# A Refinement based Verification Approach of BPMN Models using NuSMV

Salma Ayari[1], Yousra Bendaly Hlaoui[2] and Leila Jemni Ben Ayed[3]
[1]*University of Sfax, FSEGS, Tunisia*
[2]*University of Tunis Manar, FST, Tunisia*
[3]*University of Mannouba, ENSI, Tunisia*
*University of Tunis, Latice Laboratory, ENSIT, Tunisia*

Keywords: Refinement, BPMN, Verification, LTL, NuSMV.

Abstract: Modeling complex workflow systems, using BPMN (Business Process Modeling Notation), is going increasing attention by all interested researches in distributed field. The step-wise refinement technique facilitates the understanding of complex systems by dealing with the major issues before getting involved in the details. In this paper, we propose a verification technique based on refinement BPMN process which allows to model an application by refinement and to induce gradually required properties at each level from the abstract to the concrete one. We introduce refinement patterns allowing the design of a complex application at different abstract level. Hence, a formal semantics for BPMN models based on Kripke structure and BPMN refinement patterns will be provided for a formal verification of this correctness. This verification is ensured automatically by NuSMV model Checker based on a BPMN language to NuSMV language transformation. The refinement correctness are expressed as refinement safety properties specified with LTL (Linear Temporal Logic).

## 1 INTRODUCTION

BPMN(Business Process Modeling and Notations)(Allweyer, 2010) language is an ISO standard invented to express business processes. This notation is supported by a number of tools like Bizagi (OMG, 2009), Bonita (Bonitasoft, 2009), jBPM(Hat, 2017), and Intalio (Zamfir, 2011). Such tools offer a support for checking BPMN model syntactical errors. However, semantic errors remain undetected during the design time due to the lack of BPMN semantics which are ambiguous and not concise. Therefore, to detect BPMN model semantic errors, it well be proposed, in this paper, a formal semantics for BPMN models based on Kripke structure (Read, 1999).

In addition, to carry out their missions in a highly competitive context and in order to deal with market changes, companies, increasingly, need to manage complex processes, which is a hard task that cannot be done in one step. These processes must ensure facilitate, evolution and adaption to changes. The evolution which will be discussed here about, is the gradual development of the process, especially from a simple to a more complex one.

It will be proposed, in this paper, a stepwise refinement approach that facilitates the understanding and therefore the good development of such complex systems. The refinement deals with the major issues of the system to develop before getting involved in the details(Younes et al., 2013). This may be implied by the use of different refinement patterns modeling the semantic details to add in the different levels to be more semantically enriched. After studying and analyzing the semantic details that should enrich the more abstract model, four refinement patterns will be developed which specifies all alternative semantics such as *sequence* pattern, *exclusive* pattern, *parallel* pattern and *iterative* pattern. At each refinement step and after inculcating one or more refinement patterns, proofs must be verified if that the performed refinement preserves the previous abstract model requirements in addition of the current more concrete requirements. These requirements are syntactic and semantic design requirements. The syntactic requirements are checked by the design tool, as mentioned above, but the checking of semantic ones represents the object of this work. These requirements are specifyed as behavioral properties of the entire system which should be

Ayari, S., Hlaoui, Y. and Ayed, L.
A Refinement based Verification Approach of BPMN Models using NuSMV.
DOI: 10.5220/0006914105290540
In *Proceedings of the 13th International Conference on Software Technologies (ICSOFT 2018)*, pages 529-540
ISBN: 978-989-758-320-9

satisfied by the more abstract BPMN model and then by all refined models throughout the different refinement levels. It is interested in this paper to specify and to check automatically *system safety properties* using NuSMV (Cimatti et al., 2000). For the verification of semantic requirement preservation, using NuSMV, the requirements are often specified as LTL properties using the Linear Temporal Logic (LTL) (Pnueli, 1977). The NuSMV code is semantically equivalent to a Kripke Structure. This is why this paper is based on a kripke structure to propose a formal semantics for BPMN. The originality of our contribution is about the reliable refinement of BPMN processes that eases the business process designer's task and makes the business process specification more accurate and concise. This contribution could be summarized as follows:

- *Definition of refinement patterns*. In this paper, it is proposed to create a BPMN process using an incremental development based on successive model semantic refinements. Also, it will be defined here a set of refinement patterns formally, as *refinement operators*, which will be used to formalize the BPMN model safety properties.
- *Formalization of BPMN semantics*. A formal definition of BPMN semantics using the formal semantics of NuSMV which is based on Kripke structure will be proposed. This definition allows the formal specification (LTL formulae) of the safety properties that the BPMN models should satisfy to ensure its correctness and reliability.
- *Transformation of BPMN models to NuSMV models*. To check, automatically, the LTL safety properties, we apply a transformation which is based on the formal semantics of BPMN and NuSMV languages where both of them are described using Kripke structure. The common use of the Kripke structure facilitates the transformation and preserves, intrinsically, the semantics of the transformed models.

