A Hybrid Approach for Business Environment-Aware Management of Service-Based Business Processes

Olfa Bouchaala[1,2], Samir Tata[1], and Mohamed Jmaiel[2]

[1] Insitut Mines-Telecom, TELECOM SudParis, CNRS UMR Samovar, Evry, France
{olfa.bouchaala,samir.tata}@it-sudparis.eu
[2] University of Sfax, ReDCAD Laboratory, Tunisia
mohamed.jmaiel@enis.rnu.tn

Abstract. Enterprises struggle to find a balance between adapting their business processes (BPs) against business environments and keeping competitiveness. Indeed, while the imperative nature of monolithic BPs is too rigid to adapt them at runtime, the declarative one of the purely rule-based BPs is very time-consuming. Therefore, in this paper, we focus on business environment-aware management of service-based business processes (SBPs) aiming at conciliating imperative and declarative techniques. Our challenge is to develop a hybrid management approach that (1) preserves standards to describe SBPs, (2) keeps non-dependency to a specific BP engine and (3) minimizes designers efforts. Based on a semantic modeling, we are able to synthesize a controller, itself modeled as a BP, connected to the BP to be monitored and configured. Using our approach does not impact any existing business process management system since controllers are BPs that can be deployed and enacted along with the managed processes.

1 Introduction

Business processes (BPs) represent a key concept for automating enterprises' activities. As enterprises encounter highly dynamic business environments, there is a great need for business process management (BPM) at run-time. By dealing with competitive and constantly changing business environments, enterprises' policies change frequently. Thus, they need to focus on adapting their processes from a business environment point of view. The business environment connotes all factors external to the enterprise and that greatly influence its functioning. It covers many factors such as economic, social ones (*e.g.* festive season).

Business environment-aware management (BEAM for short) [1,2,3,4] of BPs consists in configuring them in order to change their behaviors in reaction to business environment events (*e.g.* during a sales promotion, there is a decrease in clothes prices). There are two types of approaches of BEAM: imperative and declarative. Declarative approaches are based on ECA-rules [1,5] which are flexible, since they are well adapted for adding, removing and changing rules at runtime. Nevertheless, they are inefficient since they are time consuming because of inference change in the business environment. In addition, they may not adopt standard notations for BP modeling such as BPMN or BPEL. On the other hand, imperative approaches consist in hard coding management actions into the BP. Consequently, they preserve standard notation for BP modeling and are very efficient, in terms of execution time. Nevertheless, they are too rigid due to over-specifying processes at design time.

An example depicting an online purchase order process of a clothing store is represented in Fig. 1. Upon receipt of customer order, the seller checks product availability. If some of the products are not in stock, the alternative branch "ordering from suppliers" is executed. When all products are available, the choice of a shipper and the calculation of the initial price of the order are launched. Afterwards, the shipping price and the retouch price are computed simultaneously. The total price is then computed in order to send invoice and deliver the order. During a sales promotion, a discount rule should be added and the relationships with the existing rules ought to be established [6]. In addition, they may require that BPs, to be monitored, are also described in terms of rules rather than standards such as BPEL and BPMN. On the contrary, imperative approaches require over-specifying processes by predicting all possible events.

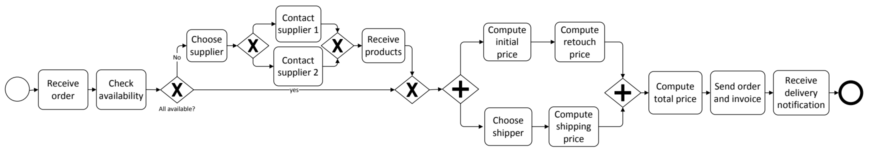


Fig. 1. Purchase order process

Given these limitations, in this paper, we address business environment-aware management of SBPs that mainly raises the following questions.

- How to conciliate between imperative and declarative techniques in an integrated hybrid approach aiming to strengthen their advantages?
- How to develop a hybrid management approach that (1) preserves industry standards to describe SBPs, (2) keeps non-dependency to a specific business process engine and (3) minimizes designers efforts?

