Visual Specification Language and Automatic Checking of Business Process

Outman El Hichami
Faculty of Sciences, UAE,
Tetouan, Morocco
el.hichami.outman@taalim.ma

Mohammed Al Achhab
National School of Applied Sciences,
Tetouan, Morocco
alachhab@ieee.ma

Ismail Berrada
Faculty of Sciences, USMBA,
Fez, Morocco
iberrada@univ-lr.fr

Badr Eddine El Mohajir
Faculty of Sciences, UAE,
Tetouan, Morocco
b.elmohajir@ieee.ma

In this work we propose a visual language for specifying behavioral properties of business processes (*BP*). We use Business process modeling notation (*BPMN*) to modelize *BP*, Petri Net as underlying formal foundations, and SPIN model checker to validate the dynamic behaviors of this process. The objective of this paper is to propose graphical property specification language which can be used during the design phase of *BP*. The proposed visual language uses the same concepts as established in *BPMN* to specify the properties to be verified. A semantic interpretation for properties expressed is given based en temporal logic formulas. The advantage of the proposed language is that it hides the temporal logic used for the specification of properties, and the knowledge of this logic is not needed.

Business process, Model-checking, BPMN, Petri Net, Dynamic behavior

1. INTRODUCTION

In recent years, researchers have become increasingly interested in developing methods and tools for the specification and validation of Business Processes (*BP*) behavior. The Business Process Modeling Notation *BPMN* OMG (2011) is emerging as a widely accepted approach in the domain of Business Process Management Hofstede et al. (2003) and becoming increasingly indispensable in business relationship, web services AbuJarour and Awad (2014), etc. *BPMN* is a standard and a well-known diagrammatic notation for supporting the specification of *BP*. It provides symbols to a simple graphical specification for defining the control flow of the business process in the early phases of process development. *BPMN* diagrams afford a notation that is readily understandable by all business users: the designer expert, the technical developers, the business people who will manage and monitor these processes.

The mix of constructs found in *BPMN* and the lack of an unambiguous definition of some notations, makes it possible to create models with semantic errors. Therefore, several approaches have been proposed to the formal validation of *BPMN* Takemura (2008); Al Achhab et al. (2014); Bryans et al. (2009); Van der Aalst and Van Dongen (2013); Fahland et al. (2011). Most of these approaches are based on the mapping of *BPMN* to a formal presentation like Petri Net Murata and Koh (1989); El Hichami et al. (2014), YAWL Sun et al. (2008), PROMELA[1], and PNML Hillah et al. (2010), in order to use the formal analysis tools available for these models. When model-cheking is considered for formally verifying *BP* properties, the specifications of this properties should be expressed by temporal logic formulas. Temporal properties are not always easy to write or read and need strong background knowledge.

The objective of this paper is to propose a user-friendly graphical interface that business experts can use to specify and verify business processes. Furthermore, this approach allows the integration of formal verification techniques of *BPMN* models in the design phase.

The intent of this paper is to collect properties (patterns) that occur commonly in the specification of *BP*. Most specification formalisms in this domain are a bit tricky to use. To make them easier to use, our patterns use the same concepts as established

[1] spinroot.com/spin/Man/promela.html

in *BPMN* and come with descriptions that illustrate how to map well-understood conceptions of *BP* behavior into precise statements in common formal specification languages like linear temporal logic (*LTL*) MANNA and PNUELI (1992) and computation tree logic (*CTL*) Heljanko (1997). The rest of the paper proceeds as follows: *Section 2* discusses the related work. *Section 3* provides formal definitions and notations of *BPMN* used in the rest of this paper and the mapping from *BPMN* modules to Petri Net. *Section 4* describes a graphical property specification language. *Section 5* describes our verification process of *BP* and case study. We develop experiments and analysis in *Section 6*. *Section 7* concludes the paper, and draws perspectives.

2. RELATED WORK

Current research in this area are largely concentrated on a translation of *BPMN* models to a formal language, and the specification of the proprieties is written in temporal logic, and does not consider a visual language for specifying these proprieties to be verified. In Dijkman et al. (2007), the authors introduce an approach, based on Petri Net, to formalize and analyze *BPMN* while abstracting data information. However, they only consider safety properties of *BPMN*. In Van der Aalst et al. (2007), the authors have developed a tool, called *Prom*[2], to verify *BPMN* models. The property specification logic supported by Prom, is restricted to the linear temporal logic (LTL). In Sakr et al. (2013), the authors have implemented a concept `proof` of their approach with existing software namely the open modeling platform *Oryx* Decker et al. (2008) and the *BPMN-Q* query language Awad (2007). This approach is based on the decomposition of *BPMN-Q*. However, This verification approach may fail because not all properties of a query can be satisfied and the decomposition phase is complicated.

