CIT. Journal of Computing and Information Technology, Vol. 24, No. 2, June 2016, 195–207
doi: 10.20532/cit.2016.1002774

Liveness and Reachability Analysis of BPMN Process Models

Anass Rachdi, Abdeslam En-Nouaary and Mohamed Dahchour

Institut National des Postes et Télécommunication, Rabat, Morocco

Business processes are usually defined by business experts who require intuitive and informal graphical notations such as BPMN (Business Process Management Notation) for documenting and communicating their organization activities and behavior. However, BPMN has not been provided with a formal semantics, which limits the analysis of BPMN models to using solely informal techniques such as simulation. In order to address this limitation and use formal verification, it is necessary to define a certain "mapping" between BPMN and a formal language such as Concurrent Sequential Processes (CSP) and Petri Nets (PN). This paper proposes a method for the verification of BPMN models by defining formal semantics of BPMN in terms of a mapping to Time Petri Nets (TPN), which are equipped with very efficient analytical techniques. After the translation of BPMN models to TPN, verification is done to ensure that some functional properties are satisfied by the model under investigation, namely liveness and reachability properties. The main advantage of our approach over existing ones is that it takes into account the time components in modeling Business process models. An example is used throughout the paper to illustrate the proposed method.

ACM CCS (2012) Classification: Applied computing → Enterprise computing → Business process management → Business process modeling

Keywords: business process modeling, BPMN, Time Petri Nets, V&V, algorithm, distributed systems

1. Introduction

Business process modeling (BPM) is an approach to describe the way organizations conduct current or future business processes. It is a fundamental prerequisite for any organization that wishes to be engaged in business process improvement or business process management initiatives (BPMI) [1]. Usually, process models describe in an informal graphical way the activities, events, dataflow and control flow logic that constitute a business process. Therefore, business process models are considered essential for the analysis and design of process aware information systems, organizational documentation and re-engineering as well as for service oriented architecture implementation and enterprise application integration [1]. All these features make it possible for an organization to align internal business operations with its business strategy and customer needs, and helps managers determine the right way to direct, monitor and measure company resources. In other words, business process management, if properly implemented, is a powerful means for reducing costs, enhancing efficiency and productivity, and minimizing errors and risks.

Given the numerous benefits and advantages of business process modeling, the challenging question for organization remains how to create correct, strong, and yet flexible, business processes. This could not be answered without conducting two essential steps. On one hand, adequate languages should be adopted for modeling business processes in terms of the organization's structure, constraints and behavior. On the other hand, the resulting models should be analyzed and verified to make sure that some desirable (respectively undesirable) properties are (respectively are not) satisfied.

One of the languages proposed in literature to model business processes, is the Business Process Modeling Notation (BPMN 2.0) [2]. It is an adopted standard in both academia and industry that was designed to provide a graphical notation for XML-based business process languages, like Business Process Execution Language (BPEL) [3]. Business analysts can take advantage of this feature to automatically generate executable BPEL code from BPMN graphical models [4] using some tools such as *bpmn2bpel* eclipse plugin [5].

Unfortunately, BPMN is informal and leaves room for ambiguities, inconsistencies (existence of superfluous attributes, deviations between process definition and its execution semantics...) [6] and misinterpretations of several concepts (lifecycle, interruptions, expression evaluation, and completion of processes…) [7]. Hence, business process model analysis is considered a critical step in BPM's life cycle. It is closely related to the modeling phase where the use of intuitive languages as BPMN is highly sought by business experts. Currently, there are several implementations of BPMN, but the one developed by BPMI and adopted by the OMG specification does not have a formal semantics. The formal semantics helps us avoid undesired situations as well as control flow anomalies (livelock, deadlock, dead activities or paths…). In order to analyze BPMN diagrams formally, we have to extract a formal model that respects the specifications on which the initial model was based. Usually, we define a "mapping" from the graphical notation to a formal language such as Communication Sequential Processes (CSP) or Petri Nets (PN). If we consider CSP as a target language, we will have to deal with some synchronization mechanisms defined within one participant (one pool) that are not necessarily needed in a functional analysis. On the other hand, standard Petri nets lack temporal aspects that are required for a more complete analysis of distributed systems.

In this paper, we basically propose a method for the verification of BPMN models by defining the formal semantics of BPMN in terms of a mapping to Time Petri Nets (TPN). The reason we have chosen TPN (in which time is associated to transitions) among other semantics (Place-TPN, Arc-TPN) is that transitions determine the elapsed time and it is more natural to associate time to transitions. Transitions represent activities which normally take time. After the translation of BPMN models into TPN, verification is done to ensure that some functional properties are satisfied by the model under investigation. Contrary to existing mapping methods found in literature, our method takes into consideration the time concepts in BPMN models and defines the fundamental verification properties, such as liveness and reachability.

The remainder of this paper is organized as follows. The next section proposes the work related to our approach. Section 3 presents the technical background needed for the rest of the paper; it is divided into two major parts. The first one introduces a core subset of BPMN elements, explores some problems found in the standard, and presents a formal definition of time based on the abstract syntax defined in [8]. The second one captures the essence of the Time Petri net language which constitutes a pre-requisite for the next sections. Section 3 presents our contribution to the analysis and the verification of BPMN models by proposing a formal mapping from BPMN to Time Petri Nets. Section 5 proposes a new algorithm for the reachability analysis of TPN. The latter is inspired by the algorithm introduced in [9] which verifies the executability of transitions in Timed Automata (TA). In our case it will help us detect control flow anomalies such as dead paths (dead transitions). Section 6 concludes the paper and presents future work.

2. Related Work

Many methods found in literature have addressed the problem of defining a formal semantics for BPMN. Dijkman et al. in [8] have proposed a formal semantics of BPMN defined in terms of a mapping to Standard Petri nets. The proposed mapping has been implemented as a tool that generates PNML (Petri Net Markup Language) code. The latter could be verified by ProM tool.

Another proposal was made in [10] by Wong and Gibbons in which the authors used CSP's semantics to principally map participants (pools) to CSP processes and actions. These CSP processes are analyzed by FDR tool to detect dead and live locks.

Another interesting contribution was introduced in [11] where the authors opted for a BPMN-YAWL mapping to check functional properties. YAWL has been chosen to overcome some limits found in BPMN-PN mapping. In fact YAWL has introduced The "nofi" activity to denote the multi-instance activities and the

Adhoc subprocesses. In [12], Ouyang et al. present a mapping to BPEL in order to facilitate the execution of the BP diagrams. Unfortunately, all the aforementioned methods have not included the time dimension in their analysis.

In this paper, the approach we propose takes into consideration the time concepts in BPMN models and proposes a formal semantics of BPMN defined in terms of a mapping to Time Petri Nets (TPN) which are equipped with very efficient analytical techniques

3. Background

This section introduces the specification models used in this paper, namely BPMN and TPN. It also presents some problems found in the BPMN standard as well as a formal definition of time that will help us constitute the BPMN-TPN mapping explained in Section 3. Throughout this section we use simple examples to illustrate these two modeling languages.

