment or correction of the change

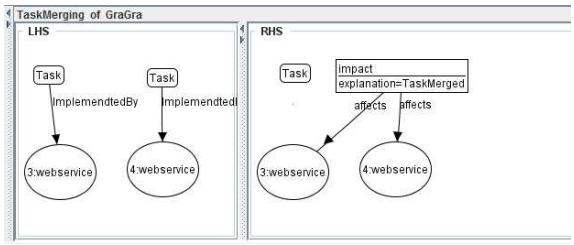

Fig. 8. Rule for the impact generated by the composite change

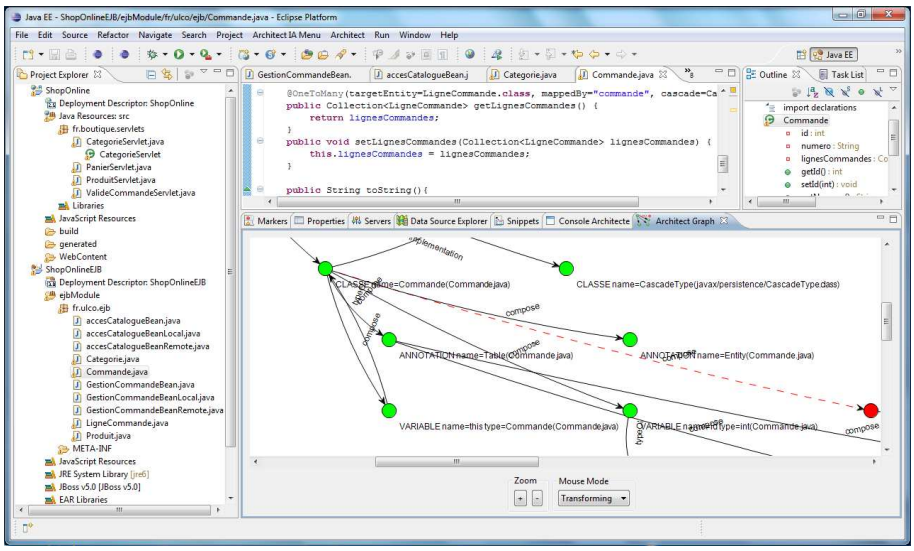

Fig. 9. Screen shot of a change impact propagation scenario in *Architect*

impact will be a transaction which itself will directly impact the address, and so on.

5 The BPM and software artefacts change ontology

The use of graph rewriting rules may help to specify a formal and flexible implementation of the BPM change propagation analysis. Such rules serve as a validation tool and are mainly syntactical since they are only based on the application of syntactical morphisms between artefacts represented by graph elements and the left and right hand sides of the rules. We also want to find a mean to automatically or semi-automatically generate such syntactical rules. The generation of such rules need a more semantic data concerning both the artefacts affected by the change or the change itself. It is therefore we decided to use ontologies allowing the representation and the manipulation of semantic knowledge associated to BPM and software artefacts. This ontology has been used to semantically

annotate the artefacts that are represented by graph elements and to generate the graph rewriting rules managing the change impact propagation.

The ontology we defined may be viewed as a transcription of the BPM metamodel (Figure 4) using the OWL. The significant difference consists of the fact that relationships between artefacts are also described by an *ISA* hierarchy. The relationships are also described by some semantic attributes like "is the relationship symmetric, antisymmetric, reflexive, transitive, etc.". Such attributes can be very useful for the change propagation process. We defined two hierarchies; one hierarchy concerns the artefacts and the second concerns the relationships. The current work is focused on the relationship hierarchy based on aspects relevant to the change impact analysis and propagation (Figure 10). Considering the change impact propagation we define five relationship types:

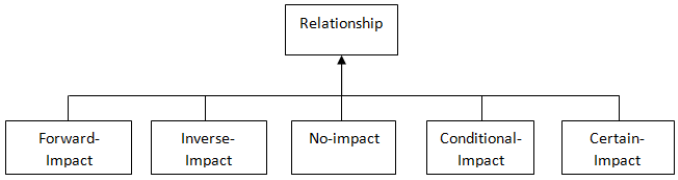

Fig. 10. Relationship hierarchy

- *ForwardImpact* relationships are those conducting the impact from their source or themselves to their destination. That means if a change affect the source of the relationship or affect the relationship itself, the impact may generate and it may also impact the destination of the relationship.
- *InverseImpact* relationships are the relationship conducting the impact from the destination to the source. That means if the destination of the relationship is affected by a change then an impact of this change may generate and this impact may affect the source of the relationship.
- *NoImpact* relationships do not conduct the impact of a change. That means if a change affects the source or destination of the relationship no impact my generate through this relationship.
- *CertainImpact* relationships are those conducting certainly the impact.
- *ConditionalImpact* relationships are those conducting the impact in some cases only.

These relationship types are not disjunctives. That means a relationship may be at the same time *ForwardImpact* and CertainImpact or *InverseImpact* and *ConditionalImpact* or *ForwardImpact* and *InverseImpact*, etc. For instance the *MappedTo* relationship is a *ForwradImpact*. A change affecting a BPM can affect the BPEL implementing it. This relationship is also *InverseImpact* since a change affecting a BPEL may have an impact affecting the corresponding BPM. This relationship is also a *ConditionalImpact* since it is not sure that a change affecting a BPM will really cause a change affecting the corresponding BPEL. With the

help of an ontology language it is then possible to define the new relationship types by combining these.

In our ontology we also represent some descriptions concerning the different kind of changes (as described in the section 3. We add complementary information regarding the artefacts affected by a change, the author of the change, etc. We then define and implement a generic algorithm that generates automatically graph rewriting rules implementing the change impact propagation (listing 1). In this algorithm we assume the existence a function called *generateImpact(ImpactType:String, AffectedNode: Node)* associated to a relationship Ri. This function generates a rule in which :

- the LHS of the rule is a subgraph containing the source and destination of the relationship Ri as linked by the relationship Ri.
- The RHS of the rule contains the source and destination of Ri that are not linked (directly related) by Ri and a node of type $ImpactType$ is then created and linked (indirectly related) to the affected node that may be the source or the destination of the relationship Ri.

Listing 1. Algorithm impactGeneration(artefact:a change:c)

```
R=relationships where a is source or destination artefact
forAll Ri in R {
    if Ri.forward then
        if Ri.impact then
            forAll d in Ri.destination {
                Ri.generateImpact('certainImpact', d)
                }
        else
            forAll d in Ri.destination {
                Ri.generateImpact('conditionalImpact', d)
                }
        endif
    endif

    if Ri.inverse then
        if Ri.impact then
            forAll s in Ri.source {
                Ri.generateImpact('certainImpact', s)
                }
        else
            forAll s in Ri.source {
                Ri.generateImpact('conditional', s)
                }
        endif
    endif
}
```

This algorithm has also been designed to assess the change impact propagation without the use of the graph rewriting rules. In this case the function *generateImpact* does not generate a rule but manipulates directly the artefacts stored in the XML repository as graph nodes and edges using the GXL data model.

6 The prototype of validation

The prototype we developed can be divided on two main parts:

- the first part concerns the graph rewriting implementation and the ontology construction. For this part we mainly used tools such as AGG and Protégé, which allow the graph rewriting specification (and execution) and the ontology construction, respectively. These two tools concerns the specification and formalization of the concepts used by our approach. These tools are not supposed to be actually used in a real project but these help in developing the second part.
- the second part of our validation concern the integration of the results of our approach on a platform we developed to deal with the change management of large distributed software applications. This platform called *Architect* is built as a set of *Eclipse*[8] IDE plug-ins. An Eclipse project manages a set of resources that can be source code files, libraries, BPEL, and BPMN files etc. *Architect* analyzes these heterogeneous sources and parses their elements to represent them as a homogenous interactive graph. The *Architect Graph* extension of eclipse visualizes the elements of the corresponding editor view. This prototype contains a graph editor which provides in a very simple way the nodes and arcs of the structural graph.
  We have used the *Java Universal Network/Graph (JUNG)*[9] Framework. It is a software library that can be re-used for the modeling, analysis, and the visualization of data as a graph or network. This library allows to define the structure of data *Graph* and also to use certain graph primitives for the construction of user interfaces associated with the graph manipulation tools. We used it in interaction with the built-in capabilities of Java API, as well as those of other existing third party Java libraries i.e *Drools*[10]. We have specialized the class *Graph* available in the *JUNG* library in a class that we called *ArchitectGraph*. By using our platform, one can friendly specify a change affecting a node. Which may be a BPM task, a web service specification, or a Java class, etc. The various rules implementing the change impact propagation may then be fired. As a result the new graph is displayed containing nodes of type "impacted" related to the different impacted artifacts along with the explanation of the propagated impact.

7 Concluding remarks and future work

In this paper, we present an approach based on a BPM meta-model intended to serve as a BPM artifacts repository data schema. We also defined the initial BPM change operations taxonomy. It involves the formalization of the change and the analysis of its impact propagation by graph rewriting rules. The graph rewriting

[8] `http://www.eclipse.org/`
[9] `http://jung.sourceforge.net/`
[10] `http://www.jboss.org/drools/`

rules have been implemented with AGG that is a graph rewriting system. This implementation can be considered as an operational or constructive specification. The use of the an ontology language (OWL) provides more semantic information about the various artefacts. The significant semantic information concerns the role of the various relationship types in the impact propagation. We especially define a relationship class hierarchy considering several relationship types from the perspective of the change impact propagation. This allows us to define a generic algorithm that has been used to both generate graph rewriting rules managing the change impact propagation or to directly implement the change impact propagation. The main concept of our approach is validated using a set of tools integrated as ECLIPSE plug-ins. These plugins are parts of a more general integrated framework which is built to deal with the software artifacts change impact propagations.

We are continuing the work by enriching the approach, in a more detailed way, with the different kind of mapping relationships related to the BPM artifacts and their implementation. The main goal is to be able to automatically track the change impact propagation. To achieve this, we must explicitly enrich the knowledge concerning the semantic of BPM elements and their relationships. We are then using the BPMN ontology[Penicina, 2013] that is expressed using the OWL[Motik et al., 2012]. We plan to enrich this ontology by concepts representing the various aspects of change impact and the relationships propagating such impacts. Another goal is to provide some forward and reverse engineering tools in order to implement some important tasks like defining flexible and adaptable tools for the generation of BPM implementations and some others for the generation of BPMs from the process implementations.

References

Ahmad and Basson, 2009. Ahmad, A. and Basson, H. (2009). Software evolution modelling: an approach for change impact analysis. In *Proceedings of the 7th International Conference on Frontiers of Information Technology*, FIT '09, pages 56:1–56:4, New York, NY, USA. ACM.

Ahmad et al., 2008a. Ahmad, A., Basson, H., and Bouneffa, M. (2008a). Software evolution control: Towards a better identification of change impact propagation. In *ICET'08: Proceedings of the 4th IEEE International Conference on Emerging Technologies*, pages 286–291. IEEE Computer Society.

Ahmad et al., 2009. Ahmad, A., Basson, H., and Bouneffa, M. (2009). Rule-based approach for software evolution management. In *IEEE APSSC 2009: IEEE Asia-Pacific Services Computing Conference*.

Ahmad et al., 2008b. Ahmad, A., Basson, H., Deruelle, L., and Bouneffa, M. (2008b). Towards a better control of change impact propagation. In *INMIC'08: 12th IEEE International Multitopic Conference*, pages 398–404. IEEE Computer Society.

Allweyer, 2010. Allweyer, T. (2010). *BPMN 2.0 : Introduction to the Standard for Business Process Modeling*. Books on Demand, Norderstedt.