This paper is organized as follows:
Section 2 discusses related works. Section 3, 4, 5 and 6 defines the BPMN formalization and the notion of transformation in terms of refinement. Section 7 gives preliminaries which are mandatory for our approach. Section 8 and 9 describes the evaluation of our approach over an example. Section 10 discusses the metric of the change. And finally, section 11 concludes the paper.

## 2 RELATED WORKS AND DISCUSSION

Different works has been elaborated in the field of BPMN specification and verification:

- Based on Petri Net (PN) (Petri and Reisig, 2008) for studying the behavior of BPMN models, authors in (Dijkman et al., 2008), (Ramadan et al., 2011) and (Van Der Aalst and Ter Hofstede, 2005) propose a mapping from BPMN to PN, CPN and Yawl to check the semantic correctness of models. According to (Kluza et al., 2011) and (Dijkman et al., 2008) errors revealed in the YAWL model can not be easily tracked in the BPMN model and the verification of YAWL is computationally more complex than the verification on PN. PN is formed with a mathematical formalism that defines its structure and its rules of firing. It can be a powerful tool and easy to understand. However, PN tend to become large even for relatively small systems. The lack of hierarchical composition makes it difficult to specify and understand complex systems using the conventional model(PN) (Cortés et al., 2003). PN models have limitations in their inability to test for exactly a specific marking in an unbounded place and to take action on the outcome of the test (Choi, 1994).
- Based on Process algebras, authors in (Raedts et al., 2007) propose to automatically transform BPMN models to PN. Subsequently, the PN models are transformed into mCRL2 (Groote et al., 2005), a process algebraic language. Also authors in (Capel and Mendoza, 2012) propose that a process algebra can be a formal description language for BPMN processes.
- Based on Automaton : The classical Finite State Machine (FSM) is a basic model of formal specifications and the most well-known model used called a transition system with a finite set of states. Authors in (Morales, 2013) present a set of guideline to transform BPMN models to timed automata (TA) which is made of a finite automaton based structure and a set of clocks and performing model checking with UPPAAL (Behrmann et al., 2004). Authors in (Kherbouche et al., 2012), use a kind of transition system with a few specific characteristics called Kripke Structure. Despite that FSM has lack of expressiveness and the state explosion problem might be a limitation from practical representation of complex control behavior, Kripke structure has a strong point is that it's an input semantic language of several model checkers like Mcheck (Sember, 2005), MlSolver (Oliver and

Martin, 2008), Xspin (Ruys, 1999), UPPAAL, Cadence(Mir et al., 2000), etc and more precisely NuSMV. In addition to that it can be tested with a temporal logic requirements. Most of this model checkers are carried out automatically. All of this points make the model checker well adapted to large scale industrial projects. The adaptability of a business process is an essential requirement for businesses to cope with the dynamic nature of their environments. Flexibility and variability are the words in the 21st century (Bulanov et al., 2011). However, implementing flexibility and variability is not an easy task. The business processes need to change, while they still have to comply with a set of requirements (Sam, 2014). Here a change that is to be discussed named **refinement**. A business will be able to prove that a process will still comply with the requirements after a certain change implied by the **refinement**. Companies will be able to check, in advance, if they can change their business process in such way that they can meet a new demand from the changing market (Sam, 2014). In our Contribution, differently to what was done, several aspects should be addressed and focused like:

1. NuSMV : which is used to analyze BPMN models and can provide a semantics for BPMN in order to verify this model and whether BPMN satisfy properties formalized in LTL. The more automatic approach in formal verification is the model checking and can produce counterexamples that represent subtle errors or interesting execution paths (compare with the theorem proving). The verification is fully exhaustive. Our formal semantic representation of BPMN processes is finite and not too big, so there is nothing to be scared off from the state explosion problem.
2. Programs are complex. In addition to that, research on flexibility and variability of process changes continue to generate growing interest from industry and the community for more than twenty years (Rajabi and Lee, 2010), (Reijers, 2006). We are the only ones to study that behavior with a different succession of layers which must be equivalent. So their contribution which constitutes a correct implementation is weaker than ours.