3.1. Preliminaries

The following notations and definitions are used to present the rest of this paper. $\mathbb{N}$ denotes the set of natural numbers while $\mathbb{N}^+$ is used for $\mathbb{N} \setminus \{0\}$. $\mathbb{Q}^+$ denotes the set of non-negative rational numbers while $\mathbb{R}^+$ stands for the set of non-negative real numbers. C denotes the set of all possible conditions. A condition is a boolean function operating over a set of propositional variables. It is assumed that a condition evaluates to true or false [8]. A vector of natural numbers is a collection $V = (v_1, v_2, \ldots, v_n)$, where $n \in \mathbb{N}$ and $v_i \in \mathbb{N}$ for $i = 1, \ldots, n$; n is called the cardinality of V. For two vectors $V_1 = (v_{11}, v_{12}, \ldots, v_{1n})$ and $V_2 = (v_{21}, v_{22}, \ldots, v_{2n})$ of cardinality n, we define the subtraction of V_2 from V_1 (written $V_1 - V_2$) as a vector V_3 of cardinality n, obtained by subtracting each element in V_2 from its pair-wise element in V_1; i.e., $V_3 = V_1 - V_2 = (v_{11} - v_{21}, v_{12} - v_{22}, \ldots, v_{1n} - v_{2n})$. Similarly, the sum of two vectors $V_1 = (v_{11}, v_{12}, \ldots, v_{1n})$ and $V_2 = (v_{21}, v_{22}, \ldots, v_{2n})$ of cardinality n, written $V_1 + V_2$, is a vector V_3 of cardinality n, obtained by adding each element in V_2 to its pair-wise element in V_1; i.e., $V_3 = V_1 + V_2 = (v_{11} + v_{21}, v_{12} + v_{22}, \ldots, v_{1n} + v_{2n})$. For a vector $V = (v_1, v_2, \ldots, v_n)$ and a rational number $d \in \mathbb{R}^+$, we define the sum of V and d (written $V + d$) as a vector V_2 obtained by adding d to each element in V; i.e., $V_2 = V_1 + d = (v_1 + d, v_2 + d, \ldots, v_n + d)$. We denote $V_1 \geq V_2$ if $V_3 = V_1 - V_2$ and $V_3 \geq 0$ (i.e for $i = 1, \ldots, n,$ $v_{1i} - v_{2i} \geq 0$)

3.2. Business Process Management Notation (BPMN)

3.2.1. General Notions

BPMN is an emerging modeling method for business people. It has received a lot of interest and support from academia, industry and tool vendors as an open standard for modeling business processes. Besides being one of the most recent modeling notations standardized by OMG and BPMI, BPMN is considered user-friendly to all organization stakeholders (managers, analysts, developers…) and allows a business process to be modeled with a single diagram type, avoiding the fragmentation problem inherent in other modeling languages like UML. As such, BPMN also helps improve and facilitate communication between business process stakeholders (managers, analysts, developers…). Processes are represented in BPMN using constructs that can be grouped into four categories: flow objects (events, activities, and gateways), connection objects (control flow, message flow and associations), artefact objects (data objects, data stores, data input and data output) and swim lanes (pools and lanes within pools), as illustrated in Figure 1. Events can be partitioned into disjoint sets of start events, intermediate events and end events. Intermediate events can be further partitioned into disjoint sets of intermediate message events, intermediate time events and intermediate error events. A start event is used to indicate the start of a process while an end event represents the end of a process. An intermediate event is basically something that might happen during the execution of a process.

An activity is either a task or a subprocess that can be used to provide some business service, wait for a message from another participant, or send a message to another participant [14]. A gateway is a connector used to control sequence

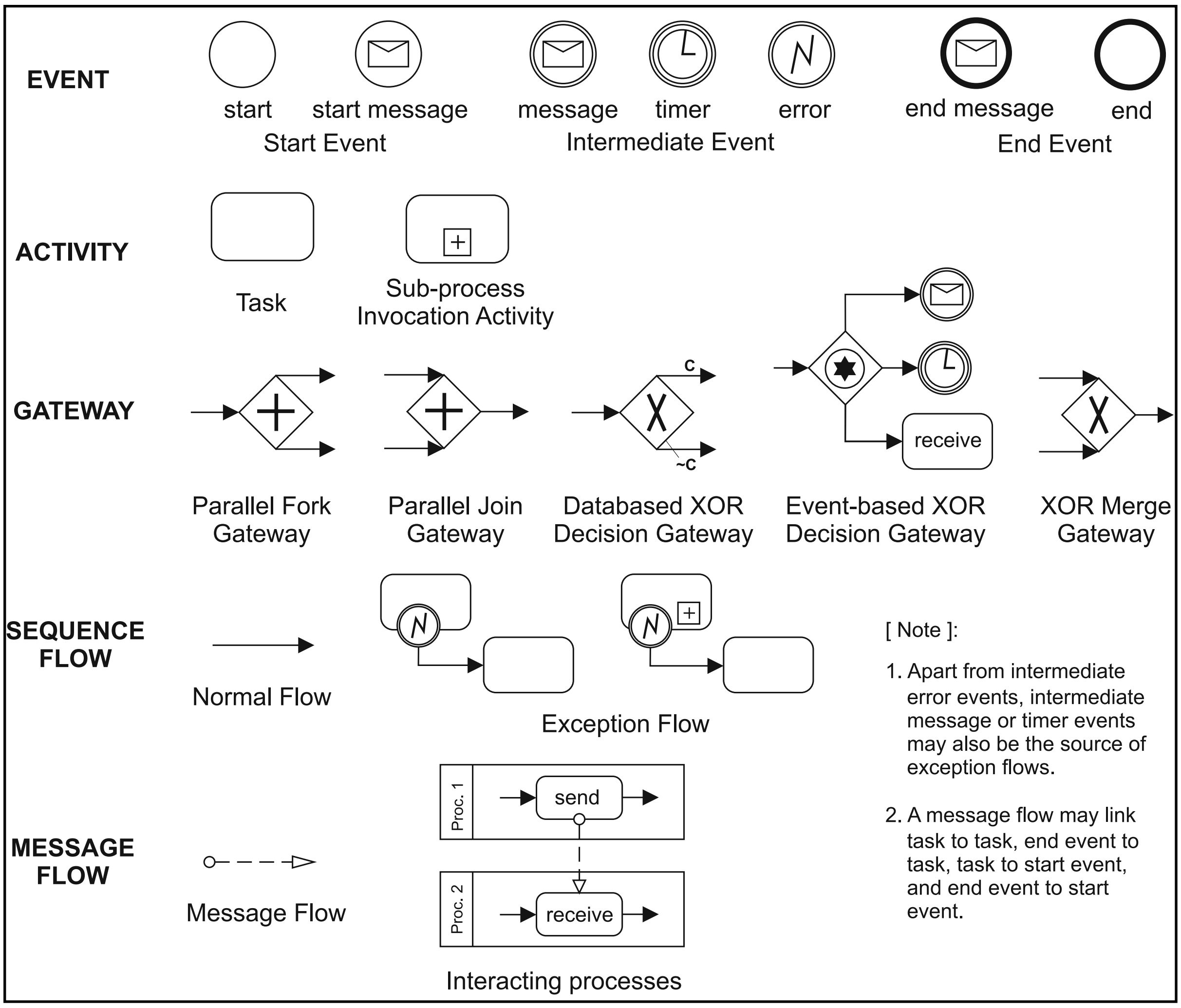
EVENT
start
start message
Start Event
message
timer
error
Intermediate Event
end message
end
End Event
ACTIVITY
Task
Sub-process Invocation Activity
GATEWAY
c
~c
receive
Parallel Fork Gateway
Parallel Join Gateway
Databased XOR Decision Gateway
Event-based XOR Decision Gateway
XOR Merge Gateway
SEQUENCE FLOW
Normal Flow
Exception Flow
[Note]:
1. Apart from intermediate error events, intermediate message or timer events may also be the source of exception flows.
2. A message flow may link task to task, end event to task, task to start event, and end event to start event.
MESSAGE FLOW
Message Flow
Proc. 1
send
Proc. 2
receive
Interacting processes