In order to address these challenges, our approach models the management of an SBP as a process connected to it for its monitoring and configuration. Monitoring reads properties of services that compose the SBP while configuration alters values of these properties. Contrary to the imperative approaches, in our approach, the management process defines several management paths. Therefore, it can encapsulate different management behaviors. The choice of a management path is based on events of the business environment which are semantically described. Consequently, our approach presents a degree of flexibility inherited from declarative approaches.

The rest of this paper is organized as follows. Section 2 gives an overview of our technique for generating a management process as well as the required semantic modeling of SBPs, business environments and relationships between them. Based on this model, section 3 describes an algorithm enabling the management process construction. Then, section 4 presents the implementation and proves our concepts. In section 5, we present a literature review of business environment-aware management approaches. Finally, section 6 summarizes the main findings of our work and presents future directions.

2 A Hybrid Approach for BEAM of SBPS

2.1 Approach Overview

In our work, we consider that the management of a composition of services offering management operations is realized through the composition of the offered management operations. The enactments of management operations are triggered by events that are captured from the business environment. The composition of management operations and the business environment events constitute a BP that manages the original SBP. Fig. 2 illustrates the purchase order process and its corresponding generated management process. The management process uses the management operations to monitor and configure the original SBP.

In fact, in order to take into account the business environment changes into the managed SBP, we use service properties that are adjusted. Indeed, service properties allow for the configuration of an implementation with externally set values. The value for a service property is supplied to the implementation of the service each time the implementation is executed. In particular, the internal value of a property can be altered at any time through business management operations which enable its monitoring and configuration. The monitoring step reads properties while the configuration one updates them if necessary. When changing a property value, the corresponding service changes its behavior. For example, the service "Compute initial price" of Fig. 2 has a property named "Discount rate" which can change its behavior by a setter operation when a sales promotion is triggered. As we already mentioned, when changing a service behavior, the BP is reconfigured and its behavior is accordingly modified.

Thus, the first step towards the automation of the managing operations composition is to identify the semantic concepts of the services properties from the initial BP. The issue is how to modify these properties and in which order. To deal with this issue, we adopted a three-phase methodology:

- **Phase 1:** Events represent the glue between business environments and SBPs. Hence, events may trigger the update of service properties.
- **Phase 2:** Service properties may depend on each others. Accordingly, modifying a service property may engender changes on others depending on it.
- **Phase 3:** The structure of the initial BP gives an idea on the order of management operations that modify properties.

Consequently, this methodology requires appropriate semantic model. Therefore, we propose an upper management ontology which correlated with a domain ontology represents a declarative description of the company management strategy against dynamic business environment (section 2.2).

Based on the upper management ontology and the structure of the initial SBP, we design an algorithm for generating the management process (Section 3).

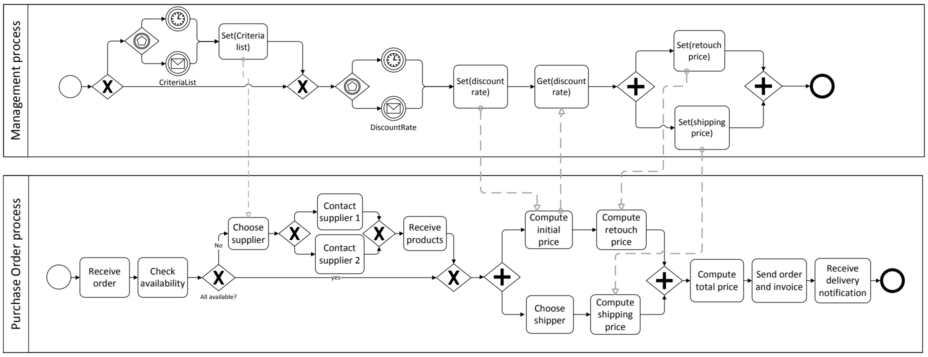


Fig. 2. Purchase order process with its corresponding management process

2.2 Semantic Modeling of SBPs and Business Environment

As shown in Fig. 3, there are three main actors in BEAM: business environment, BP and services. The BP has a service composition which is composed of activities and gateways. Activities are realized by services. Each service has a service property and management operations. Services interact with the business environment. This latter engenders **events** that trigger **management operations** which act in turn on **service properties**. These three concepts represented in grey ellipses in Fig. 3 represent the main concepts of the management ontology at a high level of abstraction.