## 3 A NEW METHODOLOGY FOR THE SPECIFICATION AND THE VERIFICATION OF BUSINESS PROCESSES

The proposed approach for the verification of BPMN models is shown in figure 1. The first part of the application is the configuration loader, where the XML files, for the abstract and the refinement (with a kind of pattern), are parsed into their models. It takes both the abstract view and the refinement view from the configuration loader and convert each one to Kripke Structure. Once this is done the LTL property's file and the Kripke Structure are converted to the input format of the NuSMV model checker. The model checker verifies the requirements and return a counterexample if they are false.

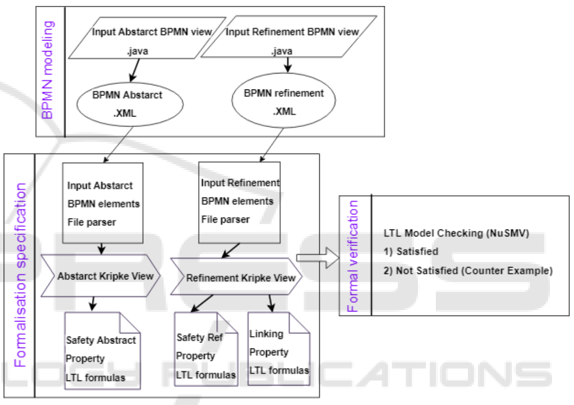

BPMN modeling
Input Abstarct BPMN view .java
Input Refinement BPMN view .java
BPMN Abstarct .XML
BPMN refinement .XML
Formalisation specification
Input Abstarct BPMN elements File parser
Input Refinement BPMN elements File parser
Abstarct Kripke View
Refinement Kripke View
Safety Abstract Property LTL formulas
Safety Ref Property LTL formulas
Linking Property LTL formulas
Formal verification
LTL Model Checking (NuSMV)
1) Satisfied
2) Not Satisfied (Counter Example)

Figure 1: Refinement and verification of BPMN.

## 4 SPECIFICATION BPMN

The Business Process Modeling and Notation is the standard for modeling business process flows proposed by the Object Management Group (OMG).
For the purpose of our refinement process, a BPMN specification can be simplified by discarding all layout information.

**Definition 1.** *(BPMN specification definition:) Let BP =(**O**,**Sf**,**Art**) be a Business process with :*

- ***O*** $= O^{Event} \cup O^{Activity} \cup O^{Gateway}$ *is a set of objects with :*
    1. $O^{Event} = \varepsilon^S \cup \varepsilon^I \cup \varepsilon^E$, *i.e.the set of events partitioned into the disjoint subsets start, intermediate and end events.*
    2. $O^{Activity} = O^{Task} \cup O^{sub-process}$, *i.e. the set of activities partitioned into the disjoint subsets task atomic activity and sub-process.*

3. *$O^{Gateway} = O^P \cup O^{Ex} \cup O^{In}$, i.e. the set of gateways partitioned into the disjoint subsets parallel, exclusive(XOR) and exclusive(Or) gateways.*

- ***Sf** is a set of sequence flow or connecting of flow Objects.*
- ***Art** is the set of artifacts used to provide additional information.*

*The BPMN specification is extended with a function T: $O \rightarrow T_{pre-post}$. With :*
*T is a type of artifact and can be presented as a textual tag that is added on flow objects. $T_{pre-post} \subseteq Art = T_{pre} \cup T_{post}$ is a set of pre and post-conditions for objects. It is necessary to remind that $T_{pre}$ and $T_{post}$ will be replaced by pre and post throughout this paper. This new sets will be used in our refinement patterns.*
***Sf*** $\subseteq$ $(Pre(A_i), A_i, Post(A_i)) \times (Pre(A_{i+1}), A_{i+1}, Post(A_{i+1})), \quad A_i, A_{i+1} \in O \quad 0 \leq i \leq n-1.$

