Towards a change-aware process environment for system and software process

Mojtaba Hajmoosaei, Hanh Nhi Tran, Christian Percebois, Agnes Front, Claudia Roncancio

Towards a Change-Aware Process Environment for System and Software Process

Mojtaba Hajmoosaei
IRIT, University of Toulouse
Toulouse, France
mojtaba.hajmoosaei@irit.fr

Hanh Nhi Tran
IRIT, University of Toulouse
Toulouse, France
hanh-nhi.tran@irit.fr

Christian Percebois
IRIT, University of Toulouse
Toulouse, France
christian.percebois@irit.fr

Agnes Front
LIG, University of Grenoble
Grenoble, France
agnes.front@imag.fr

Claudia Roncancio
LIG, University of Grenoble
Grenoble, France
claudia.roncancio@imag.fr

ABSTRACT

Managing changes for knowledge-intensive processes like System and Software Engineering is a critical issue but far from being mastered due to the lack of supporting methods and practical tools. To manage changes systematically, a process environment is needed to control processes and to handle changes at run-time. However, such an effective environment satisfying these requirements is still missing. The reason is two-folds: first, operational process environments for system and software engineering is scarce; second, there is a lack of efficient change management mechanism integrated in such process environments.

In order to address these concerns, we aimed at developing a change-aware process environment for system and software engineering. To this aim, we proposed a change management mechanism based on (1) the *Process Dependency Graph (PDG)* representing the dependencies among running process instances managed by a process environment ; (2) a *Change Observer* process to catch change events and update the PDG with run-time information; (3) a *Change Analyzer* component to extract the impacts of change by reasoning the PDG. In terms of implementation, to gain the benefits from the Business Process Community, where many mature Business Process Management Systems have been developed, we chose jBPM to enact and monitor processes. The key strengths of this study are: first, the PDG makes hidden dependencies among process instances emerge at run-time; second, the process observer inside the BPMS allows to handle the change events in a timely manner. Finally, the Neo4j graph database, used to store the PDG, enables efficient traversal and queries.

Keywords

Change Management Process, Change Impact Analysis, Change-Aware, PSEE, BPMS

1. INTRODUCTION

Nowadays, changes in System and Software Engineering projects are almost inevitable due to evolving requirements, resources and technologies. Changes occurring in control-flows, data-flows or resources of a running task can affect, in a chaining-fashion, other tasks either inside one organization or among different organizations. Badly managed changes can lead to unwanted reworks and cause projects to fall behind schedule, go over budget and even fail. Applying a holistic, structured approach to manage changes is then crucial to avoid adding extra cost and risk to both the project and organization levels.

Modern System and Software Engineering (SSE) projects are often extremely complex, involve multi-teams, multi-disciplines and can be realized on multi-sites. Moreover, they are inherently uncertain and highly constrained. Such a complexity makes change management in these domains a challenging issue that is far from being mastered [19]. Generally, SSE processes are coarsely described and manually implemented (especially in small-to-medium enterprises), rely on humans to follow the process models[11]. Consequently, for process performers, there is no clear understanding of process life-cycle and the connections of their tasks to others tasks in the same project or organization. This unawareness results in a loosely controlled process execution where the communication among process performers is ad hoc or ignored and the visibility of development's activities is low. In such a situation, although each organization does have a certain change management process, it is often specially poorly applied due to the lack of a global control on process performers' works. As pointed out in [22], no automation support for change notification and propagation is one of the most popular causes of defects in multi-disciplinary engineering environments.

From a practical point of view, we argue that even a good change management process could be failed if it is not respected by process performers. Thus, first a process environment that offers an overall control on running processes of an organization is needed. To facilitate the changes management process, this process environment should provide the effective mechanisms for first *capturing*

the change, second *analyzing impact* of the change and finally based on the results of the analysis deciding whether *implement* the change or not, all in a timely and systematically manner.

Traditionally the existing process environments are classified into two major groups known as *Process-Centered Software Engineering Environments (PSEE)* [1, 13, 2] and *Business Process Management Systems (BPMS)* [41]. The former belongs to the *Software and System Process* community and the latter belongs to *Business Process* community. In general, BPMSs have attained operational level but they are typically applied in domains with foreknown and predictable activity sequences such as production, business, and logistics [30]. But for PSEEs, afters three decades of development, their maturity has stayed low [13, 24]. This phenomena can be explained by the characteristic of low-level operational and collaborative workflows in system and software development which makes automated process coordination in these domains especially challenging [30]. As pointed out in [24], lack of change management supports is still one of the reasons to the problem of limited acceptance of process environments in system and software industry.

Motivated by the deficiency on the tooling level, we aim at a *Change-Aware Process Environment*. Of course we don't have the ambition to propose a complete solution for change management. In this paper, we are primarily interested in the change notification issue. Change propagation is out of the scope of the presented work. Our objective is to provide an effective assistance for process performers to anticipate change implementation. Concretely, this paper addresses the following questions:

1. how to integrate a change management component into a process environment to make it become *change-aware*?
2. how the change management can identify *automatically* all of the potential process elements impacted by a change in order to notify them *in a timely manner*?

In resolving the above questions, the contributions of our work are:

1. an extension of the Business Process Management System (BMPS) jBPM [20] to implement explicitly a *Change Management Component* that captures in a centralized and continuous way all change requests sent asynchronously by various process performers. By choosing jBPM, our solution benefits from all the advantageous features of an existing, operational BPMS. To some extent, our work helps to bridge the gap between the Software and System Process and Business Process communities.
2. definition and implementation of the *Process Dependency Graph (PDG)* in Neo4j [31] in order to represent dependencies existing among all process elements of the system at runtime. While the dependencies inside one process instance can be extracted at build-time from the process model, the dependencies among different running processes only emerge at run-time via shared resources. In contrast to most of existing works who have concentrated on the dependencies inside one process instance, our proposition describes also inter-process dependencies, thus allows a more thorough impact analysis of changes. When a change request is made, PDG provides a sound and effective basis to traverse inside the process instance and also through other process instances to derive the affected elements.

The remain part of the paper is organized as follows. Section 2 gives a brief description of our approach based on a motivating example. Section 3 presents the dependency graph *PDG*. The architecture of our *Change-Aware Process Environment* as well as its implementation are reported in Section 4. Some related works are discussed in Section 5 and Section 6 concludes our paper and presents some on-going and future works.

