Checking Business Process Evolution

Pascal Poizat[1,2], Gwen Salaün[3], and Ajay Krishna[3]

[1] Université Paris Lumières, Univ Paris Ouest, Nanterre, France
[2] Sorbonne Universités, UPMC Univ Paris 06, CNRS, LIP6 UMR7606, Paris, France
[3] University of Grenoble Alpes, Inria, LIG, CNRS, France

Abstract. Business processes support the modeling and the implementation of software as workflows of local and inter-process activities. Taking over structuring and composition, evolution has become a central concern in software development. We advocate it should be taken into account as soon as the modeling of business processes, which can thereafter be made executable using process engines or model-to-code transformations. We show here that business process evolution needs formal analysis in order to compare different versions of processes, identify precisely the differences between them, and ensure the desired consistency. To reach this objective, we first present a model transformation from the BPMN standard notation to the LNT process algebra. We then propose a set of relations for comparing business processes at the formal model level. With reference to related work, we propose a richer set of comparison primitives supporting renaming, refinement, property- and context-awareness. Thanks to an implementation of our approach that can be used through a Web application, we put the checking of evolution within the reach of business process designers.

1 Introduction

Context. A business process is a structured set of activities, or tasks, that is used to create some product or perform some service. BPMN [24,16] has become the standard notation for business processes. It allows one to model these processes as sequences of tasks, but also as more complex workflows using different kinds of gateways. There are now plenty of frameworks supporting BPMN modeling, *e.g.*, Activiti, Bonita BPM, or the Eclipse BPMN Designer. Most of them accept BPMN 2.0 (BPMN for short in the rest of this paper) as input and enable one to execute it using business process engines.

Modern software exhibits a high degree of dynamicity and is subject to continuous evolution. This is the case in areas such as autonomic computing, pervasive or self-adaptive systems, where parts of the system components may have to be removed or added in reaction to some stimulus. This is also the case for more mainstream software, *e.g.*, when developed using an agile method.

In this paper, we focus on software development based on BPMN. We suppose some application has been developed from a BPMN model and in order to evolve this application one wants first to evolve the BPMN model. This is sensible to keep the application and the model consistent, either for documentation purposes

or because one follows a model at runtime approach (executing business processes on process engines being a specific case of this).

Objective. Given two BPMN business processes, we want to support the (human) process designer in the evolution activity with automated verification techniques to check whether the evolved process satisfies desired properties with reference to the original process version. The designer should have different kinds of verifications at hand for different kinds of evolutions one can perform on a business process. These verifications will enable the designer to understand the impact of evolution and, if necessary, support the refinement of an incorrect evolution into a correct one.

Approach. Since BPMN has only an informal semantics, we have first to define a model transformation into a formal model that could ground the different needed verifications. For this we choose to transform BPMN processes into LNT [4] process algebraic descriptions. Using the LNT operational semantics, we can retrieve Labelled Transition Systems (LTSs) which are a formal model with rich tool support. Then we define a set of atomic evolution verifications based on LTS equivalences and LTS pre-orders originating from concurrency theory. These can be applied iteratively to get feed-back on the correctness of evolutions and perform changes on business processes until satisfaction. Our approach is completely automated in a tool we have developed, VBPMN, that designers may use through a Web application to check process evolution [1]. It includes an implementation of our BPMN to LNT transformation, and relies on a state-of-the-art verification tool-box, CADP [13], for computing LTS models from LNT descriptions and for performing LTS-level operations and atomic analysis actions used in our evolution verification techniques. We have applied our approach and tool support to many examples for validation purposes. Thanks to the use of a modular architecture in VBPMN, other workflow-based notations, such as UML activity diagrams [20] or YAWL workflows [28], could be integrated to our framework.

Organization. Section 2 introduces BPMN and our running example. Section 3 presents the process algebra and the formal model transformation we use to give a translational semantics to BPMN process. We then build on this to present and formalize in Section 4 our different notions of business process evolution. Section 5 gives details on the implementation of our approach in a tool and discusses results of experiments performed with it. We present related work in Section 6 and we conclude in Section 7.

2 BPMN in a Nutshell

In this section, we give a short introduction on BPMN. We then present the running example we will use for illustration purposes in the rest of this paper.

BPMN is a workflow-based graphical notation (Fig. 1) for modeling business processes that can be made executable either using process engines (*e.g.*, Activiti, Bonita BPM, or jBPM) or using model transformations into executable

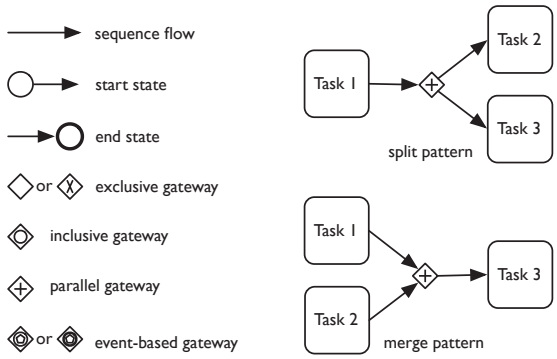


Fig. 1. BPMN Notation (Part of)

languages (*e.g.*, BPEL). BPMN is an ISO/IEC standard since 2013 but its semantics is only described informally in official documents [24,16]. Therefore, several attempts have been made for providing BPMN with a formal semantics, *e.g.*, [10,33,25,18,22]. In this paper, we abstract from the features of BPMN related to data and we focus on the core features of BPMN, that is, its control flow constructs, which is the subset of interest with respect to the properties we propose to formally analyze in this paper. More precisely, we consider the following categories of workflow nodes: *start* and *end event*, *tasks*, and *gateways*.