Figure 1. A core subset of BPMN elements [13].

flows. We distinguish between multiple types of gateways: an AND-split gateway is used to create parallel flows and an AND-join gateway is used to synchronize incoming parallel flows [14]. A XOR data-based gateway defines a set of alternative paths; each of them is associated with a conditional expression [14]. Based on this condition, only one path can be taken during the execution of the process [14]. Conditions can be based either on data-base entries or on external events [14]. An exclusive merge gateway is used as a merge for alternative sequence flows. Finally, an inclusive merge gateway synchronizes all tokens produced upstream. Sequence flows determine the execution order between two objects in the same pool [14]. However, message flows represent message exchange between two objects in different pools.

While a start message event indicates that a message arrives from a pool (participant) and triggers the start of the process, an end message event represents the fact that a message is sent to another participant at the end of the process [14]. An intermediate message event indicates that a message arrives from or is sent to a participant during the process execution [14]. A timer event indicates a specific time-date being reached. An error message is for error handling. If the error is part of a normal flow, it throws an error; if it is attached to the boundary of an activity, it catches the error [14]. The behavior of a process could be described by tracking the path(s) of a token through a process [14]. A token, in BPMN, is an abstract object that traverses the sequence flow passing through the objects of the process [14].

An example of BP is shown in Figure 2. It represents the visa application process with two participants, namely the visa applicant and the visa application center. The process goes through the following steps:

1. The process is triggered by the arrival of a visa application folder (in the form of a message sent by the visa applicant). The visa application folder contains all the required documents (passport, visa application form…)
2. To facilitate the tracking of the visa application, the front office must scan and integrate the requested documents in the Information System (IS) within a maximum period of one day.
3. Once the documents are present in the system, the back office starts processing the visa application by verifying the presence of all requested documents, financial status, social status… The verification duration cannot exceed 13 days.
4. If the applicant is accepted, the back office starts printing the visa on the passport within a maximum period of one day. Otherwise, the visa application center has to inform the applicant that he/she is refused. The visa processing time cannot exceed 15 days, whether the applicant is accepted or rejected.

Despite the richness and the simplicity of its graphical notation, BPMN (1.0 and 2.0) has always been subject to much criticism relative to its lack of formal semantics. In fact, BPMN suffers from many problems that limit the model's verification to informal techniques such as simulation. We focused in this paper on some problems that are related to the flow control analysis [7]. These problems are detailed in the following:

- Lack of state representation: BPMN remains unable to model state-related aspects of business processes. Therefore, some functional properties such as liveness and reachability cannot be verified on the BPMN model unless it is mapped to another formal language that supports state modeling, e.g. Time Petri Net.
- Weak conceptual support for essential notions such as concurrent process communication/interaction, resources… [7].
- Lack of important attributes: BPMN lacks attributes in the specification, which plays an important role in flow control analysis, e.g. BPMN standard should add the activation time attribute in the activities and timer properties in order to obtain more complete analysis.

Other problems related to artefacts, swimlanes and connection objects can be found in [7] and [6].

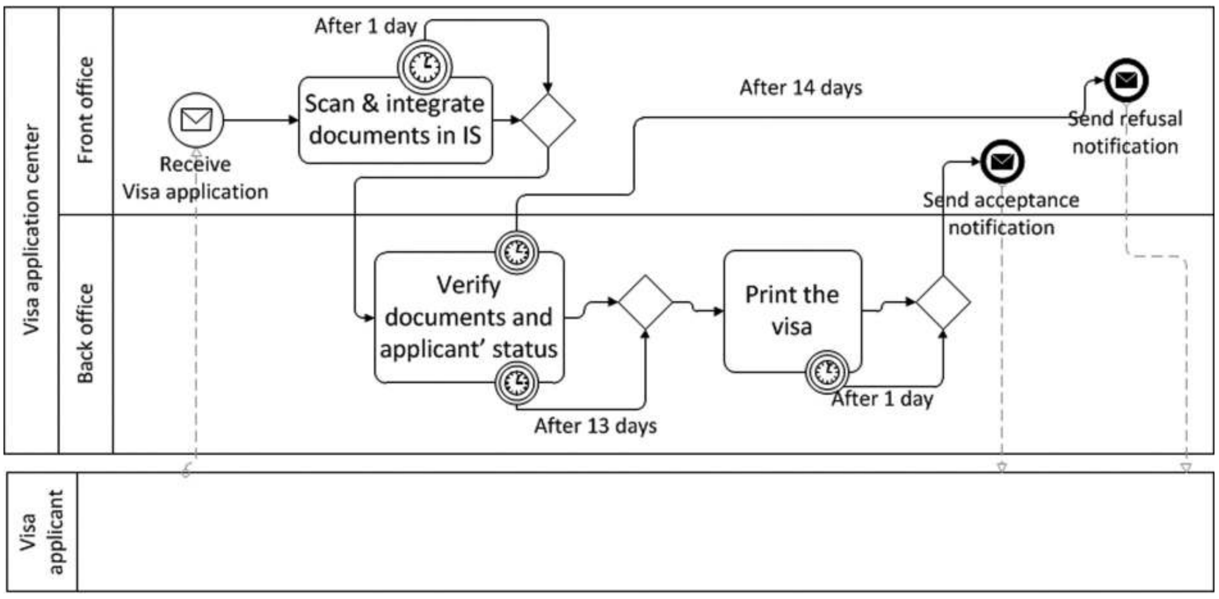

Figure 2. A simple visa application processing Process System modeled in BPMN.

3.2.2. Formal Definition of Time in BPMN

Time in BPMN is defined through TimerEventDefinition [2] element which has three attributes that are mutually exclusive. These attributes are timeDuration (Ti_{Dur}), timeDate (Ti_{Dat}) and timeCycle.