Bouneffa and Ahmad, 2013. Bouneffa, M. and Ahmad, A. (2013). Change management of bpm-based software applications. In *15th International Conference on Enterprise Information Systems (ICEIS 2013)*, pages 37–45.

DeRemer and Kron, 1975. DeRemer, F. and Kron, H. (1975). Programming-in-the large versus programming-in-the-small. *SIGPLAN Not.*, 10(6):114–121.

Dumas et al., 1990. Dumas, P., Charbonnel, G., and Calmes, F. (1990). La méthode OSSAD - Pour maîtriser les technologies de l'information - Tome 2: Guide pratique, Les Editions d'Organisation, Paris.

Emig et al., 2005. Emig, C., Momm, C., Weisser, J., and Abeck, S. (2005). Programming in the Large based on the Business Process Modeling Notation. In *Jahrestagung der Gesellschaft für Informatik (GI)*, Bonn.

Gottschalk et al., 2002. Gottschalk, K., Graham, S., Kreger, H., and Snell, J. (2002). Introduction to web services architecture. *IBM Syst. J.*, 41:170–177.

Haller et al., 2008. Haller, A., Gaaloul, W., and Marmolowski, M. (2008). Towards an xpdl compliant process ontology. In *SERVICES I*, pages 83–86.

Juric, 2006. Juric, M. B. (2006). *Business Process Execution Language for Web Services BPEL and BPEL4WS 2nd Edition.* Packt Publishing.

Lee et al., 2011. Lee, Y.-C., Chu, P.-Y., and Tseng, H.-L. (2011). Corporate performance of ict-enabled business process re-engineering. *Industrial Management and Data Systems*, 111(5).

M.O. et al., 2010. M.O., H., Deruelle, L., Basson, H., and Ahmad, A. (2010). A change propagation process for distributed software architecture. In *ENASE 2010: Proceedings of the 5th International Conference on Evaluation of Novel Approaches to Software Engineering.*

Motik et al., 2012. Motik, B., Grau, B. C., Horrocks, I., Wu, Z., Fokoue, A., and Lutz, C. (2012). Owl 2 web ontology language: Profiles. w3c recommendation (27 october 2009).

OMG, 2011. OMG (2011). Business process model and notation (bpmn) version 2.0. OMG Document Number: formal/2011-01-03, Standard document URL: `http://www.omg.org/spec/BPMN/2.0` Accessed 2011-03-18.

Parmenter, 2007. Parmenter, D. (2007). *Key Performance Indicators (KPI): Developing, Implementing,and Using Winning KPIs.* John Wiley & Sons, Inc., New York, NY, USA.

Penicina, 2013. Penicina, L. (2013). Choosing a bpmn 2.0 compatible upper ontology. In *The 5th International Conference on Information, Process, and Knowledge Management,*, pages 89 – 96, Nice, France. IARIA.

Rajlich, 1997. Rajlich, V. (1997). A model for change propagation based on graph rewriting. In *Proceedings of the International Conference on Software Maintenance*, pages 84–91, Washington, DC, USA. IEEE Computer Society.

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

Schmidt, 2006. Schmidt, D. (2006). Guest editor's introduction: Model-driven engineering. *Computer*, 39(2):25 – 31.

Silver, 2009. Silver, B. (2009). *BPMN Method and Style: A levels-based methodology for BPM process modeling and improvement using BPMN 2.0.* Cody-Cassidy Press.

Van der Aalst, 2003. Van der Aalst, W. M. P. (2003). Patterns and XPDL: A critical evaluation of the XML process definition language. Technical Report BPM-03-09, BPMcenter.org.

Weske, 2007. Weske, M. (2007). *Business process management concepts, languages, architectures.* Springer, 1 edition.

Weske, 2012. Weske, M. (2012). Business process management architectures. In *Business Process Management*, pages 333–371. Springer Berlin Heidelberg.