## 5 TRANSFORMATION PROCESS

The main of this paper introduces a method for the preservation of invariant and desirable properties under refinement. The property can be formalized as the behavior of the $BP_{i-1}$ which satisfy the formula $P_{i-1}$. $P_{i-1}$ is called the safety property of the abstract process and, by refinement of BP$_{i-1}$ using refinement patterns a BP$_i$ is given which satisfy another safety property $P_i$. A $BP_{i-1}$ implements another $BP_i$ if it was given a correspondence between them called $P_{ref_i}$. $P_{ref_i}$ is the translation property which is the resulting from the change of variables for refinement. The activity from the abstract level must be refined by using a refinement pattern and this refinement match the abstract one in the sense of preserving a linking property $P_{LinkRef_i}$ who captures how the two models(abstract and refinement models) are related.
This paper investigates a special type of transformation to extend the standard BPMN with a set of refinement patterns shown in figure 2.

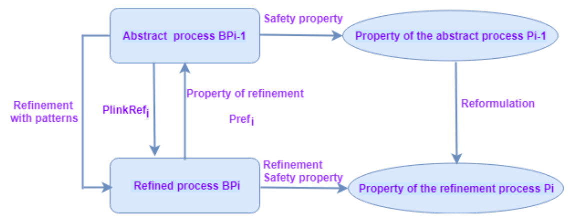


Figure 2: Refinement in BPMN.

## 6 REFINEMENT PATTERNS IN BPMN

The sequential refinement pattern (depicted by >) defines a sequential behavior. The result of applying it consists of a set of activities N (N≥2) that performs one activity first, followed by another activity, in sequence, one after the other. This pattern is presented in figure 3(a).

The parallel refinement pattern (depicted by ||) defines a parallel behavior. The result of applying it consists of a set of activities N (N≥2) that performs the concurrent execution, independently of each other, by the disjoint union of activities after a parallel gateway. This pattern is presented in figure 3(b).

The exclusive choice refinement pattern (depicted by [])defines a conditional behavior. Among the different possible activities N which can be executed(N ≥ 2), One activity is selected based on a specific condition. Once this activity is executed, the other activities cannot be reached anymore. This pattern is presented in figure 3(c).

The iterative loop refinement pattern (depicted by $\curvearrowright$N) defines a single activity which is iterated until N (boolean number) will be equal to zero. This pattern is presented in figure 3(d).

## 7 PRELIMINARY

The following definitions, briefly explain the needed semantic appropriate for the formal description of BPMN processes and the refinement transformation.

According to (Von Stackelberg et al., 2014), a task (or sub-process) is the only flow element which may have both data needs and data results. A precondition summarizes data needs of an activity. Analogously to the precondition, an activity may have post-condition, representing alternative data results.

The safety property of a process BP mentioned before representing the precondition predicate from which the process is guaranteed to terminate and result in a state satisfying the post-condition(figure 4).

**Definition 2.** *(Safety property:) For functions Pre and Post over an object $A \in O^{Activity}$ in BPMN :*

$$P \hat{=} (pre(A) \Rightarrow post(A)).$$

*The symbol $\hat{=}$ is read "is defined to be equal".*

**Definition 3.** *(Property for the refinement:) A $BP_{i-1}$ with a specification property $P_{i-1}$ implements a $BP_i$ with a specification property $P_i$ and by preserving the linking property $P_{LinkRef_i}$, if a correspondence*