To the best of our knowledge, all the above works assume that the verification phase comes after the design phase. Thus, a strong background knowledge of temporal logic is required. Our approach has the merit of integrating the verification process in the design stage allowing a gradual validation of *BP*. This phase can be avoided by implementing a specification interface.

3. BASIC DEFINITIONS

In this section, we give the basic definitions, notations of *BPMN*, and Petri Net used in this paper.

[2]http://www.promtools.org/prom6/

3.1. BPMN

BPMN is a graphical notation designed for both business process design and business process implementation. *BPMN* process models are composed of:

1. Events:
   (a) Start Event: it indicates where a particular process will start;
   (b) End Event: it indicates where a process will end.
2. Task: is a generic type of work to be done during the course of a *BP*.
3. Sequence flow: it links two objects in a process diagram.
4. Gateways:
   (a) And-split Gateway: is where a single branch is divided into two or more parallel branches which are executed concurrently;
   (b) And-join Gateway: is where two or more different branches of the process merge into one single branch;
   (c) Or-split Gateway: it routes the sequence flow to exactly one of the outgoing branches based on conditions;
   (d) Or-join Gateway: it awaits one incoming branch to complete before triggering the outgoing flow.

Fig. 1 provides an overview of a subset of *BPMN* elements related to control-flow specification, these include sequence flows. An object can be an event, activity or gateway. A sequence flow links two objects in a process diagram and denotes a control flow relation.

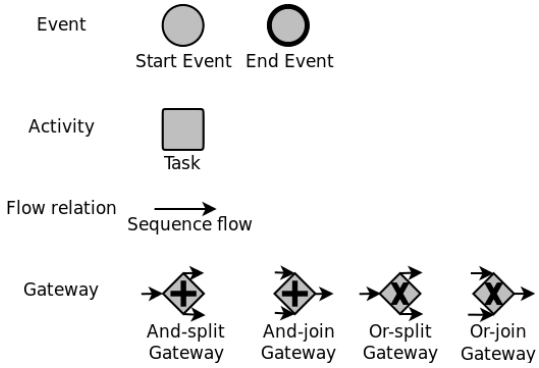

Figure 1: *The basic elements of* BPMN

3.1.1. Example

For further clarification, we give a simple example adapted from Raedts et al. (2007) of the recruitment process. Fig. 2 illustrates the *BPMN* model of the hiring process since the creation of the job until the candidate is rejected or accepted.

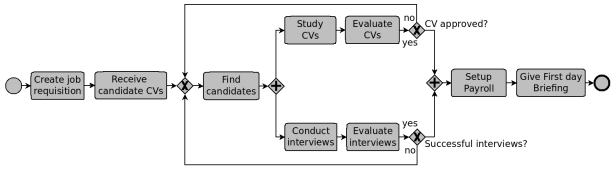

Figure 2: BPMN *model of the recruitment process*

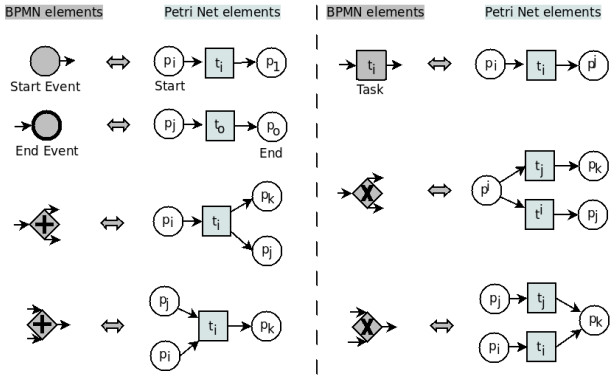

Figure 3: *Mapping* BPMN *to Petri Net*

3.2. Petri Net

Petri Net is widely used tool for the representation, validation and verification of *BP* Van der Aalst (1997, 1998); Barkaoui et al. (2007). A Petri Net is a tuple $N = (P, T, F)$ where:

1. $P \neq \emptyset$ is a finite set of places;
2. $T \neq \emptyset$ is a finite set of transitions with $P \cap T = \emptyset$;
3. $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation.

A place can contain zero or more tokens. A token is represented by a black dot. The global state of a Petri Net, also called a marking, is the distribution of tokens over places. Formally, a marking of a Petri Net N is a function $M : P \to \mathbb{N}$. The initial marking of N is denoted by M_0.
An example of Petri Net is shown in Fig. 4.