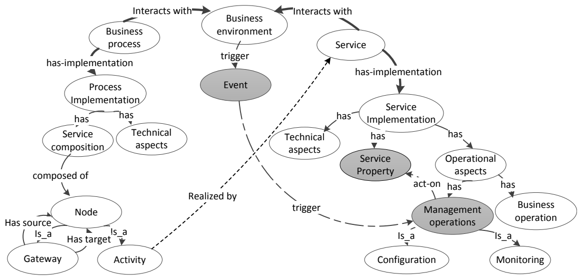


Fig. 3. Actors in the BEAM

Upper Management Ontology. Services properties, management operations and business environment events are described against a domain ontology. Such ontology is defined by domain experts. To facilitate the management process construction, we define an upper management ontology (Fig. 4). This ontology represents two main relationships:

- **Environment-Service relationship:** Events trigger Actions (management operations) which act on services properties.
- **Service-Service relationship:** Each service property has services properties that may depend on it.

Events play a prominent role in BEAM, since they are the glue between situations in the real world and SBPs. Thus, to be in line with standards, we base events semantics and definitions on the expressiveness of BPMN 2.0 [7]. Events are used to model something happening in the process lifetime. They affect the flow of the process by catching a trigger or throwing a result. Event definitions represent the semantics of events. In BPMN 2.0, there are 10 event definitions among them we use: Message, Signal, Timer and Conditional. Timer and Conditional events are implicitly thrown. When they are activated they wait for a time based or status based condition respectively to trigger the catch event (*e.g.* a timer event definition is used to detect that the promotion time date is reached). Some events (e.g. Message, Signal) have the capability to carry data (*e.g.* a Message event is used to define a Discount rate message in order to carry the discount Rate information).

The Structure of the Initial SBP. In SBPs, activities are realized by services. In this work, a semantic service S is mainly characterized by its property p, which, being adjusted, changes the service behavior. A service property has a name, a value and is annotated with a concept from the domain ontology. The initial SBP is a finite set of nodes representing activities, gateways and possibly events described using a BP standard (e.g. EPC, BPEL, BPMN, etc). Abstracting SBPs using graphs renders the management process generation possible for any BP standard language. Thus, SBPs and its corresponding management process are modeled using graphs. Each vertex/edge is annotated with a pair indicating the vertex/edge type and the vertex/edge label. As stated earlier, the available types of vertices depend on the adopted BP standard notation. In this paper, we consider the BPMN notation which distinguishes between activities ('a'), gateways ('g') and events ('e'). There are also different types of BPMN gateways and events. The activity name, the gateway type and the event type represent possible vertex labels (e.g. ('a', 'receive order'), ('g','AND-split'), ('e', 'start event')). The edge types are deduced from the control dependency graph of the BPMN process. The control dependency graph of the purchase order process (Fig. 1) is generated inspiring from [8,9] (Fig. 5).