Based on the abstract syntax of BPMN notation given in [8], we can formalize Time in BPMN by adding a new function called "Tim" that assigns to each timer a time duration. Consequently, we can define the BPMN Core BPMN Process as follows:

Definition 1. (Core BPMN Process): A core BPMN process is a tuple $P = (O_P, F_P, Cond_P, Excp_P, Tim_P)$ where [8]:

- O_P is a set of flow objects which can be partitioned into disjoint sets of activities A_P, events E_P, and gateways G_P, [8]
    - A_P can be partitioned into disjoint sets of tasks T_P and subprocess invocation activities S_P, we denote $T_P^R \subseteq T_P$ the set of receive tasks, [8]
    - E_P can be partitioned into disjoint sets of start event E_P^S, intermediate events E_P^I and end events E_P^E, we denote $E_P^{ST} \subseteq E_P^S$ the set of timer start events. E_P^I can be partitioned into disjoint sets of intermediate message events E_P^{IM}, intermediate timer events E_P^{IT}, and intermediate error events E_P^{IR} [8].
    - G_P can be partitioned into disjoint sets of parallel fork gateways G_P^F, parallel joint gateways G_P^J, data-based XOR decision gateways G_P^X, event-based XOR decision gateways G_P^V, and XOR merge gateways G_P^M [8],
- $F_P \subseteq O_P \times O_P$ is the control flow relation, i.e. a set of sequence flows connecting objects [8],
- $Cond_P$: $F_P \cap (G_P^X \times O_P) \rightarrow C$ is a function which maps sequence flows emanating from data-based XOR gateways to conditions [8],
- $Excp_P$: $E_P^I \rightarrow A_P$ is a function which assigns an activity to an intermediate event such that the occurrence of the event signals an exception and thus interrupts the performance of the activity [8].
- Tim_P: $E_P^{IT} \cup E_P^{ST} \rightarrow \mathbb{Q}^+ \cup \{\alpha\}$ is a function assigning a positive rational number or the symbol α to a timer event. The positive rational number represents the timer's duration. The latter specifies the time to be elapsed before the timer gives the flow control to its successor. We denote Tiact the instant the timer was last activated.

The timer duration can be static or dynamic: it is static when timeDuration attribute is set, otherwise it is dynamic. In this case it is calculated in two ways:

- If timeDate attribute is set, then $Ti_{Dur} = Ti_{Dat} - Ti_{act}$
- If timeCycle attribute is set, then $Ti_{Dur} = T_{inext\text{-}trigger} - Ti_{act}$ where $Ti_{next\text{-}trigger}$ is the next time the timer will trigger

If $Ti_{Dur} \geq 0$ then Tim_P returns a rational number, otherwise it returns α which means that the timer no longer has an impact on the control flow.

A core BPMN process P is a directed graph with nodes (objects) O_P and arcs (sequence flows) F_P. For any node $x \in O_P$, input nodes of x are given by $in(x) = \{y \in O_P \mid yF_Px\}$ and output nodes of x are given by $out(x) = \{y \in O_P \mid xF_Py\}$ [8].

3.3. Time Petri Nets (TPN)

Petri nets were proposed in 1962 by Carl Adam Petri [15] as a modeling formalism to be used in computer science, system engineering and many other disciplines [16]. Petri nets combine a well defined mathematical theory and tool support with a graphical representation for a precise modeling and analysis of system behavior [16]. Nevertheless, Petri nets were initially proposed as a formal language with no concept of time or probability. However, for many practical applications (in particular time critical systems), time is a mandatory aspect that designers should take into account to analyze correctly the behavior and performance of their distributed information systems. To cope with timing behavior of distributed information systems, time Petri nets [17] have been proposed as an extension of standard Petri nets by adding clocks and timing constraints to transitions in order to help describe and analyze properly time dependent systems. Time Petri nets are obtained from standard Petri nets by simply associating a firing time interval $[a, b]$ to each transition t, where a and b are rational numbers such that $0 \leq a \leq b$ and $a \neq \infty$. The times a and b, for a transition t, are relative to the moment at which t was last enabled; they are referred to as the earliest firing time and the latest firing time of t, respectively.

Formally, a time Petri net [17] is defined as a 7-tuple $Y = (P, T, F, W, M_0, f_S)$ where :

- $P = \{p_1, p_2, ..., p_m\}$ is a finite non-empty set of places ;
- $T = \{t_1, t_2, ..., t_n\}$ is a finite non-empty set of transitions, where $P \cap T = \varnothing$;
- $F \subseteq (P \times T) \cup (T \times P)$ is the set of arcs that connect transitions and places. Two types of arcs are usually used to connect places to transitions: usual arcs are denoted by small arrow next to transitions and inhibitor arcs are marked by a small circle next to transitions. Let $\bullet t = \{p \in P$ et pFt$\}$ (resp $t\bullet = \{p \in P$ et tFp$\}$) be the set of pre-places (resp post-places) of t.
- $W: (P \times T) \cup (T \times P) \rightarrow \mathbb{N}^+$ is the weight function for the directed arcs of Y. This function is very important for the execution of the Petri net, as will be explained later. For a transition t, we use $W(\bullet t)$ to denote the vector of tokens required to fire the transition t. Similarly, we use $W(t\bullet)$ to denote the vector of tokens produced by firing the transition t. Formally, $W(\bullet t) = \langle W(p_1, t), W(p_2, t), \ldots, W(p_m, t)\rangle$ and $W(t\bullet) = \langle W(t, p_1), W(t, p_2), \ldots, W(t, p_m)\rangle$.
- $M_0 \in \mathbb{N}^m$ is the initial marking of Y, which indicates the initial configuration of the Petri net at the beginning of its execution.
- $f_s: T \rightarrow \mathbb{Q}^+ \times \mathbb{Q}^+ \cup \{\infty\}$ is the function that associates each transition to a firing time interval. For any transition $t \in T$, we write $f_s(t) = (l, u)$ where l is the earliest firing time of t (denoted by eft(t)), whereas u is the latest firing time (denoted by lft(t)); if u is infinite, then the transition is said not to have a latest firing time.

In order to study the dynamic behavior of TPN, we need to define its operational semantics. Such semantics is given by a state transition system obtained by using the concepts of marking, state and firing of transitions.

A marking M of a TPN is basically a vector of natural numbers that tells us how many tokens each place holds. Tokens are a primitive concept for Petri nets in addition to places and transitions. The presence or absence of a token in a place can indicate, for instance, whether a condition associated with this place holds or not [16]. The number and position of tokens may change during the execution of a Petri net. For a marking M in $\mathbb{N}^P$, $M(p)$ is used to denote the number of tokens in place p. A state of a TPN is given by (M, V), where M is a marking and $V: T \rightarrow \mathbb{Q}^+ \cup \{\omega\}$ is a valuation such that each value $V(t)$ is the time elapsed since the transition t was last enabled. $V(t)$ is considered as a clock associated with the transition t. The transition is said to be disabled if there exists p in P such as $M(p) < W(p, t)$, in this case the clock remains off ($V(t) = \omega$). V_0 is the initial valuation with $V_0(t) = \begin{cases} 0 & M_0 \geq W(\bullet t) \\ \omega & \text{otherwise} \end{cases}$ for each transition t in TPN. Using the concept of state, the execution of a TPN can be stated as follows. A Petri net is executed by firing enabled transitions, following these two rules:

- Enabling rule: A transition t is enabled in a state (M, V) if each input place p of t contains at least the number of tokens equal to the weight of the directed arc connecting p to t (i.e. $M(p) \geq W(p, t)$ for any p in P). If an inhibitor arc connects an input place p to transition t, then enabling of transition t also requires the input place p not to have $W(p, t)$ tokens. As soon as t becomes enabled, $V(t)$ is set to 0; otherwise it remains ω.
- Firing rule: A transition t is ready to fire if it is enabled and eft(t) $\leq V(t) \leq$ lft(t), and such firing consists of removing from each input place p of t the number of tokens equal to the weight of the directed arc connecting p to t, and adding in each output place p' of t the number of tokens equal to the weight of the directed arc connecting t to p'. In other words, the firing of an ex-

plicit transition t in state (M, V) results in changing the state (M, V) to (M', V').