3.3. Transforming BPMN models to Petri Net

In order to analyze formally *BPMN* models, several transformations have been proposed Lohmann et al. (2009); Dijkman et al. (2008). Fig. 3 depicts the mapping from *BPMN* tasks, events, and gateways to Petri Net modules proposed by Dijkman et al. (2008). A task or an intermediate event is mapped onto a transition with one input place and one output place. The transition, being labelled with the name of that task (respectively event), models the execution of the task (respectively event). A start or end event is mapped onto a similar module except that a silent transition is used to signal when the process starts or ends.

The Petri net, representing the recuitment process of Fig. 2, that is produced by applying the mapping rules mentioned above is given in Fig. 4.

4. A VISUAL LANGUAGE FOR BP PROPERTY SPECIFICATION (BPVSL)

In this section, we show how the designer can specify the properties to be verified using a graphical interface based on the same concepts as established in *BPMN*. The framework uses this specification as a guide to implement the transformation to LTL MANNA and PNUELI (1992) or CTL Heljanko (1997) temporal logic, even though the designer has no notions of these temporal logic languages.

Before illustrating the BPVSL that we propose, we give the set of properties that may be needed by the designer:

- Safety property: that *"nothing bad"* will happen, ever, during the execution of a system like the absence of dead transitions (deadlocks).
- Liveness property: that *"something good"* will happen, eventually, during the execution of a system like the absence of cyclical behaviors.
- Fairness property: under certain conditions, an event may occur repeatedly, is a special type of a safety property.
- Invariant property: is a special case of a safety property. This type of property is satisfied with all system states.
- Response property: if action A occurs then eventually action B will occur.

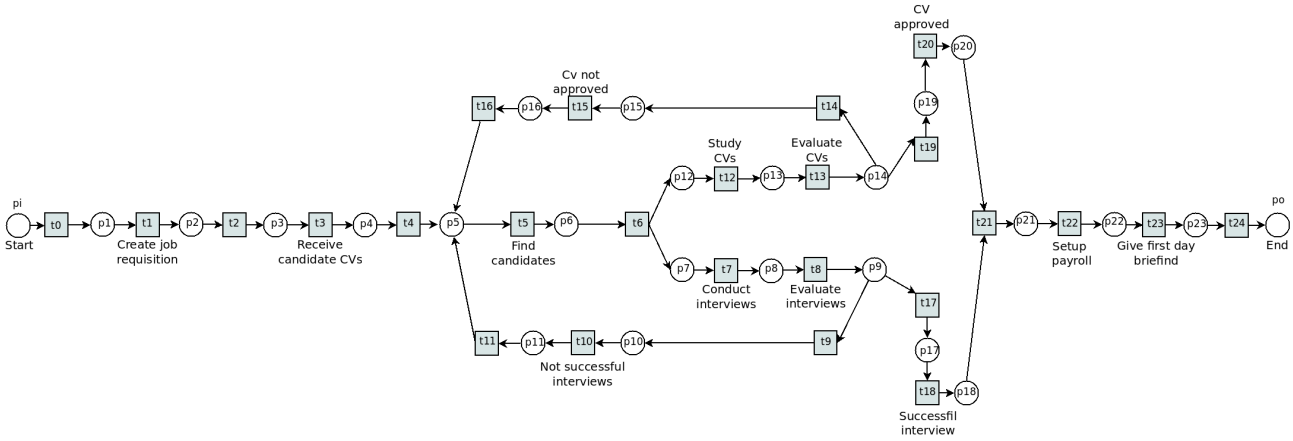

Figure 4: *Petri Net model for the recruitment process*

4.1. Response Properties

To specify the dynamic behavior of a *BP*, five models of response properties (Fig. 5) are considered in our *BP* Visual Specification Language (BPVSL).

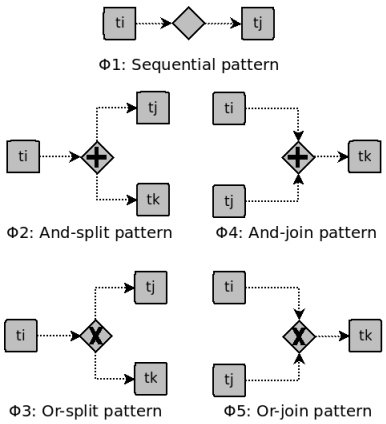

Figure 5: *Patterns of response properties*

The following sections provide some of the most frequently interpretation of this patterns. The designer can use a graphical interface to specify the source and the target extremities of property to be verified. Then, the framework proposes the collection of the gateways and arrow types in order to choose the desirable semantic.

4.2. Linear temporal logic (*LTL*)

Temporal logic as extension of boolean logic may be used as formal language to express the properties that must be satisfied by the runs of a *BPMN* model. *LTL* is the logic we use in this paper.