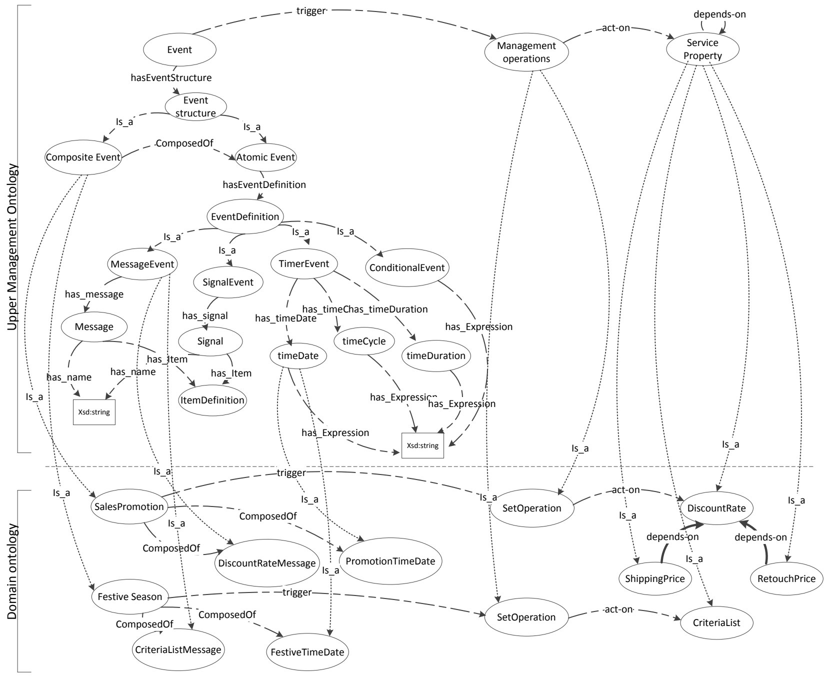


Fig. 4. Purchase order ontology

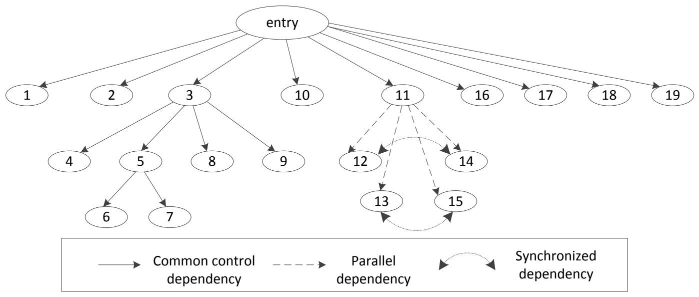


Fig. 5. Control dependency graph of the purchase order process (1: Receive order, 2: Check availability, 3: OR-split, 4: Choose supplier, 5: OR-split, 6: Contact supplier 1, 7: Contact supplier 2, 8: OR-join, 9: Receive products, 10: OR-join, 11: AND-split, 12: Compute initial price, 13: Compute retouch price, 14: Choose shipper, 15: Compute shipping price, 16: AND-join, 17: Compute total price, 18: Send order and invoice, 19: Receive delivery notification)

3 Generating Management Process

In the following, we define a management process to handle an SBP during its execution. We recall that services' properties frequently change due to business environment changes. When a new event in the business environment occurs, the adequate properties should be updated. Therefore, the management process consists in a composition of management operations that read and/or alter services' properties. At this level, we define a getter and a setter for each property.

The construction of the management solution (composition of management operations) is performed using semantic descriptions over domain ontology as well as the structure of the initial BP. Thereby, the construction of the composition comprises three main phases: (1) constructing sub-processes based on the Environment-Service relationship, (2) constructing sub-processes based on the Service-Service relationship and (3) connecting generated sub-processes.

Properties externalize the service behavior. Thus, the first step towards the automation of the managing operations composition is to capture the semantic concepts of the services properties from the initial BP. Each service property can have possible events that trigger the update of its value. Thus, Algorithm 1 is called with p as parameter for building sub-processes relating configuration operations with events (see section 3.1). Configuring a service property may engender the update of other properties related to it. Algorithm 2 is called in turn to build a sub-process connecting management operations with each other (see section 3.2). Finally, Algorithm 3 is performed to connect resulting subprocesses based on the structure of the initial SBP (see section 3.3).

3.1 Constructing Sub-processes Based on Environment-Service Relationship

In this first phase, the issue is to alter a service property based on the Environment-Service relationship introduced in section 2.2. Indeed, when an event occurs the corresponding service property is updated according to dependencies between business environment events and services. In accordance with the running example, when a "Sales promotion" happens, there is a decrease in clothes prices. Subsequently, the property named "Discount Rate" is altered. As stated in section 2.2, the event "Sales promotion" is composed of atomic events having event definitions: DiscountRateMessage and PromotionTimeDate.