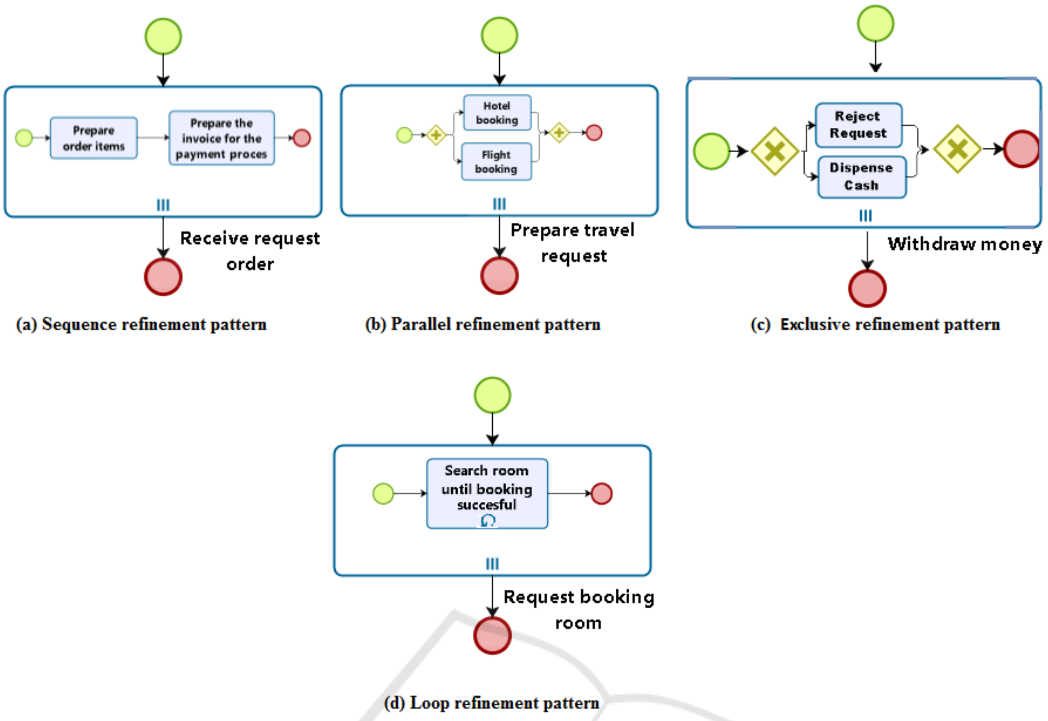

Prepare order items
Prepare the invoice for the payment proces
Receive request order
(a) Sequence refinement pattern
Hotel booking
Flight booking
Prepare travel request
(b) Parallel refinement pattern
Reject Request
Dispense Cash
Withdraw money
(c) Exclusive refinement pattern
Search room until booking succesful
Request booking room
(d) Loop refinement pattern

Figure 3: Refinement pattern examples.

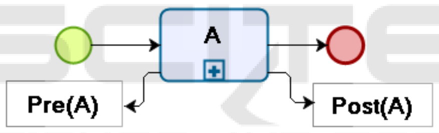


Figure 4: BPMN Object with pre & post-conditions.

*between them is given and called* $P_{ref_i}$.

$$P_{ref_i} \hat{=} P_{LinkRef_i} \wedge P_{i-1} \wedge P_i, \forall i \in [1 \cdots n].$$

**Definition 4.** *(Refinement Mapping:) Each state object belonging to the set $O_{i+1}$ of the refined process $BP_{i+1}$ depends on the state object of the abstract process $BP_i$ belonging to the set $O_i$ in terms of a function mapping $f_i$ at each refinement level where $0 \leq i \leq n-1$ such that :*
*$f_i$ : $BP_i$ -> $BP_{i+1}$*
*A refinement from a specification $BP_i(O_i, Sf_i, Art_i)$ to a specification $BP_{i+1}$ $(O_{i+1}, Sf_{i+1}, Art_{i+1})$ is a mapping.*
*$f_i$ defined by the following inductive rules :*

1. *$f_i(O_i) \subseteq op \times O_{i+1}$ where op is the operator for the refinement patterns which can be the sequence pattern (>), the parallel pattern (||), the inclusive pattern ([ ])or the loop pattern ($\curvearrowright N$).*

   *For the example illustrated in figure 5:*

   *$O_0 = A_0$ , $f_0(A_0) = ( > ,\{A_{01},A_{02},A_{03}\})$.*
   *$O_1 = A_{01}$, $f_1(A_{01}) = ([],\{A_{011},A_{012}\})$.*

2. *$f_i(Sf_i) =$*

$$\begin{cases} pre(A_0), (>, \{A_0, \cdots, A_N\}), post(A_N), \\ (\wedge A_{j \in \{0..N\}}\, pre(A_j)), (\|, \{A_0, \cdots, A_N\}), (\wedge A_{j \in \{0..N\}}\, post(A_j)), \\ (\vee A_{j \in \{0..N\}}\, pre(A_j)), ([], \{A_0, \cdots, A_N\}), (\vee A_{j \in \{0..N\}}\, post(A_j)), \\ ((pre(A_0) \wedge N), (\curvearrowright N, \{A_0\}), (post(A_0) \wedge \neg N), N \in Bool \end{cases}$$