The syntax of *LTL* is inductively defined as:

$\Phi ::= p|\neg|\Phi|\Phi \wedge \Phi|\Phi \vee \Phi|\Phi \rightarrow \Phi|\bigcirc \Phi|\Diamond\Phi|\Box\Phi|\Phi U \Phi$. Such p is a atomic proposition (task in *BPMN*). The intuitive meanings of the associated *LTL* formulas are given below:

- $\bigcirc\Phi$: means Φ is true in next state;
- $\Diamond\Phi$: means Φ is true in some future state;
- $\Box\Phi$: means Φ is true in all future states;
- $\Phi 1 U \Phi 2$: means $\Phi 1$ is true in all future states until $\Phi 2$ holds.

4.3. Computation tree logic (*CTL*)

The syntax of *CTL* is inductively defined as:

$$\Phi ::= p|\neg|\Phi|\Phi \wedge \Phi|\Phi \vee \Phi|\Phi \rightarrow \Phi.$$

Thus, in addition to introducing temporal operators, it introduces for-all and existential quantifiers:

- $A\Box\Phi$: means Φ has to hold in every future state on every execution path;
- $E\Box\Phi$: means Φ has to hold in every future state on some execution path;
- $A\Diamond\Phi$: means Φ has to hold in some future state in every execution path;
- $E\Diamond\Phi$: means Φ has to hold in some future state in some execution path.

For more details see MANNA and PNUELI (1992); Heljanko (1997).

4.4. Specification language and semantic of $\Phi 1$

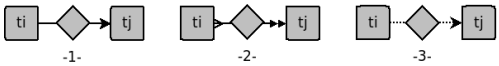

Figure 6: *Graphical specification of* $\Phi 1$

Formal semantics of $\Phi 1$:

- Every time t_i is executed, t_j has to be executed afterwards: $\Box(t_i \Rightarrow \Diamond t_j)$, (LTL formula).
- All paths from t_i to t_j : $(t_i \Rightarrow A\Diamond t_j)$, (CTL formula).

- Every time t_i is executed, t_j has to be executed afterwards: $\Box(t_i \Rightarrow \Diamond t_j)$, (LTL formula).

4.5. Specification language and semantic of $\Phi 2$

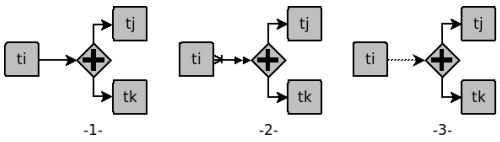

Figure 7: *Graphical specification of* $\Phi 2$

Formal semantics of $\Phi 2$:

- Every time t_i is executed, t_j and t_k have to be executed in parallel afterwards: $\Box(t_i \Rightarrow (\Diamond t_j \wedge \Diamond t_k))$, (LTL formula).
- When t_i is executed, t_j and t_k have to be executed afterwards, while the two outgoing branches are activated in parallel, each branch on all potential paths: $(t_i \Rightarrow (A\Diamond t_j \wedge A\Diamond t_k))$, (CTL formula).
- Every time t_i is executed, t_j and t_k have to be executed in parallel afterwards: $\Box(t_i \Rightarrow (\Diamond t_j \wedge \Diamond t_k))$, (LTL formula).

4.6. Specification language and semantic of $\Phi 3$

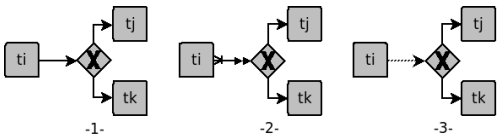

Figure 8: *Graphical specification of* $\Phi 3$

Formal semantics $\Phi 3$:

- Every time t_i is executed, one of the tasks t_i or t_k has to be executed afterwards: $\Box(t_i \Rightarrow (\Diamond t_j \vee \Diamond t_k))$, (LTL formula).
- One of the tasks t_j or t_k eventually is executed after the task t_i on each potential path: $(t_i \Rightarrow (A\Diamond t_j \vee A\Diamond t_k))$, (CTL formula).
- Every time t_i is executed, one of the tasks t_i or t_k has to be executed afterwards: $\Box(t_i \Rightarrow (\Diamond t_j \vee \Diamond t_k))$, (LTL formula).

4.7. Specification language and semantic of $\Phi 4$

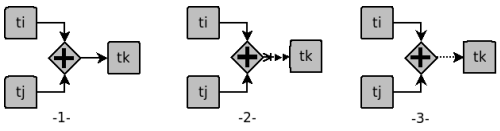

Figure 9: *Graphical specification of* $\Phi 4$

Formal semantics $\Phi 4$:

- Every time tasks t_i and t_j are simultaneously executed, t_k has to be executed afterwards: $\Box((t_i \wedge t_j) \Rightarrow \Diamond t_k)$, (LTL formula).
- When merging parallel branches, outgoing branch on all potential paths: $((t_i \wedge t_j) \Rightarrow A\Diamond t_k)$, (CTL formula).
- Every time tasks t_i and t_j are simultaneously executed, t_k has to be executed afterwards: $\Box((t_i \wedge t_j) \Rightarrow \Diamond t_k)$, (LTL formula).