In order to create subprocesses aiming at modifying a service property, Algorithm 1 is performed. These subprocesses relate a service management operation with possible events that can trigger it. Fig. 6(b) is the resulting subprocess for p="DiscountRate". Similarly, with p="CriteriaList" the subprocess described in Fig. 6(a) is generated.

The list of possible events as well as their definitions result from calling the procedure *FindEvents(p)* that executes the following SPARQL query (Line 1): "SELECT ?atomicEvent ?eventdefinition WHERE { ?event ns:trigger ?action. ?action ns:act-on ?property. ?property rdf:type ns:"+p+". ?event ns:hasEventstructure

?events. ?events ns:composedOf ?atomicEvent. ?atomicEvent ns:hasEventDefinition ?definition. ?definition rdf:type ?eventdefinition.}".

When an event occurs, the service property p will be altered automatically using a set operation. A vertex $("a", "set(p)")$ is added to the vertex-set of the managing graph MG (Line 3). If the list of possible events that can modify the property comprises only one event, we add this event to the set of vertices of MG graph (Line 5). A single edge between the event and the "set" operation is also added (Line 6). Otherwise, a node of gateway type labeled "*Event-based XOR*" is added (Line 8). Then, a node for each event and edges relating it to the gateway as well as the set operation are identified (Line 10, 11, 12).

Algorithm 1. *ConstructESR(ServiceProperty p, Managing Graph MG)*

```
Require: Managing Graph $MG$
Ensure: Managing Graph $MG$
 1: List $L_1 \longleftarrow$ FindEvents(p)
 2: if $L_1 \neq \emptyset$ then
 3:     $V_3(MG) \longleftarrow V_3(MG) \cup \{("a", "set(p)")\}$
 4:     if $L_1 = \{l_1\}$ then
 5:         $V_3(MG) \longleftarrow V_3(MG) \cup \{("e", "l_1")\}$
 6:         $E_3(MG) \longleftarrow E_3(MG) \cup \{(("e", "l_1"), ("a", "set(p)"))\}$
 7:     else
 8:         $V_3(MG) \longleftarrow V_3(MG) \cup \{("g", "Event-basedXOR")\}$
 9:         for all $l_1 \in L_1$ do
10:             $V_3(MG) \longleftarrow V_3(MG) \cup \{("e", "l_1")\}$
11:             $E_3(MG) \longleftarrow E_3(MG) \cup \{(("g", "Event-basedXOR"), ("e", "l_1"))\}$
12:             $E_3(MG) \longleftarrow E_3(MG) \cup \{(("e", "l_1"), ("a", "set(p)"))\}$
13:         end for
14:     end if
15: end if
16: return $MG$
```

3.2 Constructing Sub-processes Based on Service-Service Relationship

A service property may depend on others. Hence, updating a service property may engender the modification of others depending on it. Therefore, in this second phase, the concern is to properly identify the semantic relationship holding between service properties. For instance, the service properties named "Shipping Price" and "Retouch price" depend on "Discount Rate" property (Fig. 4). Thus, if the property "Discount Rate" is updated, both "Shipping price" and "Retouch price" properties should be updated. The corresponding resulting subprocess is depicted in Fig. 6(c).

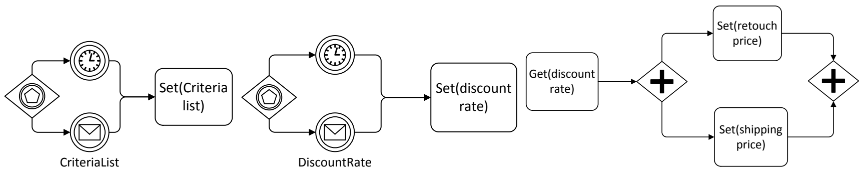
In order to generate this subprocess, Algorithm 2 explores the different dependency relationships between concepts of services' properties from the domain ontology. Two services properties have a relationship if they are related with "depends-on" relationship in the domain ontology. A SPARQL query is then sent to the domain ontology to enquire for the sources of the property p :

"SELECT ?sourceType WHERE ?source ns:depends-on ?a. ?a rdf:type ns:"+p+". ?source rdf:type ?sourceType. ?sourceType rdfs:subClassOf ns:ServiceProperty.".