2. APPROACH

In this section first we describe a typical situation of a change in process execution through a simple example and focus on some problem of synchronization if this change is not well managed. Then we present our general approach to tackle this issue.

In [19], the authors classified change process existing inside companies into two types: $official$ and $unofficial$. *Official change process* is a macro-level process defining formal protocols to be respected to handle changes concerning to a company or a product. Normally at this level the change process is rather well defined and conducted. *Unofficial change process* happens generally inside technical processes, in the pre-certification phases, as *backwards patching/debugging redesign* processes where developers attempted to fix a problem quickly during the development. This type of process is often informal or semi-formal and poorly managed due to the lack of coordination among developers.

In this paper we focus on the problem of change management in the context of unofficial change processes. The following example illustrates a situation of an unofficial change based on our industry partner's real process.

2.1 Motivating Example

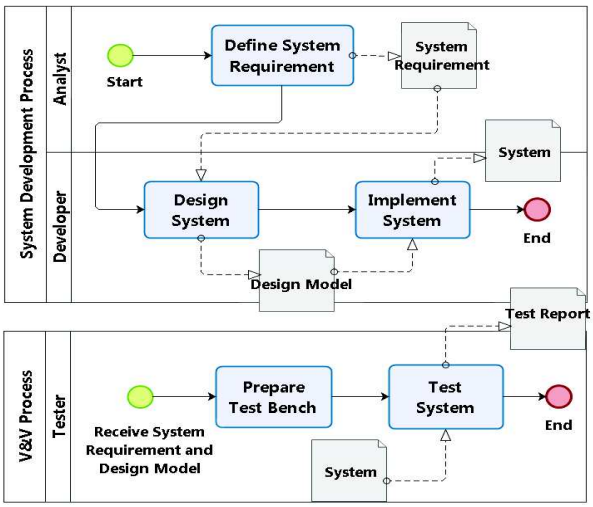


Figure 1: Processes of a System Development phase

Fig.1 shows a portion of a system engineering process simplified from the V-model. This example describes the happy path of the system development phase containing two technical processes *System Development* process denoted as P_{sd} and *Verification & Validation* process denoted as P_{vv}. In our example, these two processes are performed by two different teams, respectively *System Team* and *Test Team*. The *System Team* defines two roles *Analyst* and *Developer*; the *Test team* has one role *Tester*.

The process starts when the *Analyst* receives the stakeholders' needs and carries out the activity *Define System Requirement* to specify the work product *System Requirement* that represents characteristics of the future system (or sub-system) to be developed.

To simplify the process model, we consider the work product *System Requirement* that includes all necessary information for starting the development project, such as verification plan. Afterwards, *Developer* performs the activity *Design System* and produces the *Design Model*. Then, while the *Developer* performs the activities *Implement System* to produce the system, in parallel *Tester* starts the activity *Prepare Testbench* to develop the test environment that will be used to verify the system resulted from *Implement System*. These two general processes can be used for various development projects, thus, may be adapted to a specific context of a given project as illustrated in the following scenario.

Scenario Δ: Assume that there are two development projects, denoted as $proj_1$ and $proj_2$ which use the same processes in Fig.1 to develop two different systems. Each project contains then two process instances respectively belonging to the *System Development* and the *V&V* processes. Thus, in the development environment of the organization there are four process instances denoted as $p_{1.sd}$, $p_{1.vv}$, $p_{2.sd}$, $p_{2.vv}$ [1]. The scenario includes an analyst a_1, two developers d_1, d_2 and two testers t_1, t_2. a_1 and d_1 participate in both projects; d_2 and t_1 participate in $proj_1$; t_2 is involved in $proj_2$.

We consider the following change that is identified as a common change that frequently occurs during system development.

Change: Assume that in $proj_1$, the analyst a_1 has defined the system requirement SR_1. The developer d_1 is implementing the System S_1 and the tester t_1 is preparing the test bench for it. Analyst a_1 switches to $proj_2$ in order to define the system requirement SR_2. In the meanwhile, project manager informs the analyst a_1 of a change in stakeholders needs in $proj_1$. Then analyst a_1 returns to $proj_1$ and produces the new version of the system requirement SR_1.

In practice, if the organization does not have a supporting tool to coordinate the execution of their processes, such a change may happen with the ignorance of the concerned actors. For example, the analyst a_1 can inform the developer d_1 in his team about the change but forget to notify the tester t_1 in another team. As a result, the tester t_1 may be unaware of the change on SR_1 and continue his works based on an out of date requirement. Consequently, the test environment being developed by t_1 will be obsolete and not corresponding to the real system developed by d_1. This situation always requires costly and stressful reworks for the *Test team* as well as for the project $proj_1$. Furthermore, because of extra works for dealing with the change request in the project $proj_1$, the analyst a_1 may perform his job in the project $proj_2$ with delay. This delay may create a domino effect and add delays to the work of other actors in the project $proj_2$.

The above case illustrates some negative impacts of a change that is badly managed. Of course this situation could be avoided if all the actors respect the change management process so that no unnoticed change happens. Unfortunately, in reality, the scenario discussed in the above example is frequent, and it occurs in a *throughout manner* due to the lack of a supporting tool to handle a change.

2.2 Proposal

The image in left of Fig.2 shows the current model of a development environment where an explicit *Change Management Process* is absent. Consequently, changes are handled manually, communication between concerned actors is free and ad-hoc. In such environments, the poor coordination can lead to unawareness of changes which in its turn can lead to reworks.

[1] $p_{i.j}$ indicates the process j belonged to project i

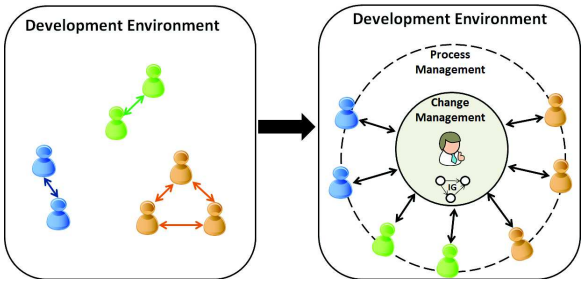


Figure 2: Current and Desired Development Environment

In this paper, we seek to remedy the problem of unnoticed changes in order to permit concerned people to anticipate and respond to changes in order to avoid obsolete works. The ultimate goal of our work is replacing the model in left by the model in right of Fig.2. In the desired development environment in the right, we aim at providing an explicit, centralized *Change Management Process* which is reactive to change requests but proactive to change implementations. More precisely, first our *Change Management Process* must be able to capture all the change requests sent from different actors in the development as they detect some problems in their current works. Second, our *Change Management Process* must have a global view on the status of running elements inside the development environment.