4.8. Specification language and semantic of $\Phi 5$

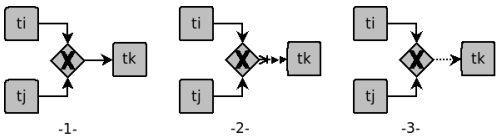

Figure 10: *Graphical specification of* $\Phi 5$

Formal semantics $\Phi 5$:

- Every time one of the tasks t_i or t_j is executed, it is followed by the task t_k: $\Box((t_i \vee t_j) \Rightarrow \Diamond t_k)$, (LTL formula).
- One of the tasks t_i or t_j will be eventually followed by the task t_k on each potential path: $((t_i \vee t_j) \Rightarrow A\Diamond t_k)$, (CTL formula).
- Every time one of the tasks t_i or t_j is executed, it is followed by the task t_k: $\Box((t_i \vee t_j) \Rightarrow \Diamond t_k)$, (LTL formula).

4.9. Safety and Liveness properties

In order to meet the requirements of the designer, we propose to add for each property to be verified a sets of properties in order to define safety, liveness, invariant and fairness properties.

4.9.1. A safety property

are conditions that are verified along any execution path. These type of properties are usually associated with some critical behaviour, thereby they should *always* hold. Then, the quantifier ($\Box$) is good for the safety property. For example, in the case of *BPMN* model of the recruitment process, the company never recruits a “bad candidate”.

4.9.2. A liveness property

These type of properties involved in the temporal concept with *eventually*. Thus, the quantifier ($\Diamond$) is good for the liveness property. Practically, in the case of *BPMN* model of the recruitment process, never reach the “good candidate”.

5. VERIFICATION PROCESS AND CASE STUDY

The verification process proposed of *BP* (Fig. 11), the designer uses a graphical interface to modelize the *BPMN* and to specify the desired properties to be verified using BPVSL. The framework uses this specification as a guide to implement the transformation to corresponding LTL temporal logic.

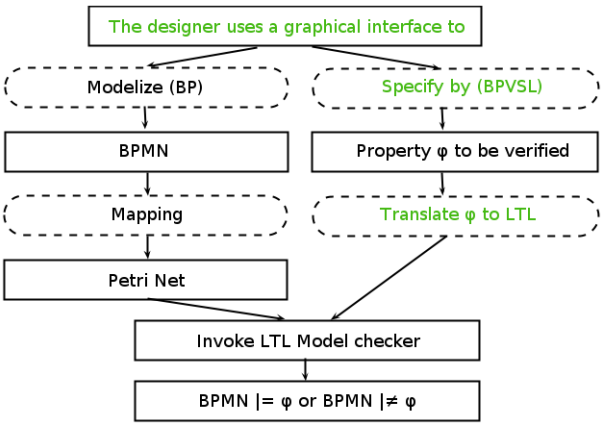

Figure 11: *Verification process of* BPMN

As examples, we show how to specify three response properties using BPVSL.

Example 1: We could use the specification language semantics of $\Phi 1$ to specify the following property ($\phi 1$): *Does the task "Setup payroll" will happen after the tasks "Find candidates"* This property can be interpreted with the visual presentation of Fig. 12. When the source extremity of $\phi 1$ is selected, the framework proposes only the reachable tasks as target extremities of $\phi 1$. Then, the designer can specify the arrow types between the task *"Find candidates"* and *"Setup payroll"*.

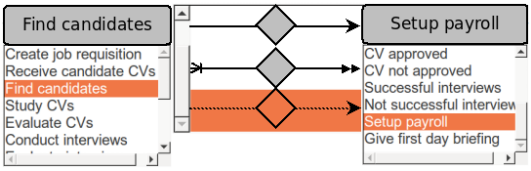

Figure 12: *Visual specification of* $\phi 1$

Example 2: Let $\phi 2$ be the property given by: *the task "Study CVs" and task "Evaluate interviews" will be executed in parallel and after task "Receive candidate CVs"*. This property can be interpreted with the visual presentation of Fig. 13. In fact, the designer has first to select a task "Receive candidate CVs", then the designer has to select only the reachable tasks *"Study CVs"* to be executed in parallel with the task *"Evaluate interviews"* as the target extremities of $\phi 2$. Finally, the framework proposes the collection of the gateways and arrow types in order to choose between the different semantics of $\phi 2$.