   *for*

$$A_{j_{j \in \{0..N\}}} \in O_{i+1}$$

   *For the example illustrated in figure 5 we have :*
   *$f_0(Sf_0) = pre(A_{01}), (>, \{A_{01}, A_{02}, A_{03}\}), post(A_{03})$;*
   *$f_1(Sf_1) = pre(A_{011}) \vee pre(A_{012}), ([], \{A_{011}, A_{012}\}), post(A_{011}) \vee post(A_{012})$.*

3. *$f_i(P_i) \subseteq (P_{ref_{i+1}})$ with :*

$$f_i(P_i) = P_{LinkRef_{i+1}} \wedge P_i \wedge P_{i+1}$$

   *For the example illustrated in figure 5 we have :*
   *$f_0(P_0) = ((pre(A_0) = Pre(A_{01})) \wedge pre(A_{01}) \rightarrow (post(A_{03}) \wedge (post(A_0) = post(A_{03}))$.*

For the example illustrated in figure 5 we distinguish three refinement patterns: sequence, exclusive and iterative loop patterns.
By referring to the works of (Hlaoui and Ayed, 2010), the formal semantics for an Activity in BPMN is defined as follows : as it is illustrated in figure 6, when

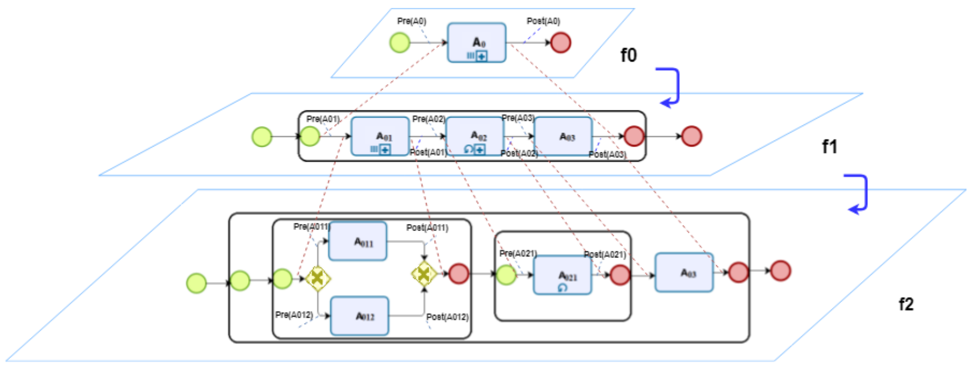


Figure 5: Illustration of the refinement process with refinement patterns for a BP.

the transition of the activity A is fired (**t-A fired**), it assigns the instance of this activity by the *in* event then the activity A is enabled. Once it is enabled(**t-A enabled**), its *out* event occurs and the transition (**t-A end**) is executed. Each state with *in* is labeled with the precondition *pre* and each state with *out* is labeled with the post-condition*post*.

The semantics of BPMN process specifications with

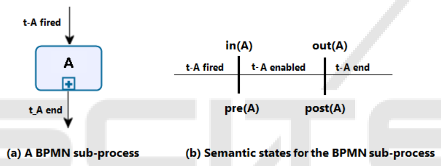


Figure 6: Semantic states for BPMN.

refinement patterns are defined in terms of Kripke structure. It is represented as an automaton and modified with helpful information by adding some atomic properties to its states. In Kripke structure, the nodes represent states of the system and edges represent state transitions.

**Definition 5.** *(Kripke Structure:) A Kripke structure is K = (S , I , T , $\mathscr{L}$) (Kherbouche et al., 2012) where:*

- *S is a finite non-empty set of states,*
- *I ⊆ S is a set of initial states,*
- *T is a transition relation between states such as T ⊆ S × S and*
- *$\mathscr{L}$ : S → $2^{AP}$ assigns truth values to the set of atomic propositions(AP).*

Figure 7 shows an example of a Kripke structure. We give AP = {p,q} a set of atomic propositions which present arbitrary boolean properties. K = ( S , I , T , $\mathscr{L}$ ) where :

- S = $\{s_0, s_1, s_2\}$ ,
- I = $\{s_0\}$ ,
- T = $\{(s_0, s_1), (s_1, s_2), (s_2, s_2)\}$ and
- $\mathscr{L}$ such that :
  - $\mathscr{L}(s_0) = \{p\}$,
  - $\mathscr{L}(s_1) = \{p, q\}$ and
  - $\mathscr{L}(s_2) = \{q\}$.