The result of this query is performed by calling the procedure *ServiceSourceOfDepends-On(p)* (Line 1). If p has properties that depend on it (Line 2), then the $get(p)$ operation is automatically invoked (Line 3). As a result, a setter for each property depending on p is defined (Line 4-6). If there is only one property, then a simple edge links its setter with $get(p)$. Otherwise, the adequate gateway relating properties setters with $get(p)$ is identified based on the control dependency graph (Fig. 5). For example, the services "Compute retouch price" and "Compute shipping price" are synchronized according to the control dependency graph of the purchase order process. Therefore, a gateway labeled ('g','AND-Split) is added. As for a well structured BP, when starting with a gateway type, we finish by the same (Line 13, 16).

Algorithm 2. *ConstructSSR(ServiceProperty p, Managing Graph MG)*

```
Require: Managing Graph $MG$
Ensure: Managing Graph $MG$
 1: List $L_2 \longleftarrow$ ServiceSourceOfDepends-On(p)
 2: if $L_2 \neq \emptyset$ then
 3:     $V_3(MG) \longleftarrow V_3(MG) \cup \{("a", "get(p)")\}$
 4:     for all $l \in L_2$ do
 5:         $V_3(MG) \longleftarrow V_3(MG) \cup \{("a", "set(l)")\}$
 6:     end for
 7:     if $L_2 = \{l_2\}$ then
 8:         $E_3(MG) \longleftarrow E_3(MG) \cup \{(("a", "get(p)"), ("a", "set(l_2)"))\}$
 9:     else
10:         String GatewayType=$ChooseGateway(L_2, p)$
11:         $V_3(MG) \longleftarrow V_3(MG) \cup \{("g", GatewayType)\}$
12:         $E_3(MG) \longleftarrow E_3(MG) \cup \{(("a", "get(p)"), ("g", GatewayType))\}$
13:         $V_3(MG) \longleftarrow V_3(MG) \cup \{("g", GatewayType)\}$
14:         for all $l_2 \in L_2$ do
15:             $E_3(MG) \longleftarrow E_3(MG) \cup \{(("g", GatewayType), ("a", "set(l_2)"))\}$
16:             $E_3(MG) \longleftarrow E_3(MG) \cup \{(("a", "set(l_2)"), ("g", GatewayType))\}$
17:         end for
18:     end if
19: end if
20: return $MG$
```



(a) Result of phase 1 for p="CriteriaList" (b) Result of phase 1 for p="DiscountRate" (c) Result of phase 2 for p="DiscountRate"

Fig. 6. Result of phase 1 and phase 2

3.3 Connecting Generated Sub-processes

So far, a set of sub-processes are created. Indeed, for each property sub-processes based on the Environment-Service and/or Service-Service relationship are built. How to connect their ends? How to determine their order?

Resuming with the running example, till now, three sub-processes are built (see Fig. 6). In order to connect them aiming to generate the whole management process (Fig. 2), in this phase, we add missing links and gateways based on the explicit semantic description of the initial BP. Doing so, we adopted the following steps: (1) identifying management process ends, (2) capturing their corresponding in the initial BP, (3) determining control dependencies for each activity in order to add corresponding gateways and (4) organizing results based on the control flow of the initial BP.

Algorithm 3 formalizes these steps as follows. The first step consists in finding nodes having no targets (set operations) and nodes having no sources (get operations) (Line 2). The second step is to identify nodes corresponding to these activities having p as property in the process graph (Line 3). Afterwards, the control dependency of each node is determined (Line 5). Control dependencies for each node are then compared in order to identify the existence or not of control dependency between subprocesses (Line 12-16). Then, with respect to control-flow relations between activities in the process graph, the subprocesses are organized and control flow edges are added to the managing graph.