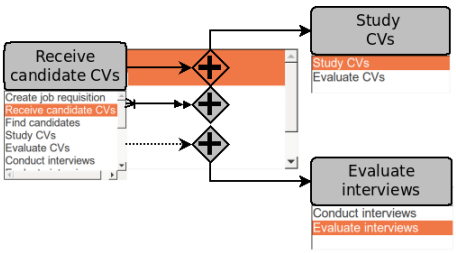

Figure 13: *Visual specification of* $\phi 2$

Example 3: Let $\phi 3$ be the property given by: *Task "Find candidates" will be executed after the tasks "CV not approved" or "Not successful interviews"*. This property can be interpreted with the visual presentation of Fig. 14. Similar to the specification of $\phi 1$ and $\phi 2$, when the first source extremity *"CV not approved"* of $\phi 3$ is selected, the designer selects the task *"CV not approved"* as the second source extremity of $\phi 3$ to be executed in conditional with the task *"CV not approved"*, and the reachable task *"Find candidates"* as the target extremity of $\phi 3$. Finally, the designer can choose between the collection of the gateways and arrow types.

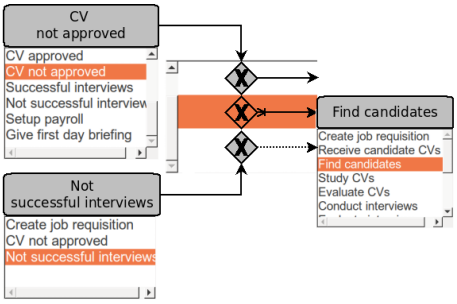

Figure 14: *Visual specification of* $\phi 3$

6. EXPERIMENTS AND ANALYSIS

In this paper, we use the SPIN tools[3] to validate the LTL formulas. In these experiments, we discuss the verification of three models of response properties $\Phi 1$, $\Phi 2$ and $\Phi 3$ of *Section 5*.

6.1. Transforming Petri Net to PROMELA

In order to simulate Petri Net using SPIN, it is necessary to translate Petri Net models into SPIN models-specified using PROMELA.

In Holzmann (2003), the authors propose a method to describe a Petri Net into PROMELA that can be simulated and verified with the SPIN model checker. In this method, a Petri Net system is represented

[3]http://spinroot.com/spin/Man/

as a single process. The process describes each firing of its transitions. An outline of the PROMELA program of the Petri Net corresponding to the recruitment process is given bellow:

```
#define Place 25
#define Transition 25
int M[Place]; /* Marking */
int X[Transition]; /* Firing count */
/* A firing of Transition t*/
/* remove specifies the change of the
   marking of preset of t */
/* add specifies the change of the marking
   of postset of t */
/* fire (x) increments the element
   corresponding to t in X[t] */
#define remove1(x) (x>0) -> x--
#define remove2(x,y) (x>0 && y>0)
-> x--; y--
#define add1(x) x++
#define add2(x,y) x++; y++
#define fire(x) x++
/* Process representing Petri Net */
init
{
M[0]=1; /* Set the initial marking */
do
:: atomic{remove1(M[0])  -> fire(X[0]);
   add1(M[1]}
:: atomic{remove1(M[1])  -> fire(X[1]);
   add1(M[2]}
:: atomic{remove1(M[2])  -> fire(X[2]);
   add1(M[3]}
:: atomic{remove1(M[3])  -> fire(X[3]);
   add1(M[4]}
:: atomic{remove1(M[4])  -> fire(X[4]);
   add1(M[5]}
:: atomic{remove1(M[5])  -> fire(X[5]);
   add1(M[6]}
:: atomic{remove1(M[6])  -> fire(X[6]);
   add2(M[7], M[12])}
:: atomic{remove1(M[7])  -> fire(X[7]);
   add1(M[8]}
:: atomic{remove1(M[8])  -> fire(X[8]);
   add1(M[9]}
:: atomic{remove1(M[8])  -> fire(X[8]);
   add1(M[17]}
:: atomic{remove1(M[9])  -> fire(X[9]);
   add1(M[10]}
:: atomic{remove1(M[10]) -> fire(X[10]);
   add1(M[11]}
:: atomic{remove1(M[11]) -> fire(X[11]);
   add1(M[5]}
:: atomic{remove1(M[12]) -> fire(X[12]);
   add1(M[13]}
:: atomic{remove1(M[13]) -> fire(X[13]);
   add1(M[14]}
:: atomic{remove1(M[13]) -> fire(X[13]);
   add1(M[19]}
:: atomic{remove1(M[14]) -> fire(X[14]);
   add1(M[15]}
:: atomic{remove1(M[15]) -> fire(X[15]);
   add1(M[16]}
:: atomic{remove1(M[16]) -> fire(X[16]);
   add1(M[5]}
:: atomic{remove1(M[9])  -> fire(X[17]);
   add1(M[17]}
:: atomic{remove1(M[17]) -> fire(X[18]);
   add1(M[18]}
:: atomic{remove1(M[14]) -> fire(X[19]);
   add1(M[19]}
:: atomic{remove1(M[19]) -> fire(X[20]);
   add1(M[20]}
:: atomic{remove2(M[18],M[20]) ->
   fire(X[21]); add1(M[21])}
:: atomic{remove1(M[21]) -> fire(X[22]);
   add1(M[22]}
:: atomic{remove1(M[22]) -> fire(X[23]);
   add1(M[23]}
:: atomic{remove1(M[23]) -> fire(X[24]);
   add1(M[24]}
od
}
```