Start and end events are used to denote respectively the starting and the ending point of a process. A task is an abstraction of some activity and corresponds in practice, *e.g.*, to manual tasks, scripted tasks, or inter-process message-based communication. In our context, we use a unique general concept of task for all these possibilities. Start (end, resp.) events must have only one outgoing (incoming, resp.) flow, and tasks must have exactly one incoming and one outgoing flow. Gateways are used, along with sequence flows, to represent the control flow of the whole process and in particular the task execution ordering. There are five types of gateways in BPMN: *exclusive, inclusive, parallel, event-based* and *complex gateways*. We take into account all of them but for complex gateways, which are used to model complex synchronization behaviors especially based on data control. An exclusive gateway is used to choose one out of a set of mutually exclusive alternative incoming or outgoing branches. It can also be used to denote looping behaviors as in Figure 5. For an inclusive gateway, any number of branches among all its incoming or outgoing branches may be taken. A parallel gateway creates concurrent flows for all its outgoing branches or synchronizes concurrent flows for all its incoming branches. For an event-based gateway, it takes one of its outgoing branches based on events (message reception) or accepts one of its incoming branches. If a gateway has one incoming branch and multiple outgoing branches, it is called a *split* (gateway). Otherwise, it should have one outgoing branch and multiple incoming branches, and it is called a *merge* (gateway). In this paper, we assume processes exhibit a balanced struc-

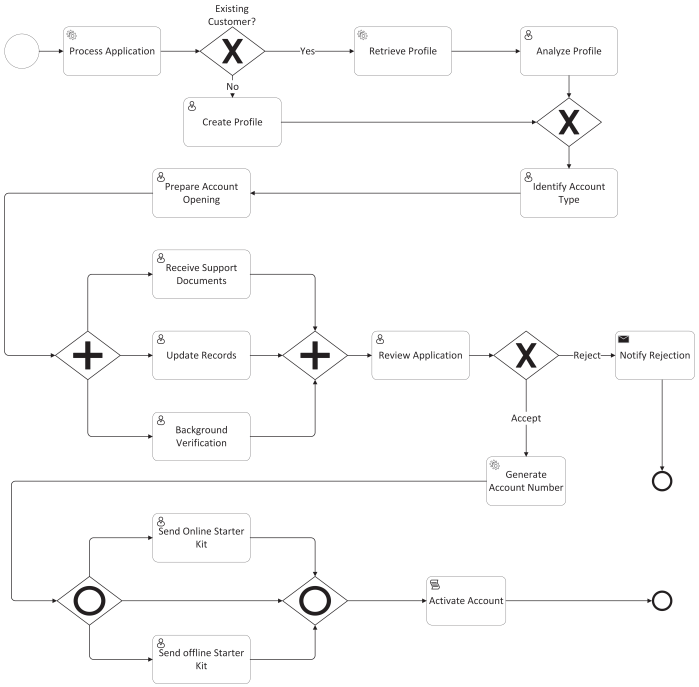


Fig. 2. Bank Account Opening Process in BPMN

ture with an exact correspondence between split and merge gateways, which is the case for most processes in practice. We also suppose that BPMN processes are syntactically correct. This can be enforced using a BPMN designer, *e.g.,* the Activiti BPM platform, Bonita BPM, or the Eclipse BPMN Designer.

Example. We use as running example the opening of a bank account depicted in Figure 2. This process starts by retrieving information about the customer (exclusive gateway, top part). Then, the type of account is identified and several documents need to be delivered (parallel gateway, middle part). Finally, the account creation is rejected or accepted, and in the latter case, some information is sent to the customer (inclusive gateway, bottom part) and the account is activated.

3 From BPMN to LTS

We present in this section a translational semantics from BPMN to LTSs, obtained through a model transformation from BPMN to the LNT process algebra, LNT being equipped with an LTS semantics.

3.1 LNT

LNT [4] is an extension of LOTOS, an ISO standardized process algebra [15], which allows the definition of data types, functions, and processes. LNT processes are built from actions, choices (select), parallel composition (par), looping behaviors (loop), conditions (if), and sequential composition (;). The communication between the process participants is carried out by rendezvous on a list of synchronized actions included in the parallel composition (par). We chose LNT first because it is expressive enough to encode the expressiveness of the BPMN constructs. Second, LNT operational semantics maps processes into LTSs. This opens the way thereafter to use the rich set of tools existing for LTSs, as those available in the CADP [13] toolbox, to implement the various checks presented in the sequel. LNT is preferred over the direct use of LTS (*i.e.*, the definition of a BPMN to LTS transformation) since this yields a simpler, high-level and declarative transformation.

3.2 From BPMN to LNT

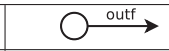
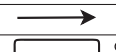
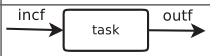
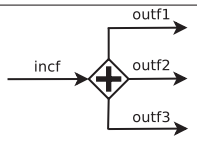
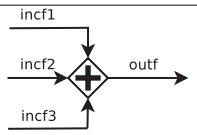
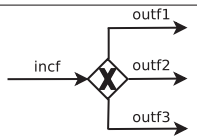
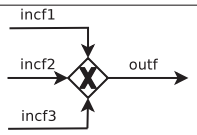
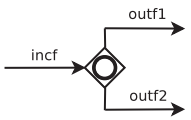
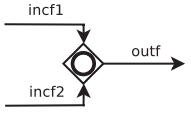
We present here the encoding into LNT of the BPMN constructs that we support. The main idea is to encode as LNT processes all BPMN elements involved in a process definition, that is, the nodes (tasks, gateways), which correspond to the behavior of the BPMN process, initial/end events, and sequence flows, which encode the execution semantics of the process. Finally, all these independent LNT processes are composed in parallel and synchronized in order to respect the BPMN execution semantics. For instance, after execution of a node, the corresponding LNT process synchronizes with the process encoding the outgoing flow, which then synchronizes with the process encoding the node appearing at the end of this flow, and so on.

Table 1 presents the encoding patterns for the main BPMN constructs. The actions corresponding to the flows (incf, outf, etc.) will be used as synchronization points between the different BPMN elements. The begin and finish actions in the initial/end events are just used to trigger and terminate, respectively, these events. The actions used in task constructs (*e.g.*, task) will be the only ones to appear in the final LTS. All other synchronizations actions will be hidden because they do not make sense from an observational point of view. Both the sequence flow and the task construct are enclosed within an LNT loop operator since these elements can be repeated several times if the BPMN process exhibits looping behaviors. We do not present the encoding of communication/interaction messages in Table 1 because they are translated similarly to tasks.

The parallel gateway is encoded using the par LNT operator, which corresponds in this case to an interleaving of all flows. The exclusive gateway is encoded using the select LNT operator, which corresponds to a nondeterministic choice among all flows. The event-based gateway is handled in the same way as the exclusive gateway, hence it is not presented here.