As a final step, nodes which have no sources are linked to the start event (Line 19, 20). In addition, nodes having no targets are connected to the end event (Line 22, 23).

Algorithm 3. *ConnectSP(Process Graph PG, Managing Graph MG)*

```
Require: Managing Graph $MG$, Process Graph $PG$
Ensure: Managing Graph $MG$
 1: for all $v \in V_3(MG)$ do
 2:   if $(S(v) = \emptyset \wedge MG.\tau_3(v) =' a') \vee (T(v) = \emptyset \wedge MG.\tau_3(v) =' a')$ then
 3:     Find $v_1$ in $V_1(PG)$ such that $v_1.p.concept = MG.\theta_3(v)$
 4:     $v_2 \longleftarrow searchControldependencies(CDG, v_1)$
 5:     if $PG.\omega_1((v_1, v_2)) = "commoncontrol - dependency"$ then
 6:       $V_3(MG) \longleftarrow V_3(MG) \cup \{('g', "OR - Split")\}$
 7:       $E_3(MG) \longleftarrow E_3(MG) \cup \{(('g', "OR - Split"), PG.\theta_1(v_1)\}$
 8:     end if
 9:     $Map \longleftarrow Map \cup (v_1, v_2)$
10:   end if
11: end for
12: if $\bigcap\{Map(i)\} = \varnothing$ then
13:   $V_3(MG) \longleftarrow V_3(MG) \cup \{('g', "OR - Join")\}$
14:   $E_3(MG) \longleftarrow E_3(MG) \cup \{PG.\theta_1(v_1), ('g', "OR - Join")\}$
15:   $E_3(MG) \longleftarrow E_3(MG) \cup \{('g', "OR - Split"), ('g', "OR - Join")\}$
16: end if
17: $V_3(MG) \longleftarrow V_3(MG) \cup \{(("e", "Startevent"), ("e", "Endevent"))\}$
18: for all $v \in V_3(MG)$ do
19:   if $S(v) = \emptyset$ then
20:     $E_3(MG) \longleftarrow E_3(MG) \cup \{(("e", "Startevent"), MG.\theta_3(v))\}$
21:   end if
22:   if $T(v) = \emptyset$ then
23:     $E_3(MG) \longleftarrow E_3(MG) \cup \{(MG.\theta_3(v), ("e", "Endevent"))\}$
24:   end if
25: end for
26: return $MG$
```

4 Implementation

As a proof of concept, we have implemented a business environment-aware management framework called BEAM4SBP. BEAM4SBP is a java library that intends to generate a management process connected to an initial business process allowing for its monitoring and configuration. The architecture and implementation details about BEAM4SBP can be found at: `http://www-inf.int-evry.fr/SIMBAD/tools/BEAM4SBP`.

5 Related Work

As business environment changes keep increasing, enterprises are always seeking for a balanced solution to manage their processes. However, most research has focused on efficiency or flexibility using either imperative or declarative techniques. Therefore, different approaches [2,10,3,4] try to integrate these two techniques in a joint approach by separating business logic (described by Business rules) and process logic (described by imperative BP).

Charfi et al. [11] focus on Aspect Oriented Programming in order to integrate Business rules and the process logic at run-time. Indeed, the business rules are proposed to be implemented in an aspect-oriented extension of BPEL called AO4BPEL. AO4BPEL is used to weave the adaptation aspects into the process at run-time. Although they preserve BP standards, the weaving phase can strongly limit the process efficiency at run-time since it can raises issues on maintainability and transformation. On the contrary, our management process is generated at deployment time and hence at run-time the managing process is connected to the managed process creating an imperative and efficient process.

Other approaches, such as [2] and [10], address management issue by process variants. When modeling process and their variants, one has to decide which control flow alternatives are variant-specific and which ones are common for all process variants. However, these process variants ought to be configured at configuration time which leads to a static instance of the process model at run-time. While in our case, the values of services properties are altered at run-time taking into account changes in the business environment.