(The PROMELA description of Petri Net shown in Fig. 4)

6.2. LTL formulas

The properties to be verified in SPIN have to be expressed as LTL formulas. LTL formulas corresponding to to the response properties $\phi 1$, $\phi 2$ and $\phi 3$ to be verified can be rewritten as follows:

- $\phi 1$: *Does the task "Setup payroll" will happen after the tasks "Find candidates"*;
  LTL formula: $[\,]((M[5] >= 1) \; -> \; <> (M[21] >= 1))$;
- $\phi 2$: *Task "Study CVs" and task "Evaluate interviews" will be executed in parallel and after task "Receive candidate CVs"*;
  LTL formula: $[\,]((M[3] >= 1) \; -> \; <> (M[12] >= 1 \;\&\&\; M[8] >= 1))$;
- $\phi 3$: *Task "Find candidates" will be executed after the tasks "CV not approved" or "Not successful interviews"*.
  LTL formula: $[\,]((M[15] >= 1 \,||\, M[10] >= 1) -> \; <> (M[5] >= 1))$.

6.3. Experimental results

In this section, we give some statistics in order to show the performance of our approach. We present the size, the memory and the verification time of the verification of $\phi 1$, $\phi 2$ and $\phi 3$ on the Petri Net related to Fig. 4.

We first investigated in Table 1 and Table 2 the results without Safety properties and with Liveness properties.
In Table 3 we present the results with Safety properties(invalid deadlock) and without Liveness properties.

Table 1: *Experiments without Safety properties and with Liveness properties(acceptance cycles)*

	States	Transitions	Memory (Mb)	Times (s)
$\phi 1$	8310907	1.4110983e+08	1023.946	83.8
$\phi 2$	8311225	1.2663418e+08	1023.946	74
$\phi 3$	8311136	1.2658587e+08	1023.946	71.5

Table 2: *Experiments without Safety properties and with Liveness properties(acceptance cycles and enforce fairness property)*

	States	Transitions	Memory (Mb)	Times (s)
$\phi 1$	8059677	1.3712059e+08	1023.946	83.5
$\phi 2$	8059995	1.2277647e+08	1023.946	71.4
$\phi 3$	8047291	1.2276921e+08	1023.946	70.1

Table 3: *Experiments with Safety properties(invalid deadlock)*

	States	Transitions	Memory (Mb)	Times (s)
$\phi 1$	8059582	64672621	1023.946	27.2
$\phi 2$	8059582	57992022	1023.946	24.1
$\phi 3$	8004931	55834127	1023.946	25.6

7. CONCLUSION

The paper proposes a visual language for specifying business process behavior. We use *BPMN* to modelize *BP*, Petri Net as underlying formal foundations, and spin model checker to perform automated analysis. The principal objective of this paper is to propose a graphical framework which use the same notation as established in *BPMN*. In this way the designer can specify and validate the *BP* properties during the design phase. Several formal semantics for these properties are expressed as temporal logic formulas.

REFERENCES

ter Hofstede, A., van der Aalst, W., A., Weske, M. (2003) *Business process management: A survey. In Weske, M., ed.: Business Process Management.* Volume 2678 of Lecture Notes in Computer Science. Springer Berlin / Heidelberg.

Mohammed AbuJarour and Ahmed Awad. (2014) *Web Services and Business Processes: A Round Trip.* Web Services Foundations : pp. 3-29.

OMG. (2011) *Business Process Modeling Notation (BPMN) Version 2.0.* OMG Final Adopted Specification. Object Management Group.

T. Takemura. (2008) *Formal semantics and verification of BPMN transaction and compensation.* In Proc. of APSCC 2008, pp. 284-290. IEEE.

J. W. Bryans, J. S. Fitzgerald, A. Romanovsky, and A. Roth. (2009) *Formal modelling and analysis of business information applications with fault tolerant middleware.* In Proc. of ICECCS 2009, pp. 68-77. IEEE Computer Society.