$(M,V) \xrightarrow{t} (M',V')$. The new state (M', V') is calculated as follows:

- ○ $M' = M - W(\bullet t) + W(\bullet t)$, and
- ○ $$\forall t' \in T,\ V'(t') = \begin{cases} \omega & \text{if } \exists p \in P \\ & M'(p) < W(p,t') \\ V(t') & \text{if } \forall p \in P \\ & M(p) \geq W(p,t'), \\ & M'(p) \geq W(p,t'), \\ & \bullet t \cap \bullet t = \varnothing \\ 0 & \text{otherwise} \end{cases}$$

Other transitions that are based on time delay are called implicit transitions. They reflect state change on time progress when a transition is last enabled and before it becomes enabled again. Their semantics is explained as follows:

- Implicit transitions on time delay d: $(M,V) \xrightarrow{d} (M',V')$ iff:
  - ○ $M = M'$,
  - ○ $V(t) + d \leq \text{lft}(t)$ for any transition t, and
  - ○ $V' = V + d$ for any enabled transition in TPN.

Figure 3 shows an example of TPN $Y = (P, T, I, O, M_0, f_S)$, with three places ($p_1$, p_2 and p_3) and two transitions (t_1 and t_2), such that:

- $P = \{p_1, p_2, p_3\}$,
- $T = \{t_1, t_2\}$,
- $I(t_1) = \{p_1\}$,
- $I(t_2) = \{p_1\}$,

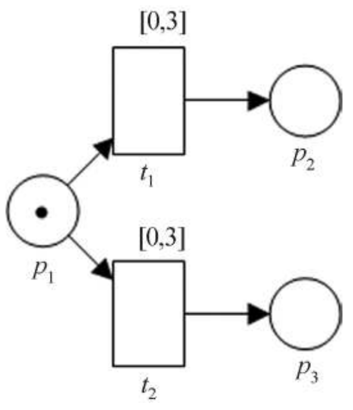

Figure 3. A Simple Time Petri Net.

- $O(t_1) = \{p_2\}$,
- $O(t_2) = \{p_3\}$,
- $M_0 = (1, 0, 0)$,
- $f_S(t_1) = [0, 3]$, and $f_S(t_2) = [0, 3]$.

After introducing the specification languages used in this paper, we present in the next section our method for the analysis of BPMN based on TPN.

4. Our Approach for BPMN Model Analysis

As mentioned so far, BPMN is an adopted standard used in academia and industry for modeling business processes. However, BPMN is informal and leaves room for misinterpretations, ambiguities, and inconsistencies about the execution and operation of business processes being modeled. Hence, we need to define semantics for BPMN in order to analyze business processes properly and remove any possible errors before implementations. To this end, we chose TPN as target formalism. This choice is motivated by several reasons. First, TPN is a well-founded semantics; it has a mathematical background and is equipped with very efficient analytical tools (TINA). Secondly, it takes into consideration the time concepts that are essential to obtain a more complete functional analysis.

Our approach for the analysis of BPMN models basically consists of two main steps. First, we transform the BPMN into a TPN; secondly, we do formal verification on the resulting TPN. Such verification consists of checking the marking reachability and the liveness of transitions of TPN. These properties can be defined as follows:

- A state $z = (M,V)$ is reachable from $z_0 = (M_0,V_0)$ iff there exist $w \in T$ (a sequence of transitions) and $\xi \in \mathbb{Q}^{\text{length}(w)+1}$ (implicit transitions) such as $z \xrightarrow[\xi]{w} z'$.

  We can limit ourselves to the definition of reachability that is just related to the markings and not to the state in general (the same definition found in standard Petri net [15]): A marking M' (in state z') is reachable from a marking M (in state z) if there exists a sequence of transitions $w \in T$ such as $M \xrightarrow{w} M'$.

- In order to define liveness in TPN [16], we will need to introduce $R_Y(z)$ as the set of all reachable states from the state $z = (M, V)$. We denote $z_0 = (M_0, V_0)$ the initial state in TPN. $\forall z = (M, V) \in R_Y(z_0)$
  - t is live in z iff: $\forall z' \in R_Y(z) \rightarrow \exists z'' \in R_Y(z')$ such as t is ready to fire in z''
  - t is dead iff $\forall z' \in R_Y(z_0)$ t is not ready to fire in z'.

4.1. Mapping of BPMN Silent Events and Gateways

Contrary to what is used in the classical mapping of BPMN-PN [8], we will not use two notations to differentiate the silent transitions from the timed ones. One type of transition is capable of capturing both the external and internal actions that cannot be observed by users. We can differentiate the silent transitions from those consuming time based on the time intervals that are associated to these transitions.

Figure 4 illustrates the mapping of silent events and gateways to TPN. The same principle applies to complex and event-based gateways.

4.2. Timed Activities and Events

Figure 5 illustrates the mapping of timed tasks and events to Time Petri Net. The "Timer" component is highlighted in this mapping; its main function is to determine the maximum execu-

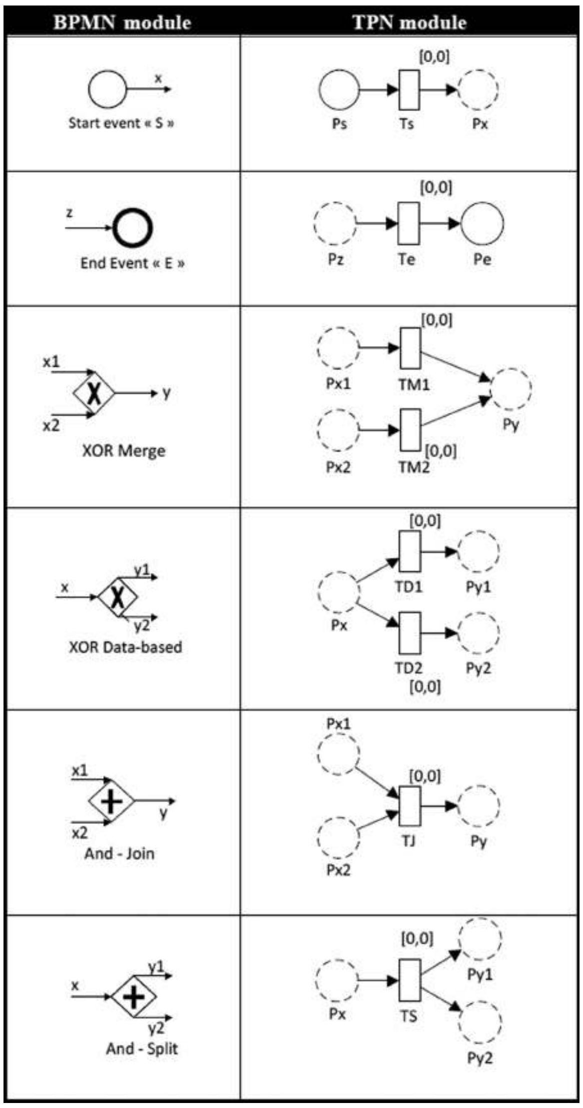

Figure 4. Mapping of silent events and gateways to TPN.