The semantics of inclusive gateways is quite intricate [5]. We assume here that each inclusive merge gateway has a corresponding inclusive split gateway

Table 1. Encoding Patterns in LNT for the Main BPMN Constructs

BPMN construct	BPMN notation	LNT encoding
Initial event		begin ; outf
End event		incf ; finish
Sequence flow		**loop** begin ; finish **end loop**
Task		**loop** incf ; task ; outf **end loop**
Parallel gateway (split)		incf ; **par** outf1 || outf2 || outf3 **end par**
Parallel gateway (merge)		**par** incf1 || incf2 || incf3 **end par** ; outf
Exclusive gateway (split)		incf ; **select** outf1 [] outf2 [] outf3 **end select**
Exclusive gateway (merge)		**select** incf1 [] incf2 [] incf3 **end select** ; outf
Inclusive gateway (split)		incf ; **select** (* s_i *if one matching merge* *) outf1 ; *s1* [] outf2 ; *s2* [] **par** outf1 || outf2 **end par** ; *s3* **end select**
Inclusive gateway (merge)		**select** (* s_i *if one matching split* *) *s1* ; incf1 [] *s2* ; incf2 [] *s3* ; **par** incf1 || incf2 **end par** **end select** ; outf

(balanced workflows). The inclusive gateway uses the select and par operators to allow all possible combinations of the outgoing branches. Note the introduction of synchronization points (s_i), which are necessary to indicate to the merge gateway the behavior that was executed at the split level. Without such synchronization points, the corresponding merge does not know whether it is supposed to wait for one or several branches (and which branches in this second case).

Once all BPMN elements are encoded into LNT, the last step is to compose them in order to obtain the behavior of the whole BPMN process. To do so,

```
process main [processApplication:any, createProfile:any, ...] is
 hide begin:any, finish:any, flow1_begin:any, flow1_finish:any, ... in
   par flow1_begin, flow1_finish, flow2_begin, flow2_finish, ... in
    par
      flow [flow1_begin, flow1_finish] ||... || flow [flow29_begin, flow29_finish]
    end par
   ||
    par
      init [begin,flow1_begin]
    || final [flow21_finish, finish] || final [flow27_finish, finish]
    || task [flow1_finish, processApplication, flow2_begin] || task [...] || ...
    || xorsplit [flow2_finish, flow3_begin, flow4_begin]
    || xormerge [flow6_finish, flow7_finish, flow29_begin]
    end par
   end par
 end hide
end process
```

Fig. 3. Main LNT Process for the Bank Account Opening Process

we compose in parallel all the flows with all the other constructs. All flows are interleaved because they do not interact one with another. All events and nodes (start/end events, tasks, gateways) are interleaved as well for the same reason. Then both sets are synchronized on flow sequences (sf0_begin, sf0_finish, etc.). These additional actions are finally hidden because they should not appear as observable actions and will be transformed into internal transitions in the resulting LTS. Each process call is accompanied with its alphabet, that is, the list of actions used in that process. For instance, each call of a flow process comes with a couple of actions corresponding to the initiation and termination of the flow.

Example. The translation of the bank account opening process in LNT results in several processes. The main process is given in Figure 3, whose LTS is shown in Figure 4.

4 Comparing Processes

In this section, we formally define several kinds of comparisons between BPMN processes. Their analysis allows one to ensure that the evolution of one process into another one is satisfactory.

Notation. LNT processes are denoted in italics, *e.g.*, p, and BPMN processes are denoted using a bold fond, *e.g.*, $\mathbf{b}$. In the sequel, we denote with $||p||$ the semantic model of an LNT process p, that is the LTS for p. Further, we denote the BPMN to LNT transformation introduced in the previous section using Θ, and the application of it to a BPMN process $\mathbf{b}$ using $\Theta(\mathbf{b})$. Accordingly, $||\ \Theta(\mathbf{b})\ ||$ denotes

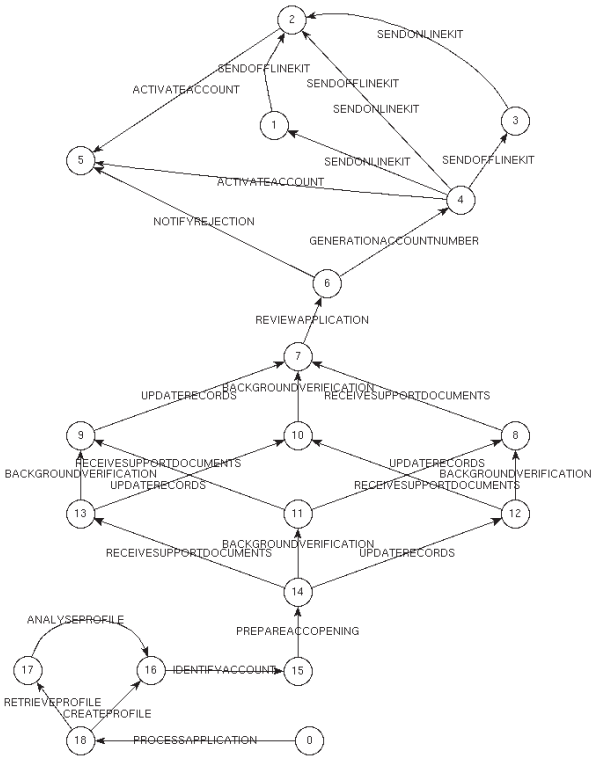


Fig. 4. LTS Formal Model for the Bank Account Opening Process

the LTS for this process. As far as the comparisons are concerned, we suppose we are in the context of the evolution of a BPMN process $\mathbf{b}$ into a BPMN process $\mathbf{b}'$, denoted by $\mathbf{b} \dashrightarrow \mathbf{b}'$.