O. El Hichami, M. Al Achhab, I. Berrada and B. El Mohajir. Graphical specification and automatic verification of business process, the International Conference on Networked systems. NETYS 2014, LNCS 8593, Springer, pp. 1-6.

W.M.P. van der Aalst and B.F. van Dongen. (2013) *Discovering Petri Nets From Event Logs.* T. Petri Nets and Other Models of Concurrency 7: 372-422.

Dirk Fahland, Cdric Favre, Jana Koehler, Niels Lohmann, Hagen Volzer, Karsten Wolf. (2011) *Analysis on demand: Instantaneous soundness checking of industrial business process models.* Data Knowl. Eng. 70(5): pp.448-466.

T. Murata and J.Y. Koh (1989) *Petri nets: Properties, Analysis and Applications.* an invited surve y paper, Proceedings of the IEEE, Vol.77, No.4 pp.541-580.

O. El Hichami, M. Al Achhab, I. Berrada, R. Oucheikh and B. El Mohajir. (2014) *An Approach Of Optimisation And Formal Verification Of Workflow Petri Nets.* Journal of Theoretical and Applied Information Technology, Vol.61, No.3 pp. 486-495.

J.-H. Ye, S.-X. Sun, L. Wen, and W. Song. (2008) *Transformation of BPMN to YAWL.* In CSSE (2), pp. 354-359. IEEE Computer Society.

L. Hillah, F. Kordon, L. Petrucci, and N. Trves. (2010) *PNML Framework: an extendable reference implementation of the Petri Net Markup Language,* LNCS 6128, pp. 318–327.

R. M. Dijkman, M. Dumas, and C Ouyang. (2007) *Formal semantics and analysis of BPMN process models using Petri nets.* Technical Report 7115, Queensland University of Technology, Brisbane.

W.M.P. van der Aalst, B.F. van Dongen, C.W. G nther, R.S. Mans, A.K. Alves de Medeiros, A. Rozinat, V. Rubin, M. Song, H.M.W. Verbeek, and A.J.M.M. Weijters. (2007) *ProM 4.0: Comprehensive Support for Real Process Analysis.* In J. Kleijn and A. Yakovlev, editors, Application and Theory of Petri Net and Other Models of Concurrency (ICATPN 2007), volume 4546 of LNCS, Springer-Verlag, Berlin, pp. 484-494.

Sherif Sakr, Ahmed Awad, Matthias Kunze. (2013) *Querying Process Models Repositories by Aggregated Graph Search,* in Springer Berlin Heidelberg. Volume 132, pp 573-585.

Decker, G., Overdick, H., Weske, M. (2008) *Oryx - Sharing Conceptual Models on the Web.* In: Conceptual Modeling - ER. LNCS 5231, Springer Verlag, pp. 536-537.

Awad, A. (2007) *BPMN-Q: A Language to Query Business Processes.* In EMISA, pp. 115-128.

Ivo Raedts, Marija Petkovic, Yaroslav S. Usenko, Jan Martijn E. M. van der Werf, Jan Friso Groote, and Lou J. Somers. (2007) *Transformation of BPMN Models for Behaviour Analysis.* MSVVEIS, INSTICC PRESS, pp. 126-137.

W.M.P. van der Aalst. (1997) *Verification of Workflow Nets.* ICATPN 97, Volume 1248 of LNCS, pp. 407-426.

W.M.P. van der Aalst. (1998) *The Application of Petri Net to Workflow Management.* The Journal of Circuits, Systems and Computers, Vol.8, No.1, pp. 21-66.

Kamel Barkaoui and Rahma Ben Ayed and Zohra Sba. (2007) *Workflow Soundness Verification based on Structure Theory of Petri Net,* IJCIS Journal, 51-62.

Niels Lohmann, Eric Verbeek, Remco M. Dijkman. (2009) *Petri Net Transformations for Business Processes - A Survey. T. Petri Net and Other Models of Concurrency 2.* 46-63, Springer Berlin Heidelberg.

Dijkman, Remco M., Dumas, Marlon, Ouyang, Chun. (2008) *Semantics and analysis of business process models in BPMN.* Information and Software Technology, 50(12), pp. 1281-1294.

Z.MANNA, A.PNUELI. (1992) *The temporal logic of reactive and concurrent systems,* Springer-Verlag New York, Inc., New York, NY, USA.

Heljanko, K. (1997) *Model checking the branching time temporal logic CTL,* Research Report A45, Helsinki University of Technology, Digital Systems Laboratory, Espoo, Finland.

G.J. Holzmann. (2005) *The Model Checker Spin, Addison-Wesley,* p.596, 2003.