Authors in [10], present an adaptation of BPEL language called VxBPEL. They emphasize on the lack of flexibility and variability when deploying BPEL processes. Thus, they propose to extend BPEL language by adding `Variation Points` and `Variants`. The former represents the places where the process can be configured, while the latter defines the alternative steps of the process that can be used. In this work, the variability is focused on BP aspects written in VxBPEL language. The designers should consider this extension and add their variation when designing the BP. However, in our work, variability are integrated in services and the process designer will not re-write its process.

Ouyang et al. [3] introduce an ECA-based control-rule formalism to modularize the monolithic BPEL process structure. Only one classification of rules is defined that handle the control flow part of the composition linking activities together. In this work, the designer should also take into account the defined ECA-control rule and specify its process accordingly.

6 Conclusion

In this paper, we proposed a novel hybrid approach for managing SBPs against highly dynamic business environments. This approach conciliate between imperative and declarative techniques while addressing the following issues: preserving standards for describing SBPs, minimizing designers efforts and non-dependency to a specific Business process engine. Our approach consists in generating, at deployment time, a management process for an initial SBP connected to it allowing its monitoring and configuring. The management process generation is performed thanks to a semantic model. This semantic model involves an upper management ontology, describing relationship between SBPs and business environments, and an explicit semantic description of the initial BP. This latter is based on identifying control dependencies to facilitate the organization of the whole management process.

However, data dependencies are important in turn to identify other aspects when creating the management process. Thereby, we are working on explicitly defining semantic data dependencies between inputs, outputs and properties which can include other service properties relationships such as mutuality and exclusivity. Hence, as part of our short term perspective, we foresee to detail more the service properties relationships in the upper management ontology. In addition, at this level, our approach involves only getters and setters as managing operations. Thus, we plan to specify a composition for managing operations given by the service provider.

References

1. Boukhebouze, M., Amghar, Y., Benharkat, A.-N., Maamar, Z.: A rule-based modeling for the description of flexible and self-healing business processes. In: Grundspenkis, J., Morzy, T., Vossen, G. (eds.) ADBIS 2009. LNCS, vol. 5739, pp. 15–27. Springer, Heidelberg (2009)
2. Gottschalk, F., van der Aalst, W.M.P., Jansen-Vullers, M.H., Rosa, M.L.: Configurable workflow models. Int. J. Cooperative Inf. Syst. 17(2), 177–221 (2008)
3. Ouyang, B., Zhong, F., Liu, H.: An eca-based control-rule formalism for the bpel process modularization. Procedia Environmental Sciences 11(1), 511–517 (2011)
4. Gong, Y., Janssen, M.: Creating dynamic business processes using semantic web services and business rules. In: ICEGOV, pp. 249–258 (2011)
5. Weigand, H., van den Heuvel, W.J., Hiel, M.: Business policy compliance in service-oriented systems. Inf. Syst. 36(4), 791–807 (2011)
6. Boukhebouze, M., Amghar, Y., Benharkat, A.N., Maamar, Z.: Towards an approach for estimating impact of changes on business processes. In: CEC (2009)
7. Business process model and notation 2.0, `http://www.omg.org/spec/BPMN/2.0/`
8. Ferrante, J., Ottenstein, K.J., Warren, J.D.: The program dependence graph and its use in optimization. ACM Trans. Program. Lang. Syst. 9(3), 319–349 (1987)
9. Mao, C.: Slicing web service-based software. In: SOCA, pp. 1–8 (2009)
10. Koning, M., Ai Sun, C., Sinnema, M., Avgeriou, P.: Vxbpel: Supporting variability for web services in bpel. Information & Software Technology 51(2), 258–269 (2009)
11. Charfi, A., Dinkelaker, T., Mezini, M.: A plug-in architecture for self-adaptive web service compositions. In: ICWS, pp. 35–42 (2009)