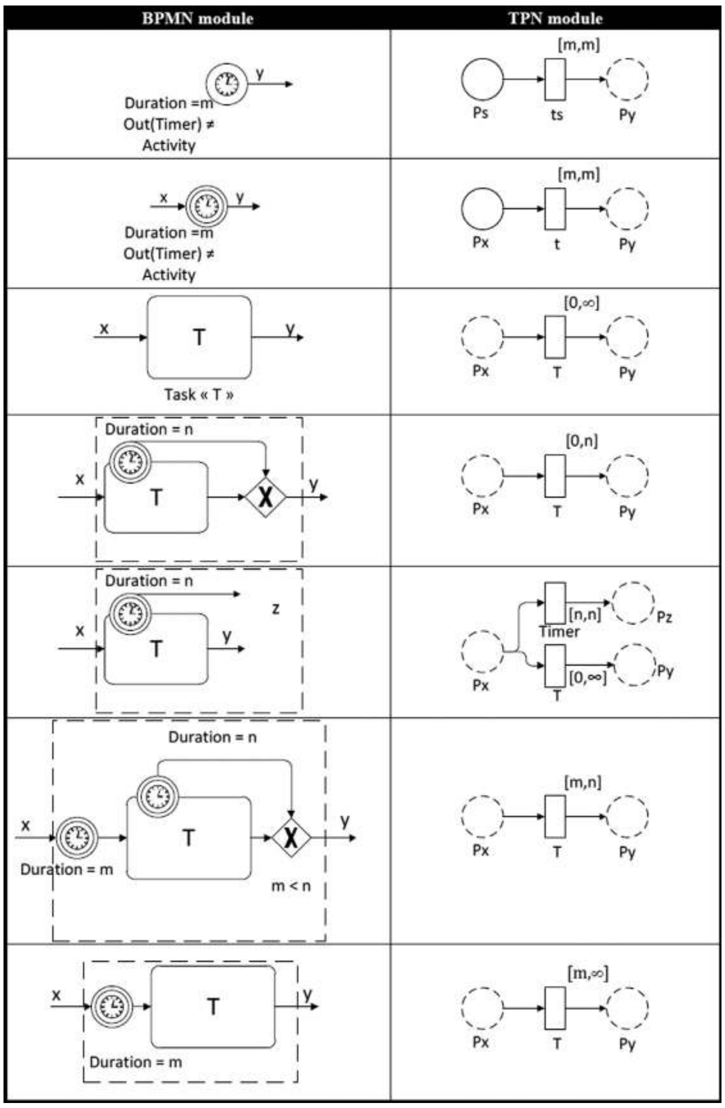

Figure 5. Mapping timed activities and events to TPN.

tion time of the BPMN task. The “Timer” event does not specify the maximal execution time of the BPMN task unless its output sequence flow form with the task’s one , the input of a gateway of type “XOR-MERGE”.

If this condition is not fulfilled, the “Timer” event will constitute a normal exception. The mapping of the exception was already covered in BPMN-PN mapping [8]. Nevertheless, we must associate a time interval to the transition representing the “Timer” event (line 5 in Figure 5).The timer can also determine the minimum execution time of a task (in case min > 0) if its input node is a timer (line 7 in Figure 5).

4.3. Formal Definition of BPMN-TPN Mapping

We will define a formal BPMN-TPN mapping based on the BPMN-PN mapping established in [8]. We enrich the latter (The Petri net $PN_M = (P_M, T_M, F_M)$) by adding the time dimension in the form of a time function that has as a domain the transition set. By Z we denote the set of well-formed core BPMN processes [8].

Fs: $T_M \rightarrow \mathbb{Q}^+ \times \mathbb{Q}^+ \cup \{\infty\}$ is a function assigning to each transition a rational interval. It remains to point out that the set of tasks of all well-formed core BPMN processes is a part of the resulting Petri net transitions. ($\cup_{P \in Z} T_P \subset T_M$)

- $\mathrm{Fs}(t_m) = [0, m]$ if $\exists \mathrm{P} \in \mathrm{Z}, \exists (e_t, g_x) \in E_P^{IT} \times G_P^X$ such as $t_m \in T_p$, $\mathrm{in}(t_m) \neq \varnothing$, $\mathrm{excp}(e_t) = t_m$, $\mathrm{Tim}(e_t) = m$ and $\mathrm{in}(g_x) = \{t_m, e_t\}$. (cf. Figure 5 line 4)
- $\mathrm{Fs}(t_m) = [0, 0]$ if $\exists P \in Z$ such as $t_m \in \{t_a \mid a \in G_P^F \cup G_P^J\} \cup \{t_{(\mathrm{a,\,b})} \mid \mathrm{a} \in G_P^X, b \in \mathrm{out}(a)\} \cup \{t_{(\mathrm{a,\,b})} \mid a \in G_P^M, b \in \mathrm{in}(a)\} \cup \{t_a \mid a \in E_P^S\} \cup \{t_b \mid b \in E_P^E\}$ (cf. Figure 4)
- $\mathrm{Fs}(t_m) = [n, m]$ if $\exists P \in Z, \exists (e_t, e_{t'}, g_x) \in (E_P^{IT})^2 \times G_P^X$ such as $t_m \in T_p$, $\mathrm{excp}(e_t) = t_m$, $\mathrm{Tim}(e_t) = m$, $\mathrm{in}(t_m) = e_{t'}$, $\mathrm{in}(g_x) = \{t_m, e_t\}$ and $\mathrm{Tim}(e_{t'}) = n$, with $n \leq m$. (cf. Figure 5 line 6)
- $\mathrm{Fs}(t_m) = [n, \infty]$ if $\exists P \in Z, \exists e_t \in E_P^{IT} \cup E_P^{ST}$ such as $t_m \in T_p$, $\mathrm{in}(t_m) \neq \varnothing$, $\mathrm{Tim}(e_t) = n$ and $\mathrm{in}(t_m) = e_t$. (cf. Figure 5 line 7)
- $\mathrm{Fs}(t_m) = [0, \infty]$ if $\exists P \in Z$, $t_m \in T_p$ and $\forall e_t \in E_P^{IT} \cup E_P^{ST}$ $\mathrm{in}(t_m) \neq e_t$ and $\mathrm{excp}(e_t) \neq t_m$. (cf. Figure 5 line 3)

After the translation of BPMN models into TPN, verification is done to ensure that some functional properties are satisfied by the model under investigation, such as liveness and marking reachability. In the next section we present a new algorithm that helps us verify these properties.

5. Algorithm for Liveness and Reachability Analysis of TPN

The algorithm used to check the liveness (resp reachability) of TPN’s transitions (resp markings) is shown in Algorithm 1. The algorithm takes as input the TPN and returns a Boolean value for each transition that says whether or not the transition is live. The algorithm starts by calculating the initial state of the TPN and initializing all the variables to be used, namely RS (the set of reachable states) and HS (the set of the handled states among RS) as well as TF (the set of firable transitions).