4.1 Conservative Evolution

Our first comparison criterion is quite strong. Given an evolution $\mathbf{b} \dashrightarrow \mathbf{b}'$, it ensures that the observable behavior of $\mathbf{b}$ is exactly preserved in $\mathbf{b}'$. It supports very constrained refactorings of BPMN processes such as grouping or splitting parallel or exclusive branches (*e.g.*, $\langle\!\times\!\rangle(\langle\!\times\!\rangle(\mathbf{a},\mathbf{b}),\mathbf{c}) \overset{\dashrightarrow}{\dashleftarrow} \langle\!\times\!\rangle(\mathbf{a},\mathbf{b},\mathbf{c})$ where $\langle\!\times\!\rangle(x_1,\ldots,x_n)$ denotes a balanced exclusive split-merge). At the semantic level, several behavioral equivalences could be used. We have to deal with internal transitions introduced by hiding (see Section 3.2). Hence, we chose to use branching equivalence [30], denoted with $\overset{\text{br}}{\equiv}$, since it is the finest equivalence notion in presence of such internal transitions.

Definition 1. *(Conservative Evolution) Let* $\mathbf{b}$ *and* $\mathbf{b}'$ *be two processes,* $\mathbf{b} \dashrightarrow \mathbf{b}'$ *is a conservative evolution iff* $|| \Theta(\mathbf{b}) || \overset{\text{br}}{\equiv} || \Theta(\mathbf{b}') ||$.

4.2 Inclusive and Exclusive Evolution

In most cases, one does not want to replace a business process by another one having exactly the same behavior. Rather, one wants to be able to add new functionalities in the process, without interfering with the existing ones. A typical example is adding new choices, *e.g.*, $\diamondsuit(\mathbf{a}, \mathbf{b}) \dashrightarrow \diamondsuit(\mathbf{a}, \mathbf{b}, \mathbf{c})$, or evolving an existing one, *e.g.*, $\diamondsuit(\mathbf{a}, \mathbf{b}) \dashrightarrow \diamondsuit(\mathbf{a}, \diamondsuit(\mathbf{b}, \mathbf{c}))$. So here, we ground on a preorder relation rather than on an equivalence one, ensuring that, given $\mathbf{b} \dashrightarrow \mathbf{b}'$, all observable behaviors that were in $\mathbf{b}$ are still in $\mathbf{b}'$. For this we rely on the branching preorder [30], denoted by $\overset{\text{br}}{<}$.

Definition 2. *(Inclusive Evolution) Let* $\mathbf{b}$ *and* $\mathbf{b}'$ *be two processes,* $\mathbf{b} \dashrightarrow \mathbf{b}'$ *is an inclusive evolution iff* $|| \Theta(\mathbf{b}) || \overset{\text{br}}{<} || \Theta(\mathbf{b}') ||$.

Similarly, one may refine a process by implementing only a part of it. Here, in $\mathbf{b} \dashrightarrow \mathbf{b}'$, one does not want that $\mathbf{b}'$ exposes any additional behavior that is outside what is specified in $\mathbf{b}$. This is a reversed form of inclusive evolution.

Definition 3. *(Exclusive Evolution) Let* $\mathbf{b}$ *and* $\mathbf{b}'$ *be two processes,* $\mathbf{b} \dashrightarrow \mathbf{b}'$ *is an exclusive evolution iff* $|| \Theta(\mathbf{b}') || \overset{\text{br}}{<} || \Theta(\mathbf{b}) ||$.

The duality between inclusive and exclusive evolution is usual when one formalizes the fact that some abstract specification $\mathbf{a}$ is correctly implemented into a more concrete system $\mathbf{c}$. For some people, this means that at least all the behaviors expected from $\mathbf{a}$ should be available in $\mathbf{c}$. Taking the well-known "coffee machine" example, if a specification requires that the machine is able to deliver coffee, an implementation delivering either coffee or tea (depending on the people interacting with it) is correct. For others, *e.g.*, in the testing community, an implementation should not expose more behaviors than what was specified.

4.3 Selective Evolution

Up to now, we have supposed that all tasks in the original process were of interest. Still, one could choose to focus on a subset of them, called tasks of interest. This gives freedom to change parts of the processes as soon as the behaviors stay the same for the tasks of interest. For this, we define selective evolution up to a set of tasks T. Tasks that are not in this set will be hidden in the comparison process. Formally, this is achieved with a restriction operation $[T]$ on LTSs, which, given an LTS l, transforms all transitions whose label is not in T into *internal* transitions (their label becomes τ). Again, here we can rely on branching equivalence to deal with these internal transitions.

Definition 4. *(Selective Conservative Evolution) Let* $\mathbf{b}$ *and* $\mathbf{b}'$ *be two processes, and* T *be a set of tasks,* $\mathbf{b} \dashrightarrow \mathbf{b}'$ *is a selective conservative evolution with reference to* T *iff* $|| \Theta(\mathbf{b}') || [T] \overset{\text{br}}{\equiv} || \Theta(\mathbf{b}) || [T]$.

A specific interesting case of selective evolution is when the set of tasks of interest corresponds exactly to the tasks of the original process. This lets the designer add new behaviors not only in a separate way (as with inclusive evolution) but also within the behaviors of the original process. For example, $\mathbf{a};\mathbf{b} \dashrightarrow \diamondplus(\mathbf{a},\mathbf{log});\mathbf{b}$, that is a way to log some information each time $\mathbf{a}$ is done is not an inclusive evolution but is a selective conservative evolution with reference to $\{\mathbf{a},\mathbf{b}\}$. Accordingly to selective conservative evolution, we can define selective inclusive evolution (respectively selective exclusive evolution) by using the branching preorder, $\overset{\text{br}}{<}$, instead of $\overset{\text{br}}{\equiv}$.

4.4 Renaming and Refinement

One may also want to take into account renaming when checking an evolution $\mathbf{b} \dashrightarrow \mathbf{b}'$. For this we use a relabelling relation $R \subseteq T_{\mathbf{b}} \times T_{\mathbf{b}'}$, where $T_{\mathbf{b}}$ (respectively $T_{\mathbf{b}'}$) denotes the set of tasks in $\mathbf{b}$ (respectively $\mathbf{b}'$). Applying a relabelling relation R to an LTS l, which is denoted by $l \triangleleft R$, consists in replacing in l any transition labelled by some t in the domain of R by a transition labelled by $R(t)$.