To satisfy the first requirement, we consolidated process management mechanism and change management mechanism. Thus, our *Change Management Process* was developed as a component of a process environment that provides a controlled environment to systematically execute and monitor processes. Section 4 will present in details this solution.

The second requirement was fulfilled by integrating the process management mechanism and the query mechanism on the runtime process data. The result of this consolidation is the abstract graph *Process Dependency Graph (PDG)* which represents dependencies existing among running processes in the system (c.f. Section 3 for a more throughout description of PDG). This knowledge provides a sound base to analyze the impact of change among process elements in a timely and systematically manner.

3. PROCESS DEPENDENCY GRAPH

In general, existence of shared process elements lead to the dependencies among processes. As mentioned before, because of these dependencies, effects of a change in one process can extend to other dependent elements in other processes (as illustrated in the scenario Δ via the share of actor a_1 by two projects $proj_1$ and $proj_2$). In order to investigate the impact of change, we need to have a global view about the system. In other words, we need a structure that specifies the existing dependencies between all running processes in the system. To this aim, we define a structure named as Process Dependency Graph (PDG). PDG is constructed based on the process information existing in the development environment. In the next section, we describe this information thoroughly.

3.1 Process Information

Generally, process information existing in the development environment can be seen in two levels:

Schema-level: this level relates to process information existing at build-time that is extracted from the process models. For example the process models in Fig.1 give us the following information:

• List of system projects:
$Project = \{system\ project\}$
• List of processes models:
$P^{models} = \{P_{sd}, P_{vv}\}$
• List of roles:
$role = \{analyst, developer, tester\}$
• List of task definitions:
$task = \{define\ system\ requirement,\ design\ system,\ implement\ system,\ prepare\ testbench,\ test\ system\}$
• List of work products:
$workproduct = \{stakeholder\ need, system\ requirement, design\ model,\ system,\ test\ report\}$
• Temporal information:
any information about processes and activities duration.

Instance-level: this level relates to process information existing at run-time. When a process is executed, its model is instantiated and we obtain a process instance representing the running process. If a Process Management Suite (PMS) is used to manage and execute processes, at run-time PMS stores the information of running process instances in its process logs (repositories). Thus, based on these process logs, we can extract information about process instances in the system. For instance, the process logs of the system corresponding to the scenario Δ give us the run time information as illustrated in Fig.3. This figure illustrates all process information in both schema and instance levels. An element at instance level (marked as the white circle) is an instance of an element in the process model at schema-level (marked as the gray circle). For example, TR_1 and TR_2 are two instances of the work product *Test Report*, respectively belonged to process instances $P1.vv$ and $P2.vv$.

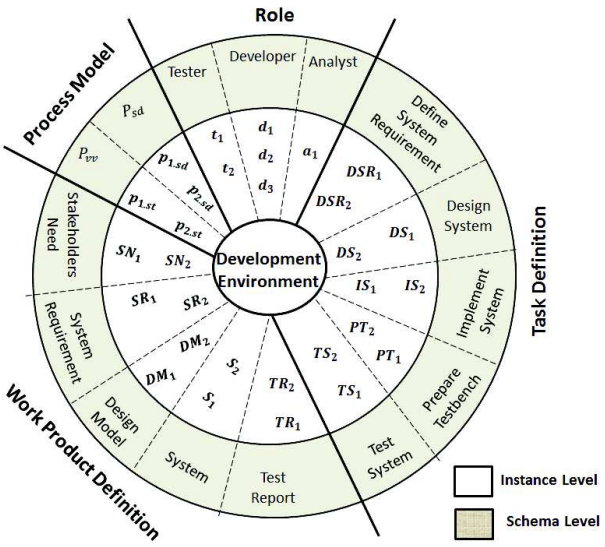


Figure 3: Process Information for the scenario Δ at schema and instance level

3.2 Structure of PDG

As mentioned above, PMS maintains a significant amount of information about the project under development in the process logs that can be manipulated as a database. However, making complex queries directly from the process logs is not an easy task because of the heterogeneity of the process data together with the process log's access characteristics (within the context of activities that are long-lived, open-ended, and interactive) [4]. That's why we propose PDG, an abstract graph to represent process information existing in the development environment. Querying about the status of running process instances on the PDG will be more efficient than querying directly the process logs.

PDG is defined as a directed graph composed of a set V of nodes and a set E of edges. Nodes and edges of a PDG are typed to describe different types of process elements (at instance level) and relations between them. We have three types of nodes and three types of edges, specified in the following definitions:

$PDG = (V, E)$
$V : \{node_{task}\} \cup \{node_{workproduct}\} \cup \{node_{actor}\}$
$E : \{edge_{data}\} \cup \{edge_{precede}\} \cup \{edge_{perform}\}$

3.2.1 Nodes of PDG

Each type of nodes has properties and status which are defined as follows:

• $\boldsymbol{node_{task}}$: represents a task of a process instance.
$node_{task} = (task_{name}, task_{id}, process_{id}, task_{status}, duration)$
$task_{status} \in \{created, inprogress, completed, failed\}$

A newly created task starts in the "created" stage. The task will stay "created" until one of actors claims the task, indicating that he or she will be executing it. When the user who has claimed the task starts executing it, the task status will change from "created" to "inprogress". Lastly, once the user has performed and completed the task, the task status will change to "completed". In this step, the user can optionally specify the result data related to the task. If the task could not be completed, the user could also indicate this by using a fault response, possibly including fault data, in which case the status would change to "failed".

• $\boldsymbol{node_{workproduct}}$: represents a concrete work product (data) inside a process instance. The status of a work product is the same as the status of the task producing it.
$node_{workproduct} = (wproduct_{name}, wproduct_{id}, wproduct_{status})$
$wproduct_{status} \in \{created, inprogress, completed, failed\}$

• $\boldsymbol{node_{actor}}$: represents a real actor in the system. An actor may involve in several process instances. We assume that the status of an actor is managed by both the task management component and the resource management component integrated to the PMS.
$node_{actor} = (actor_{info}, actor_{status})$
$actor_{status} \in \{not\ available, idle, active, waiting\}$

Actor can be seen as a special kind of resources of an organization. Other kind of share-able resources as test-benches, locals, etc. can also be treated in a similar way.