Then, it goes through all the states in RS and handles all the outgoing transitions from each of these states. Indeed, for each reachable state, the algorithm checks all of the outgoing transitions from the marking of the state and verifies if they are firable by calculating the minimum lft of all enabled transitions (let us have $A = \{t \in T_M \,/\, V_Z(t) \neq \omega\}$ and $T_d \subseteq A$ such as $T_d = \{t_s \in A \,/\, \mathrm{lft}(t_s) = \min(\mathrm{lft}(t))_{(t\,\in A)}\}$) then the minimum eft of all enabled transitions except for the ones in T_d ($\min(\mathrm{eft}(t))_{(t \in A/Td)} = e$). If e is greater than l, *then the Td‘s transitions are the only firable transitions* and added to TF, and the resulting state is calculated and added to RS if it is not already there. Otherwise, *All the enabled transitions are firable*, they are added to TF and the resulting state for each transition is calculated and added to RS if it is not already there. When the algorithm terminates the handling of all the reachable states (i.e., all the states in RS), it goes through all the transitions to check if they have been marked so far, if a transition has not been marked then the transition is declared dead.

Algorithm 1. Our algorithm for the reachability analysis of a TPN.

```
**Reachability Analysis (INPUT: TPN)**
$Z_0 = (M_0, V_0)$ with for each transition $t \in T_M$, $V_0(t) = 0$ if t is enabled, $\omega$ otherwise
RS $\leftarrow Z_0$ // RS is the set of reachable states.
RY $\leftarrow M_0$ // RY is the set of reachable markings.
HS $\leftarrow \varnothing$ // HS is the set of handled states.
TF $\leftarrow \varnothing$ // TE is the set of firable transitions.
Function Pre(t) // the function that returns the set of preplaces of a transition t.
While (RS $\neq$ HS)
    Take $Z = (M,V) \in RS$
    Add Z to HS
    $A \leftarrow \{ t \in T_M / V_Z(t) \neq \omega \}$
    Calculate $\min (lft(t))_{(t \in A)} = l \in Q^+ \cup \{\infty\}$
    $T_d \leftarrow \{t_s \in A / lft(t_s) = l\}$ and $T_G \leftarrow A / T_d$
    Calculate $\min (eft(t))_{(t \in TG)} = e$
    If $l < e$ then
        $B \leftarrow T_d$ // B is a variable
    Else
        $B \leftarrow A$
    End if
    For each $(t \in B)$
        t is firable $(M, V) \xrightarrow{t} (M', V')$
        $M' = M + W(O(t)) - W(I(t))$
        Add $Z'=(M',V')$ to RS, t to TF and M' to RY
        For each $(t' \in T_M)$
            If $W(I(t')) > M'$ then $V'(t') = \omega$
            End if.
            If($M \geq W(I(t'))$ and $M' \geq W(I(t'))$ and $pre(t) \cap pre(t') \neq \varnothing$ then $V'(t') = V(t')$
            Else
                $V'(t') = 0$
            End if
        End for
    End for
End while.
For each $(t \in T_M / TF)$
    t is dead. // the notion of transition death as explained in the fourth section.
End for
```

To illustrate the advantages of TPN as semantics as well as the algorithm elaborated above, we consider again the example of BPMN model in Figure 2. The resulting TPN obtained by applying our approach is shown in Figure 6. A simple investigation of the obtained TPN makes it possible to notice that a mistake has been made in modeling the fourth specification of the visa application processing business process in Figure 2. Indeed, the second timer in the BPMN model that is linked to the activity **"Verify documents and status"** can never be triggered because the maximum time this action can last is 13 days; therefore the visa applicant will never be notified if he/she is not accepted. To make this analysis formal, we will base our study on the TPN semantics defined above, we can verify the liveness and reachability of transitions and states as follows. By executing the Algorithm 1 detailed above, we get results as shown in Table 1.

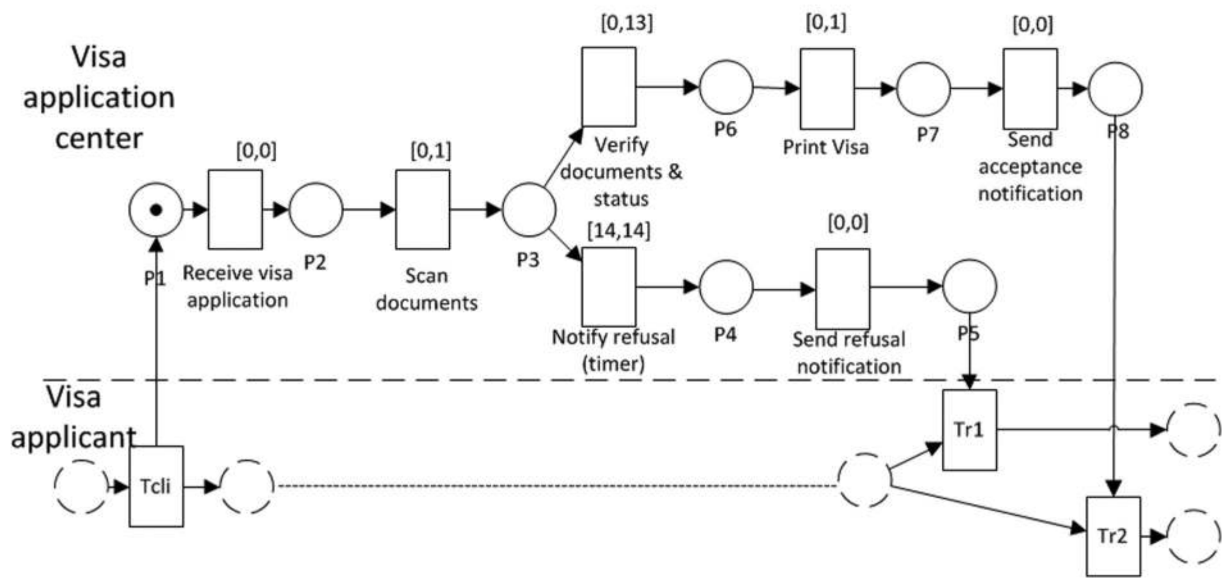

Figure 6. The TPN corresponding to the business process in Figure 2.

Table 1. Results of Algorithm 1.

Previous state	Firable transitions	Next state
[(1,0,0,0,0,0,0,0), (0,ω,ω,ω,ω,ω,ω)]	Receive visa application	[(0,1,0,0,0,0,0,0), (ω,0,ω,ω,ω,ω,ω)]
[(0,1,0,0,0,0,0,0), (ω,0,ω,ω,ω,ω,ω)]	Scan documents	[(0,0,1,0,0,0,0,0), (ω,ω,0,0,ω,ω,ω)]
[(0,0,1,0,0,0,0,0), (ω,ω,0,0,ω,ω,ω)]	Verify documents & status	[(0,0,0,0,0,1,0,0), (ω,ω,ω,ω,ω,0,ω)]
[(0,0,0,0,0,1,0,0), (ω,ω,ω,ω,ω,0,ω)]	Print visa	[(0,0,0,0,0,0,1,0), (ω,ω,ω,ω,ω,ω,0)]
[(0,0,0,0,0,0,1,0), (ω,ω,ω,ω,ω,ω,0)]	Send acceptance notification	[(0,0,0,0,0,0,0,1), (ω,ω,ω,ω,ω,ω,ω)]

In consequence, all transitions are firable, except for **"Notify refusal (timer)"** and **"Send refusal notification" (dead transitions)** as the earliest firing time of refusal timer is 14. Regarding reachability, the markings (0,0,0,1,0,0,0,0) and (0,0,0,0,1,0,0,0) will never be reached.