To take into account task renaming in any of the above-mentioned evolutions, we just have to perform the equivalence (or preorder) checking up to relabelling in the formal model for $\mathbf{b}$. For example, $\mathbf{b} \dashrightarrow \mathbf{b}'$ is a conservative evolution up to a relabelling relation R for $\mathbf{b}$ and $\mathbf{b}'$ iff $|| \Theta(\mathbf{b}) || \triangleleft R \overset{\text{br}}{\equiv} || \Theta(\mathbf{b}') ||$.

Sometimes renaming is not sufficient, *e.g.*, when evolution corresponds to the refinement of a task by a workflow. We define a refinement rule as a couple (t, W), noted $t \dashrightarrow W$[4], where t is a task and W a workflow. A set of refinement rules, or refinement set, $\mathcal{R} = \bigcup_{i \in 1 \ldots n} t_i \dashrightarrow W_i$ is valid if there are no multiple refinements of the same task ($\forall i, j \in 1 \ldots n, i \neq j \Rightarrow t_i \neq t_j$) and if no refinement rule has in its right-hand part a task that has to be refined ($\forall i, j \in 1 \ldots n$, $t_i \notin W_j$). These constraints enforce that refinements do not depend on the application ordering of refinement rules, *i.e.*, they are deterministic.

To take into account refinement in evolution, a pre-processing has to be performed on the source process. For example, given that $\mathbf{b} \blacktriangleleft \mathcal{R}$ denotes the replacement in $\mathbf{b}$ of t_i by W_i for each $t_i \dashrightarrow W_i$ in $\mathcal{R}$, $\mathbf{b} \dashrightarrow \mathbf{b}'$ is a conservative evolution up to a refinement set $\mathcal{R}$ iff $|| \Theta(\mathbf{b} \blacktriangleleft \mathcal{R}) || \overset{\text{br}}{\equiv} || \Theta(\mathbf{b}') ||$.

4.5 Property-Aware Evolution

A desirable feature when checking evolution is to be able to focus on properties of interest and avoid in-depth analysis of the workflows. This gives the freedom to perform changes (including some not possible with the previous evolution

[4] The $\dashrightarrow$ symbol is overloaded since a refinement rule is an evolution at the task level.

relations) as long as the properties of interest are preserved. Typical properties include general ones such as deadlock freedom, and process specific ones such as safety and liveness properties defined over the alphabet of process tasks and focusing on the functionalities expected from the process under analysis. Such properties are written in a temporal logic supporting actions and, to make the property writing easier, the developer can rely on well-known patterns as those presented in [11].

Definition 5. *(Property-Aware Evolution) Let* $\mathbf{b}$ *and* $\mathbf{b}'$ *be two processes,* T *be a set of tasks, and* ϕ *be a formula defined over* T*,* $\mathbf{b} \dashrightarrow \mathbf{b}'$ *is a property-aware evolution with respect to* ϕ *iff* $|| \Theta(\mathbf{b}) || \models \phi \;\Rightarrow\; || \Theta(\mathbf{b}') || \models \phi$*.*

4.6 Context-Aware Evolution

A process is often used in the context of a collaboration, which in BPMN takes the form of a set of processes ("pool lanes") communicating via messages. When evolving a process $\mathbf{b}$, one may safely make changes as soon as they do not have an impact on the overall system made up of $\mathbf{b}$ and these processes. To ensure this, we have to compute the semantics of $\mathbf{b}$ communicating on a set of interactions I (a subset of its tasks) with the other processes that constitute the context of $\mathbf{b}$. We support two communication modes: synchronous or asynchronous. For each mode m we have an operation $\overset{m}{\times}_I$, where $|| \Theta(\mathbf{b}) || \overset{m}{\times}_I || \Theta(\mathbf{c}) ||$ denotes the LTS representing the communication on a set of interactions I between $\mathbf{b}$ and $\mathbf{c}$. For synchronous communication, $\overset{\mathrm{m}}{\times}_I$ is the LTS synchronous product [2]. For asynchronous communication, $\overset{\mathrm{m}}{\times}_I$ is achieved by adding a buffer to each process [3]. Here, to keep things simple, we will suppose without loss of generality, that a context is a single process $\mathbf{c}$.

Definition 6. *(Context-Aware Conservative Evolution) Let* $\mathbf{b}$*,* $\mathbf{b}'$*, and* $\mathbf{c}$ *be three processes,* $\mathbf{c}$ *being the context for* $\mathbf{b}$ *and* $\mathbf{b}'$*,* m *be a communication mode (*$m \in \{\mathrm{sync}, \mathrm{async}\}$*), and* I *be the set of interactions taking place between* $\mathbf{b}$ *and* $\mathbf{c}$*,* $\mathbf{b} \dashrightarrow \mathbf{b}'$ *is a context-aware conservative evolution with reference to* $\mathbf{c}$*,* m*, and* I *iff* $|| \Theta(\mathbf{b}) || \overset{m}{\times}_I || \Theta(\mathbf{c}) || \overset{\mathrm{br}}{\equiv} || \Theta(\mathbf{b}') || \overset{m}{\times}_I || \Theta(\mathbf{c}) ||$*.*

Accordingly, we may define context-aware inclusive and exclusive evolution, or combine them with renaming and refinement.

Example. We introduce in Figure 5 a revised version of the bank account opening process presented in Figure 2. In this new process, if the application is rejected, additional information may be asked to the customer. This is achieved adding a split exclusive gateway and a task "request additional info".

The two versions of the bank account opening process are not conservative because all traces including the task "request additional info" are present only in the new version of the process. However, both versions are related with respect to the inclusive/exclusive evolution notions. The new version includes all possible

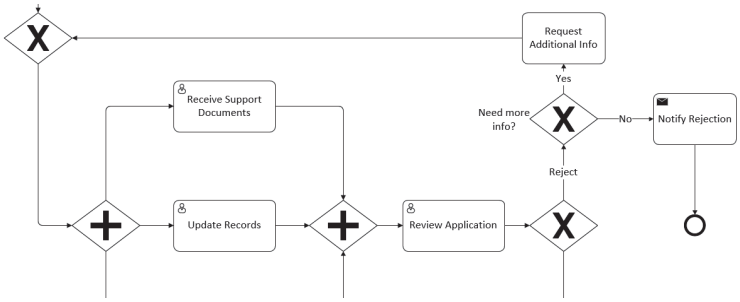


Fig. 5. Bank Account Opening Process in BPMN (V2, Partial View)