3.2.2 Edges of PDG

Edges in PDG are used to represent different types of relationships among process elements. Properties and status of each type of edges are defined as follows:

• $\boldsymbol{edge_{perform}}$: represents the association between an actor and the task that he performs. Status of the edge is the same as the status of the task that it points to. The reason to store this information twice is to facilitate the traversal that we will discuss later.
$edge_{perform} = (edge_{info}, process_{id}, t_{start}, duration, status)$
$status \in \{inprogress, completed, waiting\}$

• $\boldsymbol{edge_{data}}$: represents the association between a data and the task that uses or produces it:
$edge_{data} = (edge_{info}, node_{src}, node_{des}, status)$
$status \in \{sent, not-sent\}$

• $\boldsymbol{edge_{precede}}$: represents association between two sequential tasks that share nothing between each other (i.e., no exchanged work

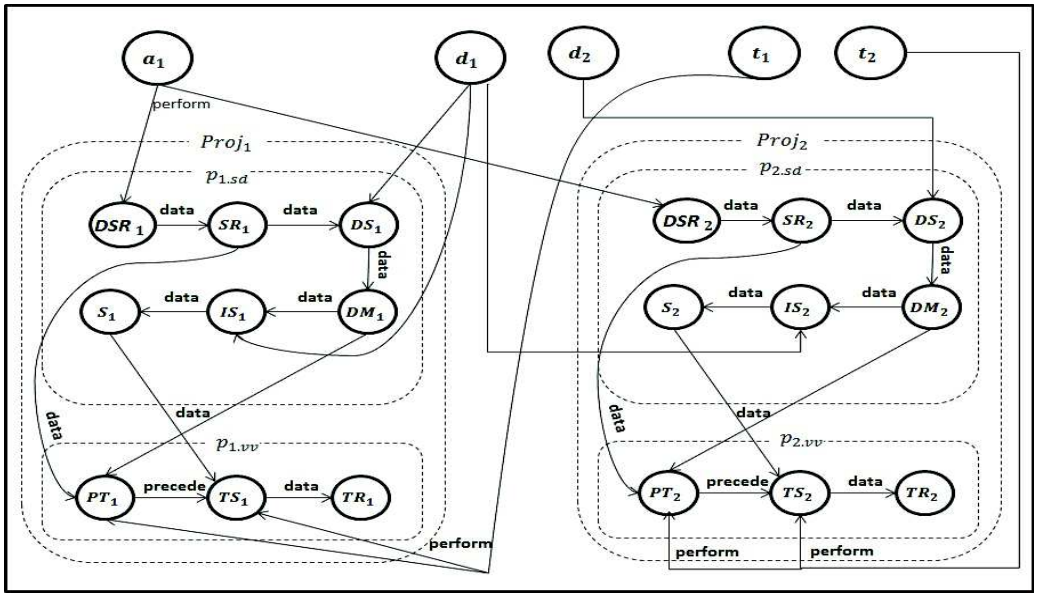


Figure 4: PDG representing the scenario Δ

product or shared actor).

$edge_{precede} = (\ edge_{info})$

In order to clarify the structure of the PDG, Fig.4 is provided to illustrate the PDG presenting the system in the scenario Δ at run time. To simplify the visualization of the PDG, two projects along with their processes are annotated inside the dashed rectangles. In our scenario, shared elements among processes are established by the work products *System Requirement SR* and *System S* inside each project and the actor a_1 among two projects.

3.3 Storing PDG in a Graph Database

The PDG describing the running processes of an organization can be huge. Thus, in order to store and manipulate efficiently the PDG, we explored the use of NoSQL data management system, in particular on systems proposing native graph data management. A graph database is typically substantially faster for connected data sets and uses a schema-less, bottoms-up model that is ideal for capturing ad-hoc and rapidly changing data [39].

To this aim, Neo4j [31], an object oriented and open-source graph databases, has been chosen. Neo4j allows us to store and query the PDG in an efficient way thanks to its following advantages:
- powerful traversal framework for high-speed traversals, [2]
- declarative graph query language Cypher,
- high integration capability for running as embedded in JVM process via a Neo4j Core-Java-API,
- access to the standalone Neo4j Server via its HTTP API.

4. CHANGE-AWARE PROCESS ENVIRONMENT

This section describes our solution to deal with poorly managed changes during execution of processes. Although this work targets more specially on technical software and system engineering process, it can be also used by processes in other domains. Our aim is to consolidate two mechanisms of process management and change management to form a change-aware process environment as illustrated in Fig.5 .

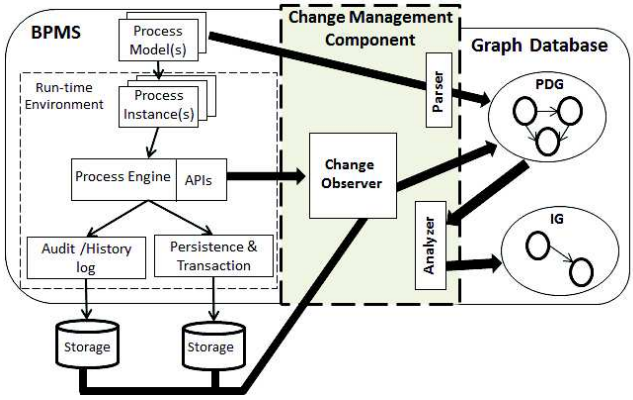


Figure 5: Change-Aware Process Environment

This environment is composed of two main components: a *PMS* to support process management and a *Change Management Component* to handle changes. In the next sections, we explain each component in details.

4.1 Process Management System

As introduced in Section 1, *Process Management System* or PMS is a software that supports modeling, execution, and monitoring of processes throughout their life cycles. In our proposal, PMS provides the basis to control systematically process worker's activities and to obtain a global view on the organization's development environment status. Our idea is to enrich an existing PMS with a change management mechanism and not develop a new PMS from scratch. Moreover, to facilitate the adoption of our work, we consider the PMSs supporting the standardized process modeling languages. The benefits of this choice is obvious, but it also imposes some limitations on our selection (not too many choices) and some

[2]According to [15] Neo4j traverses depths of 1000 levels and beyond at millisecond speed.

difficulties on our implementation (dependent on a complex architecture and API of an existing system).

Our motivation is assisting *software and system processes*, so first we looked for PSEEs supporting the SPEM [34] modeling language. However, very few academic works as [5, 36] attempting to offer prototypes to enact SPEM process model. There is the commercial tool IBM Rational Team Concert [17], but it offers a limited extensibility. To enact SPEM, the solutions suggested by OMG [34] are: mapping the processes into Project Plans and enacting them with project planning systems; or mapping the process to a business flow or execution language and then executing this representation using a workflow engine.