6. Conclusion

In this paper, we proposed a formal analysis of BPMN models based on Time Petri Nets. The Mapping BPMN-TPN allows us to overcome some problems found in BPMN (2.0) (explained in the Section 3) and have a more complete analysis that verifies the temporal properties of the designed process. This analysis is significant because it reduces the cost of software development by allowing business analysts to detect some type of errors found in business process models (modeling phase) before starting the development phase. The visa application processing was taken as an example to illustrate the advantages brought by the TPN semantics.

In our future work, we intend to include the data aspect in our analysis, using a more complex semantics that is based on ITCPN (Interval Timed Colored Petri Net). Contrary to previous work [18] that dealt only with data dimension, ITCPN will allow us to include Data and Time in our verification, so we could have a more complete analysis.

References

[1] M. Indulska *et al.*, "Business Process Modeling: Perceived Benefits", in *Proc. the 28th Int Conf on Conceptual Modeling*, Gramado, Brazil, 2009, pp. 458–471. http://dx.doi.org/10.1007/978-3-642-04840-1_34

[2] Business Process Modeling Notation (BPMN) Specification, OMG, Needham, USA 2011.

[3] Web Services Business Process Execution Language. OASIS., Burlington, USA, 2007.

[4] C. Ouyang *et al.*, "Translating BPMN to BPEL", BPM Center Report, Brisbane, BPM-06-02, 2006.

[5] L. García-Bañuelos "Pattern Identification and Classification in the Translation from BPMN to BPEL", in *Proc. the Confederated Int. Conf., CoopIS, DOA, GADA, IS, and ODBASE*, Monterrey, Mexico, 2008, pp. 436–444.

[6] F. Kossak *et al.*, *A Rigorous Semantics for BPMN 2.0 Process Diagrams*, Switzerland Springer Int Pub, 2014.
http://dx.doi.org/10.1007/978-3-319-09931-6

[7] G. Aagesen and J. Krogstie, "BPMN 2.0 for Modeling Business Processes" in *Handbook on Business Process Management*, vol. 1, Berlin, Germany, Springer, 2015, pp. 219–250.

[8] M. Dijkman *et al.*, "Formal semantics and analysis of BPMN Process Models using Petri Nets", Brisbane, Tech. Univ. QLD, Technical Report 7115, 2007.

[9] A. En-Nouaary and R. Dssouli, "A Practical Method for the Reachability Analysis of Real-Time Systems Modelled as Timed Automata", in *6th Int Conf on Software Engineering Advances*, Barcelona, Spain, 2011.

[10] P. Y. H. Wong and J. Gibbons, "A Process Semantics for BPMN", in *the Proc. 10th Formal Engineering Methods Int. Conf. ICFEM*, Kitakyushu, Japan, 2008, pp. 355–374.
http://dx.doi.org/10.1007/978-3-540-88194-0_22

[11] J. Ye *et al.*, "Transformation of BPMN to YAWL", in *the Proc. Computer Science and Software Engineering Int. Conf*, Washington, DC, USA, 2008, pp. 354–359.
http://dx.doi.org/10.1109/csse.2008.980

[12] C. Ouyang *et al.*, "From Business Process Models to Process-Oriented Software Systems", *ACM Trans. Software Engineering and Methodology*, vol. 19, pp. 1–37, Aug 2009.
http://dx.doi.org/10.1145/1555392.1555395

[13] Object Management Group. (2011, January).[Online]. Available: http://www.bpmn.org/

[14] D. Prandi *et al.*, "Formal Analysis of BPMN via a Translation into Cows", in *the Proc. 10th COORDINATION Int Conf ICFEM*, Oslo, Norway, 2008, pp. 249–263.
http://dx.doi.org/10.1007/978-3-540-68265-3_16

[15] C. A. Petri, "Kommunikation mit Automaten", Ph.D dissertation, Darmstadt Univ, 1962.

[16] J. Wang, "Petri Nets for Dynamic Event-Driven System Modeling" in *Handbook of Dynamic System Modeling*, vol. 1, London, UK, Chapman and Hall/CRC, 2007.

[17] L. Popova, "On Time Petri Net", *Journal of Information Processing and Cybernetics EIK*, vol. 27, pp. 227–244, 1991.

[18] S. Stackelberg *et al.*, "Detecting Data-Flow Errors in BPMN 2.0", *Open Journal of Information Systems*, vol.1, pp. 1–19, 2014.

Received: August, 2015
Revised: December, 2015
Accepted: December, 2015

Contact addresses:

Anass Rachdi
Institut National des Postes
et Télécommunications (INPT)
Rabat, 10112, Morocco
e-mail: anass.rach@gmail.com

Abdeslam En-Nouaary
Institut National des Postes
et Télécommunications (INPT)
Rabat, 10112, Morocco
e-mail: abdeslam@inpt.ac.ma

Mohamed Dahchour
Institut National des Postes
et Télécommunications (INPT)
Rabat, 10112, Morocco
e-mail: dahchour@inpt.ac.ma

ANASS RACHDI received his Diploma (engineer degree) in Telecommunication Option Information Systems for Management from INPT "Institut National des Postes et Télécommunication" Rabat, Morocco, in 2011. He is a PhD student in the doctoral center of INPT. His current research topics are workflow management, process analysis, data and privacy in workflows. He has a modest experience in project work, as well as with industry.

ABDESLAM EN-NOUAARY received his engineer degree in Computer Engineering, Option Data Communication and Computer Networks, from ENSIAS (École Nationale Supérieure d'Informatique et d'Analyse des Systèmes), Rabat, Morocco, in 1996, the M. Sc. and Ph. D. degrees in Computer Science from the University of Montreal in 1998 and 2001 respectively. Dr. En-Nouaary is currently an Associate Professor at INPT (Institut National des Postes et Télécommunications), Rabat, Morocco. Before joining INPT in 2008, Dr. En-Nouaary has been an Associate Professor at the Electrical and Computer Engineering Department of Concordia University, Montreal, Canada, From 2001 to 2008. His main research interest are modeling and validation of distributed, real-time, and embedded systems.

MOHAMED DAHCHOUR received the doctoral (2001) in Computer Science from Ecole polytechnique de Louvain, Université catholique de Louvain, Belgium. Currently, he is full professor of computer science at the National Institute of Posts and Telecommunications (INPT), Morocco. He is also the head of the Department of Mathematics and Informatics at INPT and the leader of a research team on software systems engineering and information systems management. His scientific interests include software engineering, conceptual modeling, web semantic, distributed systems, and information systems management.