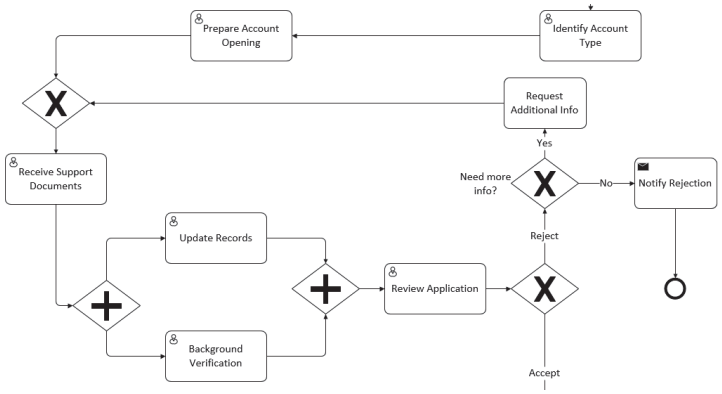


Fig. 6. Bank Account Opening Process in BPMN (V3, Partial View)

executions of the former one (the opposite is false) while incorporating new traces (those including "request additional info").

If we make another update to our process by taking the task "receive support documents" out of the parallel gateway (central part of the original process given in Figure 5). This slight modification is shown in Figure 6. In this case, both versions (V2 and V3) of the process are not conservative, because V3 is more restrictive and V2 exhibits behavior, *e.g.*, the trace "process application", "create profile", "identify account", "prepare opening", and "background verification", which does not appear in V3. However, all behaviors appearing in V3 are included in V2, so both versions are related *wrt.* the inclusive evolution relation.

As far as property-aware evolution is concerned, one can check for instance whether any process execution eventually terminates by a rejection notification or by an account activation. This is formalized in the MCL [23] temporal logic as shown below using box modalities and fix points. This property is actually satisfied for the original process, but not for its two extensions because in those processes a possible behavior is to infinitely request additional information.

[true* . PROCESSAPPLICATION] mu X .
(⟨ true ⟩ true and [not (NOTIFYREJECTION or ACTIVATEACCOUNT)] X)

5 Tool Support

The goal of this section is to present the implementation of the approach presented beforehand into our VBPMN tool and some experimental results. VBPMN is available online with a set of BPMN samples [1].

5.1 Architecture

VBPMN heavily relies on model transformation as depicted in Figure 7. The

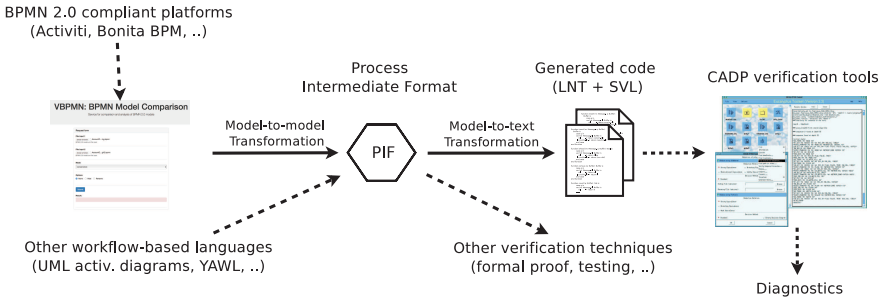


Fig. 7. Overview of VBPMN

central part is a pivot language called Process Intermediate Format (PIF). We propose this format to make our approach more modular, generic, and easily extensible. PIF gathers common constructs and operators one can find in any workflow-based modeling language. The interest of such an intermediate language is that several front-end modeling languages could be used as input (*e.g.*, UML activity diagrams or YAWL workflows). Further, several analysis techniques and tools could also be connected as a back-end to the PIF format (*e.g.*, to deal with data or timed aspects of processes).

As front-end, we integrate with BPMN editors by providing a Web application to which designers may submit the BPMN models they want to compare, together with parameters for the evolution. A model-to-model transformation is used to transform the BPMN processes to compare into PIF models. Then, a model-to-text transformation is used to generate from the PIF models two LNT encodings and a CADP verification script in the SVL language [12]. Equivalences ($\overset{\text{br}}{\equiv}$), preorders ($\overset{\text{br}}{<}$), and relabelling ($\lhd$, used for renaming) are directly supported by SVL commands. Restriction ($[T]$, used for selective evolution) is achieved by

using the SVL command for hiding all labels but for the ones to restrict to. The communication products ($\overset{\text{synch}}{\times_I}$ and $\overset{\text{asynch}}{\times_I}$) are achieved at the LNT level by using the par operator and (for $\overset{\text{asynch}}{\times_I}$) using additional LNT processes encoding buffers. The refinement (◀) and context-aware evolutions are not yet available in the current version of VBPMN. When one of the checks in the SVL scripts fails, one gets a witness (counter-example) that is presented in our Web application so that the designer can use it to modify the erroneous process evolution.

5.2 Experiments

We used a Mac OS laptop running on a 2.3 GHz Intel Core i7 processor with 16 GB of Memory. We carried out experiments on many examples taken from the literature or hand-crafted, and we present in Table 2 some of these results.

Table 2. Experimental Results

BPMN	Size			LTS (states/transitions)		Evol.
Proc.	Tasks	Flows	Gateways	Raw	Minimized	$\equiv$ $<$ $>$
1	6	11	2⟨✖⟩	29/29	8/9	× √ ×
1'	7	15	2⟨✖⟩+ 2⟨✚⟩	78/118	11/14	15s
2	4	7	1⟨○⟩	70/105	7/9	√ √ √
2'	8	14	2⟨✖⟩	36/38	10/12	15s
3	7	14	2⟨✖⟩+ 2⟨✚⟩	62/87	10/11	× × ×
3'	8	16	4⟨○⟩	1,786/5,346	28/56	15s
4	15	29	3⟨✖⟩+ 2⟨✚⟩+ 2⟨○⟩	469/1,002	24/34	× √ ×
4'	16	33	5⟨✖⟩+ 2⟨✚⟩+ 2⟨○⟩	479/1,013	26/37	15s
5	12	24	6⟨○⟩	742,234/3,937,158	148/574	× × √
5'	12	24	4⟨○⟩+ 2⟨✖⟩	6,394/21,762	60/152	31s
6	20	43	6⟨○⟩+ 6⟨✚⟩	4,488,843/26,533,828	347/1,450	× √ ×
6'	20	39	8⟨○⟩	4,504,775/26,586,197	348/1,481	9m31s

Each example consists of two versions of the process (original and revised). For each version, we first characterize the size of the workflow by giving the number of tasks, sequence flows, and gateways. We show then the size (number of states and transitions) of the resulting LTS before and after minimization. These reductions are useful for automatically removing unnecessary internal transitions, which were introduced during the process algebra encoding but do not make sense from an observational point of view. We use branching reduction [30], which is the finest equivalence notion in presence of internal

transitions and removes most internal transitions in an efficient way. Finally, the last column gives the results when comparing the LTSs for the two versions of the process using conservative, inclusive, and exclusive evolution, resp., and the overall computation time.