The later solution is more tempting for us thanks to its large choices of operational and open-source Business Process Management Systems. Most of BPMSs support the business process modeling standard BPMN [35] and their workflow engine execute the operational process model in BPEL [32] or transforming them to their internal process definition structure (object process definition). BPMN is not dedicated to software and system processes, but several solutions have been proposed to convert process models in SPEM to an operational process modeling language for BPMS, such as the works in [42, 9]. Although we did some adjustments and learned some interesting lessons on using BPMN to model software and system processes, discussion about the similarity between these two standards is out of scope of this paper. Here, we make an assumption that we have well-defined process models of software and system engineering processes in BPMN that can be executed in a supporting BPMS.

Based on the features of existing BPMSs, we chose jBPM [20], a flexible, light-weight, fully open-source and extensible BPMS, for developing our PMS component. Like other BPMS, jBPM is composed of several components that each one resolves one particular function inside the BPMS architecture as shown in Fig.5. The core part of jBPM is the run-time environment who receives the process models modeled by an integrated web editor and stored in a process repository. The jBPM process engine is the module in charge of enacting processes. It creates new process instances and keeps track of their states and their internal steps. jBPM also provides persistence for the process's executions by storing running process steps into the *Persistence and Transaction database* to make restoring the process instance as efficient as possible. Furthermore, in order to keep historical information about process executions separately, it uses the *Audit/History logs* that later can be used to conduct process analysis.

jBPM is used as the core of our process environment. Thanks to its API we can extend jBPM by:

1. Developing a new end user interface to allow process workers notifying a change during executing a task in an asynchronous manner,

2. Developing a central Change Management Process and integrating it as a component of the whole jBPM system to capture and analyze any signaled change in the development environment.

The next section presents in detail the *Change Management Component* and the above extensions.

4.2 Change Management Component

The core of our proposal is the change management component. This component turns the process environment (PMS) to a change-aware environment by providing a layer between the process working environment and the core process execution environment (process engine). This layer provides coordination among main roles of the process environment known as process manager, process workers and change manager in order to manage changes systematically inside the system. At this stage of our work, to stay generic, we don't focus on integrating tools or service-tasks but concentrate on human tasks. Thus, we used the jBPM task management component and developed a new end user interface for human process performers. The *Change Management Component* provides different functionalities for each type of process performers. Fig.7 shows some screen shots for different process performers. A *process manager* can instantiate a process as shown in Fig.7(a). For a *process worker*, beside the basic interactions with the process environment to request their task list, to claim and complete the tasks assigned to them, now thanks to the new layer interface, he can signal a change request during his execution as illustrated in Fig.7(b). After signaling the change, the process worker can continue or terminate his work without notifying directly the concerned persons of the change. Fig.7(c) shows the interface for a *change manager* who can capture a change request sent by any process worker then perform an impact analysis, by using the PDG of the development's environment, to identify the elements affected by the potential change. Based on the analysis result, the change manager can decide about the change and inform the concerned elements in a timely manner.

The whole centralized change management process's behavior is described by the sequence diagram in Fig.6.

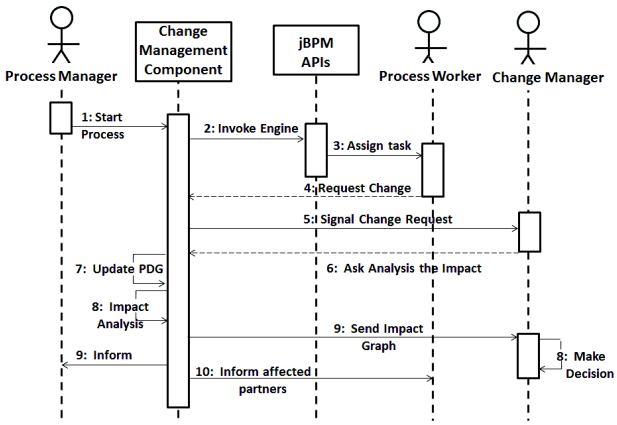


Figure 6: Procedure of Change Management

The above functionalities are provided by three sub-components inside the Change Management : the *Parser*, the *Change Observer* and the *Analyzer* as shown in Fig.5. We discuss each sub–component separately in the following sections.

4.2.1 Parser

The functionality of this sub-component is constructing the structure of a new process instance and adding this structure to the PDG of the development environment (c.f. Section 3). To this aim, we developed a parser that receives a process model and extracts from this model the process information at schema-level to construct the structure of the PDG. Whenever a process engine creates a process instance, it informs the parser of the new instance. The parser then looks into the process repository to find the process model corresponding to the process instance's name, parses the found process model (BPMN file) and finally converts it to the corresponding

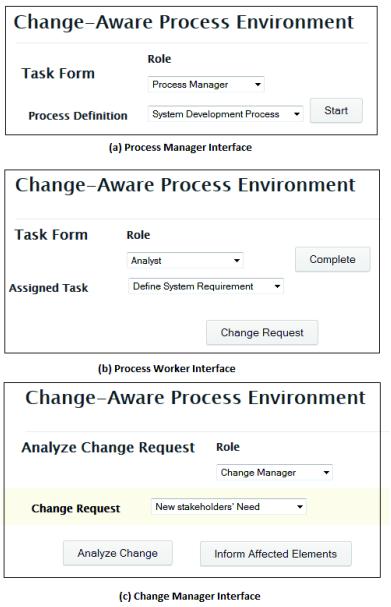


Figure 7: Interfaces for different roles in the Change-Aware Process Environment

graph structure in PDG. Algorithm 1 resumes these steps.

input : *ProcessInstanceName* pi_{name}
output: *ProcessInstaceStructure* pi_{struct}
Procedure `Parser()`
  p_{model} = SearchProcessModel(processrepository, pi_{name});
  $pi_{struct} = Parse(p_{model})$;
  add (pi_{struct}, *PDG*);

Algorithm 1: Parser Algorithm

Afterwards, at the time of change, the PDG will be updated with run-time information (i.e. state of the tasks and process instances) by the *Observer* component as discussed in the next section.

4.2.2 Observer

As explained in Section 4.2, our process environment provides a mechanism to allow process workers to send a change request at any moment when executing a task (c.f. Fig.7(b)). The sub-component *Observer* is responsible to catch the change requests and invoke the *Analyzer* to handle each received change request.