Examples 4 and 4' correspond to the first and second versions of our running example. Medium-size examples (*e.g.*, examples 5 and 6) can result in quite huge LTSs involving millions of states and transitions. This is not always the case and this is due to our choice to show processes in the table containing several parallel and inclusive gateways, which result in many possible interleaved executions in the corresponding LTSs. In our database, we have much larger examples of BPMN processes in terms of tasks and gateways, which result in small LTSs (thousands of states and transitions) due to their sequential behavior. Another comment concerns the considerable drop in size of the LTSs before and after minimization. Example 3' for example goes from about 2,000 states/5,000 transitions to about 30 states/60 transitions. This drastic reduction is due to all sequence flow actions encoded in LNT for respecting the BPMN original semantics. They do not have any special meaning *per se*, and are therefore hidden and removed by reduction.

As far as computation times are concerned, we observe that the final column of Table 2 gives the overall time, that is, the time for generating both LTSs, minimizing and comparing them. The comparison time is negligible. It takes 568 seconds for instance for generating and minimizing both LTSs for examples 6 and 6', and only 3 seconds for comparing both LTSs *wrt.* the three evolution notions considered in the table. On a wider scale, computation times remain reasonable (about 10 minutes) even for LTSs containing millions of states and transitions.

6 Related Work

Several works have focused on providing formal semantics and verification techniques for business processes using Petri nets, process algebras, or abstract state machines, see, *e.g.*, [21,26,10,33,34,9,25,14,22,18]. Those using process algebras for formalizing and verifying BPMN processes are the most related to the approach presented in this paper. The authors of [33] present a formal semantics for BPMN by encoding it into the CSP process algebra. They show in [34] how this semantic model can be used to verify compatibility between business participants in a collaboration. This work was extended in [32] to propose a timed semantics of BPMN with delays. In a previous work [25,14], we have proposed a first transformation from BPMN to LNT, targetted at checking the realizability of a BPMN choreography. We followed a state machine pattern for representing workflows, while we here encode them in a way close to Petri net firing semantics, which favours compositionality and is more natural for a workflow-based language such as BPMN. In [6], the authors propose a new operational semantics of a subset of BPMN focusing on collaboration diagrams and message exchange. The BPMN subset is quite restricted (no support of the inclusive merge gateway

for instance) and no tool support is provided yet. Compared to the approaches above, our encoding also gives a semantics to the considered BPMN subset by translation to LNT, although it was not our primary goal. The main difference with respect to these related works is our focus on the evolution of processes and its automated analysis.

In the rest of this section, we present existing approaches for comparing several BPMN processes (or workflows). In [19], the author proposes a theoretical framework for comparing BPMN processes. His main focus is substitutability and therefore he explores various sorts of behavioral equivalences in order to replace equals for equals. This work applies at the BPMN level and aims at detecting equivalent patterns in processes. In a related line of works, [17] studies BPMN behaviors from a semantic point of view. It presents several BPMN patterns and structures that are syntactically different but semantically equivalent. This work is not theoretically grounded and is not complete in the sense that only a few patterns are tackled. The notion of equivalence is similar to the one used in [19]. The authors of [17] also overview best practices that can be used as guidelines by modelers for avoiding syntactic discrepancies in equivalent process models. Compared to our approach, this work only studies strong notions of equivalence where the behavior is preserved in an identical manner. We consider a similar notion here, but we also propose weaker notions because one can make deeper changes (*e.g.*, by introducing new tasks) and in these cases such strong equivalences cannot be preserved.

In Chapter 9 of [27], the authors study the evolution of processes from a migration point of view. They define several notions of evolution, migration, and refactoring. Our goal here is rather complementary since we have studied the impact of modifying a workflow *wrt.* a former version of this workflow on low-level formal models, but we do not propose any solutions for applying these changes on a running instance of that initial workflow. In [31], the authors address the equivalence or alignment of two process models. To do so, they check whether correspondences exist between a set of activities in one model and a set of activities in the other model. They consider Petri net systems as input and process graphs as low-level formalism for analysis purposes. Their approach resides in the identification of regions (set of activities) in each graph that can coincide with respect to an equivalence notion. They particularly study two equivalence notions, namely trace and branching equivalences. The main limit of this approach is that it does not work in the presence of overlapping correspondences, meaning that in some cases, the input models cannot be analyzed. This work shares similarities with our approach, in particular the use of low-level graph models, hiding techniques and behavioral equivalences for comparing models. Still, our approach always provides a result and considers new notions of model correspondence such as property-aware evolution.

7 Concluding Remarks

This paper has introduced our approach for checking the evolution of BPMN processes. To promote its adoption by business process designers, we have implemented it in a tool, VBPMN, that can be used through a Web application. We have presented different kinds of atomic evolutions that can be combined and formally verified. We have defined a BPMN to LNT model transformation, which, using the LTS operational semantics of LNT enables us to automate our approach using existing LTS model checking and equivalence checking tools. We have applied our approach to many examples for evaluation purposes. It turns out that our tool support is rather efficient since it can handle quite huge examples within a reasonable amount of time.

In the implementation of our BPMN to LNT transformation, we rely on an intermediate format including the main workflow-based constructs. This paves the way for new front-end DSLs and other back-end verification techniques. Another perspective of this work is to propose quantitative analysis for comparing business processes as studied in [29,8]. Our goal is thus to consider non-functional requirements in BPMN processes, such as the throughput and latency of tasks, which can be modeled by extending LTSs with Markovian information and computed using steady-state analysis [7].