It means that the *Change Management* has to allow the actions *sending change request* which cannot be foreseen and modeled previously in the process models. To enable this concurrency and asynchronous execution of change request signaling, we implemented the *Observer* as an asynchronous handler. Thanks to the advanced features of jBPM 6.0, i.e. *Multiple knowledge sessions, persistence* and *WorkItemHandler backed with jbpm executor*, the implementation of *Observer*, even complicated, was done successfully.

Whenever a change occurs, the process worker (also known as change initiator) signals the change request *CR* along with the specification of the concerned element.

DEFINITION 1. *A change request* (*CR*) *is defined as a tuple* $CR(t_{id}, P_{id}, n_{concerned})$ *where,*
- t_{id} *is the task id,*
- P_{id} *is the process id,*
- $n_{concerned}$ *is the concerned change element.*

In the implementation of the change request *CR*, the t_{id} and P_{id} of the change initiator process are obtained automatically from jBPM logs. On the other side, *Observer*, as a change handler, takes care of these change requests as described in Algorithm 2.

For each received change request, *Observer* updates the schema of the PDG (that is already constructed by the *Parser* as presented in the previous section) by adding to PDG's nodes and edges the run-time information such as current state of the processes, tasks, list of actors, temporal information, etc. This run-time information, achieved thanks to the APIs provided by jBPM, is obtained from two jBPM's process logs: persistence and history log. The former stores the information of the active processes (in-memory) and the latter stores the historical information of completed process instances. Result of this phase is a PDG that represents the run-time situation of process instances at the time of change. Based on the information of *CR*, *Observer* calls the *Analyzer* to extract from PDG the elements affected by the change request *CR*.

input : Change Request *cr* , PDG = (V, E)
output: $Updated\ PDG\ updg = (V, E)$
Procedure `Observer()`
  listen to the change request cr;
  impact graph $\leftarrow n$;
  for *each cr* **do**
    update PDG;
    call Analyzer (updg, $node_{concerned\ node}$);
  end

Algorithm 2: Observer Algorithm

4.2.3 Analyzer

This component has the responsibility of traversing the PDG and extracting the elements potentially affected by a change request *CR*. The result of the traversal is a digraph so-called *Impact Graph (IG)* which is defined as follows:

$IG = (V, E)$
$V : affected\ nodes$
$E = \{e_1, e_2, ..., e_n\}$
$\forall e \in E \mid e = (n_i, n_j) \Rightarrow n_i$ impacts n_j

Invoked by *Observer*, *Traversal* starts by receiving the updated *PDG* and the concerned change initiator $node_{concerned}$. It traverses the *PDG* in two directions by using three types of edges: by $edge_{data}$ and $edge_{precede}$ to traverse inside a process instance; by $edge_{perform}$ and $edge_{data}$ to go outside a process instance.

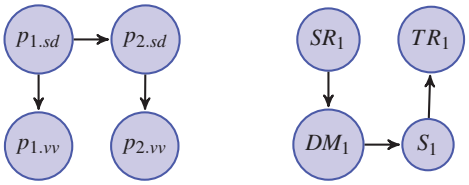
Finally *Analyzer* outputs the impact graph *IG* which is an extraction of the global *PDG* but contains only the process elements impacted by a (future) change. These elements are detected by the emerging dependencies among run-time process elements in the *PDG* based on shared data, actor or on the temporal sequences.

Algorithm 3 gives the pseudo code of the traversal. For its implementation we used the Neo4j Core-Java-API to develop a Neo4j embedded application in our JVM process. Thus, we can benefit not only an object-oriented approach to manipulate the graph database, working with Nodes, Relationships and Paths, but also highly customizable high-speed traversal- and graph-algorithm implementations.

The obtained *IG* is considered as the base to conduct impact analysis at different levels according to a specific need of change manager. For instance, *IG* can show only the dependencies at process, data or actor level. This can be achieved by defining some

query templates in Cypher language [31] that look up the *IG* from different points of interest. For instance, Fig.8 shows the *IG* of the scenario Δ in the process and data levels. In Fig.8(a) we can observe the dependency between two process instances $p_{1.sd}$ and $p_{2.sd}$ (by sharing the same actor a_1, but this information is hidden in the process models). Fig.8(b) shows only the data dependencies among the work products.

In principle, we can go further in such analysis by annotating the nodes of the *IG* with some interesting metrics such as work product completion percentage. That means that, for any affected work product the percentage of its completion at the time of change can be estimated based on the duration of the task that produces it. However, this feature, which is dependent on a specific given domain for calculating the required metrics, is not presented in this paper but in our previous work [14, 27].



(a) IG for Process-level **(b) IG for Data-level**

Figure 8: Example of different Impact Graphs

input : $UPDG = (V,E)$ & $node_{concerned\ node} = n$
output: $ImpactGraph = (V,E)$
Procedure `GenImpactGraph` $(UPDG, n)$
  $queue \leftarrow n$;
  $impact\ graph \leftarrow n$;
  while *queue is not empty* **do**
    $x \leftarrow queue$;
    for *each* y *such that* (x, y) *is* $edge_{data}$ *or* $edge_{perform}$ *or* $edge_{percede}$ **do**
      if $(x,y)\ is\ edge_{data}\ \&\ t_{src}^{before} \neq t_{src}^{after}\ \&\ t_{src}^{after} \not< t_{des}$ **then**
        $GenImpactGraph(PDG, y)$;
        $queue \leftarrow y$;
      else if $(x, y)\ is\ edge_{perform}\ \&\ status \neq completed\ \&\ edge_{perform}\ affects\ by\ adaptation$ **then**
        for *each* z *such that* (y, z) *affects by adaptation* **do**
          $GenImpactGraph(UPDG, z)$;
          $queue \leftarrow z$;
        end
      else if $(x, y)\ is\ edge_{percede}\ \&\ (x.y)^{status} == not\ visited$ **then**
        $GenImpactGraph(UPDG, z)$;
    end
  end

Algorithm 3: Analyzer Algorithm

5. RELATED WORK

Change management can be tackled from different perspectives, such as process perspective, tool perspective and product perspective [19]. Also according to [19], tools and methods to support the change process can be divided into two groups: (1) those that help managing the workflow or documentation of the process and (2) those which support engineers in making decisions at a particular point in the engineering change process (e.g. the risk/impact analysis phase).

We use this structure to discuss some similar works on the tool perspective in Business Process, System and Software Engineering communities.