References

1. VBPMN Framework. https://pascalpoizat.github.io/vbpmn/.
2. A. Arnold. *Finite Transition Systems - Semantics of Communicating Systems*. Prentice Hall international series in computer science. Prentice Hall, 1994.
3. D. Brand and P. Zafiropulo. On Communicating Finite-State Machines. *J. ACM*, 30(2):323–342, 1983.
4. D. Champelovier, X. Clerc, H. Garavel, Y. Guerte, F. Lang, C. McKinty, V. Powazny, W. Serwe, and G. Smeding. Reference Manual of the LNT to LOTOS Translator, Version 6.1. INRIA/VASY, 2014.
5. D. R. Christiansen, M. Carbone, and T. T. Hildebrandt. Formal Semantics and Implementation of BPMN 2.0 Inclusive Gateways. In *Proc. of WS-FM'10*, volume 6551 of *LNCS*, pages 146–160. Springer, 2011.
6. F. Corradini, A. Polini, B. Re, and F. Tiezzi. An Operational Semantics of BPMN Collaboration. In *Proc. of FACS'15*, volume 9539 of *LNCS*, pages 161–180. Springer, 2015.
7. N. Coste, H. Garavel, H. Hermanns, F. Lang, R. Mateescu, and W. Serwe. Ten Years of Performance Evaluation for Concurrent Systems using CADP. In *Proc. of ISoLA'10*, pages 128–142, 2010.
8. A. K. Alves de Medeiros, W. M. P. van der Aalst, and A. J. M. M. Weijters. Quantifying Process Equivalence Based on Observed Behavior. *Data Knowl. Eng.*, 64(1):55–74, 2008.
9. G. Decker and M. Weske. Interaction-centric Modeling of Process Choreographies. *Information Systems*, 36(2):292–312, 2011.
10. R.M. Dijkman, M. Dumas, and C. Ouyang. Semantics and Analysis of Business Process Models in BPMN. *Inf. Softw. Technol.*, 50(12):1281–1294, 2008.

11. M. B. Dwyer, G. S. Avrunin, and J. C. Corbett. Patterns in Property Specifications for Finite-State Verification. In *Proc. of ICSE'99*, pages 411–420. ACM, 1999.
12. H. Garavel and F. Lang. SVL: A Scripting Language for Compositional Verification. In *Proc. of FORTE'01*, pages 377–394, 2001.
13. H. Garavel, F. Lang, R. Mateescu, and W. Serwe. CADP 2011: A Toolbox for the Construction and Analysis of Distributed Processes. *STTT*, 2(15):89–107, 2013.
14. M. Güdemann, P. Poizat, G. Salaün, and A. Dumont. VerChor: A Framework for Verifying Choreographies. In *Proc. of FASE'13*, pages 226–230, 2013.
15. ISO. LOTOS — A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour. Technical Report 8807, ISO, 1989.
16. ISO/IEC. International Standard 19510, Information technology – Business Process Model and Notation. 2013.
17. K. Kluza and K. Kaczor. Overview of BPMN Model Equivalences. Towards Normalization of BPMN Diagrams. In *Proc. of KESE'12*, pages 38–45, 2012.
18. F. Kossak, C. Illibauer, V. Geist, J. Kubovy, C. Natschläger, T. Ziebermayr, T. Kopetzky, B. Freudenthaler, and K.-D. Schewe. *A Rigorous Semantics for BPMN 2.0 Process Diagrams*. Springer, 2014.
19. V. Lam. Foundation for Equivalences of BPMN Models. *Theoretical and Applied Informatics*, 24(1):33–66, 2012.
20. C. Larman. *Applying UML And Patterns: An Introduction To Object-Oriented Analysis And Design And Iterative Development*. Prentice Hall, 2005.
21. A. Martens. Analyzing Web Service Based Business Processes. In *Proc. of FASE'05*, pages 19–33, 2005.
22. R. Mateescu, G. Salaün, and L. Ye. Quantifying the Parallelism in BPMN Processes using Model Checking. In *Proc. of CBSE'14*, pages 159–168, 2014.
23. R. Mateescu and D. Thivolle. A Model Checking Language for Concurrent Value-Passing Systems. In *Proc. of FM'08*, pages 148–164, 2008.
24. OMG. *Business Process Model and Notation (BPMN) – Version 2.0*. January 2011.
25. P. Poizat and G. Salaün. Checking the Realizability of BPMN 2.0 Choreographies. In *Proc. of SAC'12*, pages 1927–1934, 2012.
26. I. Raedts, M. Petkovic, Y. S. Usenko, J. M. van der Werf, J. F. Groote, and L. Somers. Transformation of BPMN Models for Behaviour Analysis. In *Proc. of MSVVEIS'07*, pages 126–137, 2007.
27. M. Reichert and B. Weber. *Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies*. Springer, 2012.
28. W. M. P. van der Aalst and A. H. M. Ter Hofstede. YAWL: Yet Another Workflow Language. *Information Systems*, 30:245–275, 2003.
29. B. F. van Dongen, R. M. Dijkman, and J. Mendling. Measuring Similarity between Business Process Models. In *Proc. of CAISE'08*, pages 450–464, 2008.
30. R. J. van Glabbeek and W. P. Weijland. Branching Time and Abstraction in Bisimulation Semantics. *J. ACM*, 43(3):555–600, 1996.
31. M. Weidlich, R. M. Dijkman, and M. Weske. Behaviour Equivalence and Compatibility of Business Process Models with Complex Correspondences. *Comput. J.*, 55(11):1398–1418, 2012.
32. P. Y. H. Wong and J. Gibbons. A Relative Timed Semantics for BPMN. *Electr. Notes Theor. Comput. Sci.*, 229(2):59–75, 2009.
33. P.Y.H. Wong and J. Gibbons. A Process Semantics for BPMN. In *Proc. of ICFEM'08*, pages 355–374, 2008.
34. P.Y.H. Wong and J. Gibbons. Verifying Business Process Compatibility. In *Proc. of QSIC'08*, pages 126–131, 2008.