Work flow/documentation support: computer-based tools has been recognized as an essential to support engineering change [16]. In terms of academic works, Chen et al. [8] proposed a tool to support distributed engineering change management linking with Concurrent Engineering. Lee et al [23] introduced a prototype for collaborative environment for engineering change management which combines ontology-based representations of engineering cases, case-based reasoning for retrieval and a collaboration model. In Business Process community, many of existing works on change management focus on proposing mechanisms to enable process adaptation and changes propagation. Most of the researches [37, 28, 29, 25, 26, 6] have investigated solutions for process adaptation. Reichert et al. gave in [38] a good survey on the flexibility of workflow system in order to response better to changes. However, these studies stay as prototypes which are difficult to be validated for industries.

In terms of commercial tools, in Engineering domain, tools such as IBM Rational Team Concert [17] and Siemens TeamCenter [40] provide the control for collaborative work. However, these tools in general are costly, very complex to use and to customize, thus few companies have adopted them in their environment. In business process domain, many commercial tools have been developed, among them we cite the most interesting such as Bonita [7], jBPM [20] and AristaFlow [3]. The good point of these BPMSs is that they offer partial or full open-source API that enable extensions in staying with standardized and operational environments. Envisaging an validation of our prototype in software and system industry, we need to keep this vision as standard and operational. That's why we developed our academic prototype based on jBPM.

Decision making support: a wide variety of techniques are used in the context of impact analysis and change propagation [16]. There is currently no commercial package that helps predict the effect of a change, however some work is being carried out in academic institutions [19]. A tool called Change Prediction Method (CPM) aims at realizing of how changes spread through a product by using Design Structure Matrix (DSM) as the basis of the product model. The tool uses a simple model of risk, where the likelihood of a change propagating is differentiated from the impact of such an occurrence. This technique has been used in many other works [18, 33, 21]. Grantham-Lough et al [12] applied prediction methods for change propagation and risk estimations based on the functional decomposition of the product in early design stages. Their methods utilize history of design failures and assume that the behavior of past products is sufficiently similar to current or new products. As a result, tools show a diversity of approaches but based on prediction approaches. Their applicability reduces in the real-time change analysis when time plays a major role in informing the change affected partners.

Impact Analysis of change is also an important topic in business process research. In [10], a change propagation approach called Refine Process Structure Tree is proposed to deal with change in process choreographies. This approach addresses both phases of process change but only inside one process instance. Approval of a

change is done by negotiations among change initiator and affected partners. By contrast, our approach derives the impact of change inside and among process instances and also provides useful metrics in order to facilitate the negotiation phase. Muler in [29] dealt with logical failures management in inter-workflow collaboration scenarios and extends the previous work [28] by adding temporal and qualitative implications of workflow adaptation. Temporal implications of an adaptation are determined by estimating the duration required to execute the dynamically adapted workflow and by comparing it with originally fixed time constraints. The metrics used in their approach are for deriving the essence of adaptation not for measuring the impact of the adaptation. Impact of change among process instances was not investigated as they considered only the impact of adaptation in one process instance.

6. CONCLUSIONS

To date, lack of gap between the process management and change management is visible. Companies have a general lack of understanding of how changes are connected; if the change situation is not really realized (lack of communication), a tool can only provide limited help. Advantages of combing both approaches in a tool can lead to decreasing the highly cost of reworks by providing communication and synchronization through the system [19].

Convinced by the above statement, we have developed a prototype of a *Change-Aware Process Environment* that consolidates the process management and the change management into one tool. The assistance provided to process performers allows them handle the change in a centralized and proactive way so that they can better anticipate and response to changes.

One key strength of this work is the tool development that is aligned to existing standards (BPMN, XPDL) and operational process environment (jPBM). Another contribution is the exploitation of run-time process information to analyze the hidden dependencies via shared resources. Thanks to this emergent dependencies, our solution can detect the affected elements outside a process instance initiating the change. The prototype, implemented in Java with the APIs of jBPM and Neo4j, is operational and is being validated with the case studies provided by our industry partners.

In the first stage, this work confirms the possibility of extending a BPMS to manage software and system process. The major limitation of the proposed process environment is that it was not integrated to the working environment of process performer, i.e. separated to their own development tools. We recognized that this point is one of the biggest obstacles for making process environments adopted by industry, especially by software and system engineers. We envisage to study how to make some connections between the process environment and the process performer working tools without losing the genericity of the process environment.

On learning using a (B)PMS for modeling and enacting a real software and system process, we also perceived an important limit of the traditional process environments in practice: they require as input the well-defined, executable process models which, in general do not exist. To remedy this problem, our future works aim at a bottom-up approach for process environment in order to enable operational process model to emerge from the end-users side.

7. ACKNOWLEDGMENTS

Part of this research has been supported by the French research projects ANR InnoServ (Innovation de Services pour personnes fragiles).

8. REFERENCES

[1] V. Ambriola, R. Conradi, and A. Fuggetta. Assessing process-centered software engineering environments. *ACM Trans. Softw. Eng. Methodol.*, 6(3):283–328, July 1997.

[2] S. Arbaoui, J.-C. Derniame, F. Oquendo, and H. Verjus. A comparative review of process-centered software engineering environments. *Annals of Software Engineering*, 14(1-4):311–340, 2002.

[3] AristaFlow. Aristaflow website : http://www.aristaflow.com/.

[4] N. S. Barghouti, W. Emmerich, W. SchÃd'fer, and A. Skarra. Information management in process-centered software engineering environments. In *In A. Fugetta and A. Wolf, editors, Software Process Trends in Software*, pages 53 – 87. Wiley, 1996.

[5] R. Bendraou, B. Combemale, X. Cregut, and M.-P. Gervais. Definition of an executable spem 2.0. In *Proceedings of the 14th Asia-Pacific Software Engineering Conference*, APSEC '07, pages 390–397, Washington, DC, USA, 2007. IEEE Computer Society.

[6] R. Bergmann, A. Freßmann, K. Maximini, R. Maximini, and T. Sauer. Case-based support for collaborative business. In *ECCBR*, pages 519–533, 2006.

[7] BonitaSoftware. Bonita website:http://www.bonitasoft.com/.

[8] Y.-M. Chen, W.-S. Shir, and C.-Y. Shen. Distributed engineering change management for allied concurrent engineering. *Int J Comput Integr Manuf*, 15(2):127–151, 2002.

[9] M. P. Cota, D. Riesco, I. Lee, N. Debnath, and G. Montejano. Transformations from spem work sequences to bpmn sequence flows for the automation of software development process. *J. Comp. Methods in Sci. and Eng.*, 10(1-2S1):61–72, Sept. 2010.

[10] W. Fdhila, C. Indiono, S. Rinderle-Ma, and M. Reichert. Dealing with change in process choreographies: Design and implementation of propagation algorithms. *Information Systems*, 49:1–24, April 2015.

[11] G. Grambow, R. Oberhauser, and M. Reichert. Towards automatic process-aware coordination in collaborative software engineering. In *6th Int'l Conference on Software and Data Technologies (ICSOFT'11)*, pages 5–14. SciTePress, July 2011.

[12] T. I. Grantham-Lough K, Stone MC. Prescribing and implementing the risk in early design (red) method. 2006.

[13] V. Gruhn. Process-centered software engineering environments, a brief history and future challenges. *Ann. Softw. Eng.*, 14(1-4):363–382, Dec. 2002.

[14] M. Hajmoosaei. Impact analysis of workflow adaptation at run-time. Master thesis, University of Grenoble, 2014.

[15] T. Hoff. Neo4j - a graph database that kicks buttox at http://highscalability.com.

[16] G. Huang and K. Mak. Computer aids for engineering change control. *Journal of Materials Processing Technology*, 76:187 – 191, 1998.

[17] IBM. Rational team concert : http://www.ibm.com/developerworks/downloads/r/rtc/.

[18] T. A. W. Jarratt, C. M. Eckert, P. J. Clarkson, and L. Schwankl. Product architecture and the propagation of engineering change. In *7th International Design Conference (Design 2002)*, pages 75–80, 2002.

[19] T. A. W. Jarrett, C. M. Eckert, N. H. M. Caldwell, and P. J. Clarkson. Engineering change: an overview and perspective

on the literature. *Research in Engineering Design*, 22(2):103–124, April 2011.

[20] JBoss. jbpm website : http://www.jbpm.org.

[21] S. F. KöNigs, G. Beier, A. Figge, and R. Stark. Traceability in systems engineering - review of industrial practices, state-of-the-art technologies and new research solutions. *Adv. Eng. Inform.*, 26(4):924–940, Oct. 2012.

[22] O. Kovalenko, D. Winkler, M. Kalinowski, E. Serral Asensio, and S. Biffl. Engineering process improvement in heterogeneous multi-disciplinary environments with the defect causal analysis. In *Proceedings of the 21th EuroSPI Conference on Systems Software and Service Process Improvement, Communication in Computer and Information Science*, pages 73–85. Springer, 2014.

[23] H. J. Lee, H. J. Ahn, J. W. Kim, and S. J. Park. Capturing and reusing knowledge in engineering change management: A case of automobile development. *Information Systems Frontiers*, 8(5):375–394, 2006.

[24] R. Matinnejad and R. Ramsin. An analytical review of process-centered software engineering environments. In *IEEE 19th International Conference and Workshops on Engineering of Computer-Based Systems, ECBS 2012, Novi Sad, Serbia, April 11-13, 2012*, pages 64–73, 2012.

[25] M. Minor, R. Bergmann, S. Görg, and K. Walter. Towards case-based adaptation of workflows. In *Proceedings of the 18th International Conference on Case-Based Reasoning Research and Development*, ICCBR'10, pages 421–435, Berlin, Heidelberg, 2010. Springer-Verlag.

[26] M. Minor, D. Schmalen, Koldehoff, and R. Bergmann. Structural Adaptation of Workflows Supported by a Suspension Mechanism and by Case-Based Reasoning. In *Proceedings of WETICE 2007*, pages 370–375, 2007.

[27] H. Mojtaba, T. Hanh Nhi, P. Christian, F. Agnes, and R. Claudia. Impact analysis of process change at run-time. In *Proceedings of The 24th IEEE International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (to be appeared)*, WETICE 2015, 2015.

[28] R. Muller, U. Greiner, and E. Rahm. Agent work: a workflow system supporting rule-based workflow adaptation. *Data Knowledge Engineering*, pages 223–256, 2002.

[29] R. Muller and E. Rahm. Dealing with logical failures for collaborating workflows. In O. Etzion and P. Scheuermann, editors, *CoopIS*, volume 1901 of *Lecture Notes in Computer Science*, pages 210–223. Springer, 2000.

[30] B. Mutschler, M. Reichert, and J. Bumiller. Unleashing the effectiveness of process-oriented information systems: Problem analysis, critical success factors, and implications. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 38(3):280–291, May 2008.

[31] Neo4j. Neo4j website : http://www.neo4j.com.

[32] Oasis. Web services business process execution language version 2.0, 2007.

[33] S. Oh, B. Park, S. Park, and Y. S. Hong. Design of change-absorbing system architecture for the design of robust products and services. In J. A. Jacko, editor, *HCI (4)*, volume 4553 of *Lecture Notes in Computer Science*, pages 1110–1119. Springer, 2007.

[34] O. M. G. (OMG). Software Process Engineering Metamodel (SPEM), 2007.

[35] O. M. G. (OMG). Business process model and notation (bpmn) version 2.0, jan 2011.

[36] C. Portela, A. Vasconcelos, S. Oliveira, A. A. Silva, and S. Elder. Spider-pe: A set of support tools to software process enactment. In *Proceedings of the 9th International Conference on Software Engineering Advances*, ICSEA'14, 2014.

[37] M. Reichert, S. Rinderle, and P. Dadam. Adept workflow management system: Flexible support for enterprise-wide business processes. In *Proceedings of the 2003 International Conference on Business Process Management*, BPM'03, pages 370–379, Berlin, Heidelberg, 2003.

[38] M. Reichert and B. Weber. *Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies.* Springer, 2012.

[39] I. Robinson, J. Webber, and E. Eifrem. *Graph Databases*. O'Reilly Media, Inc, 2013.

[40] Siemens. Teamcenter website:http://www.plm.automation.siemens.com/.

[41] W. M. P. Van Der Aalst, A. H. M. T. Hofstede, and M. Weske. Business process management: A survey. In *Proceedings of the 2003 International Conference on Business Process Management*, BPM'03, pages 1–12, Berlin, Heidelberg, 2003. Springer-Verlag.

[42] F. Yuan and L. Ming-Shu. Towards Software Process Enactment Based on the SPEM2XPDL Model Transformation. *Journal of Software*, 18, 2007.