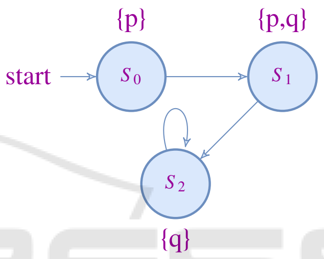


Figure 7: Kripke Structure.

As it was explained before, this structure will be used in order to illustrate the utilization of the defined semantics for a BP.

The semantic description is the behavior of the business process depicted by the instantiate function of flow object (Fo) and more precisely of the activity (task or sub-process) which shows the steps of work performed in a process. The behavior is presented by a sequence of permitted states $\{in, out\}$ ∈ S.

**Definition 6.** *(A proposition of a formal semantic using the Kripke structure for BPMN processes:) A process BP =(Fo,Sf,Art) induces a Kripke structure K = (S,I,T,$\mathscr{L}$) with :*

1. *S being the set of all valid system states which present the behavior of objects in BP called Ins(O) where O ∈ $Fo^{Activity}$ such that :*

$$Ins : Fo^{Activity} \rightarrow In\_Fo \times Out\_Fo$$

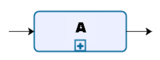
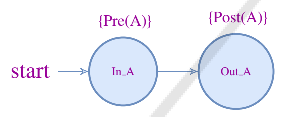
   *where :*
   *$In\_Fo = \{in\_O | O \in Fo^{Activity}\}$ and $Out\_Fo = \{out\_O | O \in Fo^{Activity}\}$*
2. *I being the set of initial states;*
3. *T being the transition relation between the instantiate object Flow ; Ins(O) × Ins (O);*

4. *$\mathscr{L}: S \rightarrow 2^P$. P is a set of elementary properties verified by each state of the entire system.*

Our description of BPs is based on a description of the process and a description of the property. We will focus also on the BPMN model's refinement properties. We give their definitions and then explain how required properties are depicted from the system behavior constructs of BPMN models with refinement and how they can be formally stated.

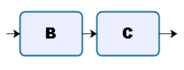
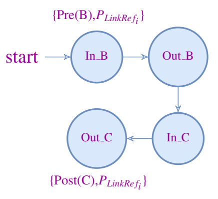
I ) The formal semantic using Kripke structure for a BPMN abstract Activity A $\in$ Fo is given as follows:
$\mathscr{L}(in\_A) = \{pre(A) | \nexists s \in S \wedge s \neq in\_B, (s, in\_A) \in T\}$
$\mathscr{L}(out\_A) = \{post(A) | \forall s \in S, (out\_A, s) \notin T\}$
Table 1 gives a semantic description of a BPMN refined sub-process A to the Kripke structure.

Table 1: Adopted Semantics for a refined sub-process A.

| BPMN Object | Kripke Structure |
|---|---|
|  |  |
| $P_{i-1} \hat{=} (Pre(A) \rightarrow Post(A))$<br>$P_{LinkRef_i} \hat{=} \emptyset$ | |

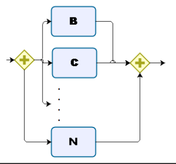
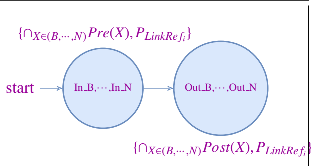
II ) The formal semantic using Kripke structure for a sequence refinement pattern $(>, \{B, \cdots, N\})$ where A is the refined activity:
$\mathscr{L}_{>}(in\_B) = \{pre(B), P_{linkRef_i} | \nexists s \in S \wedge s \neq in\_B; (s, in\_B) \in T \wedge P_{LinkRef_i} \hat{=} pre(B) = pre(A)\}$
$\mathscr{L}_{>}(out\_N) = \{post(N), P_{LinkRef_i} | \forall s \in S; (out\_N, s) \notin T \wedge P_{LinkRef_i} \hat{=} post(N) = post(A)\}$
Table 2 gives a semantic description of a BPMN sequence refinement pattern $(>, \{B, C\})$ to the Kripke structure.

Table 2: Adopted Semantics for a sequence refinement pattern.

| BPMN Object | Kripke Structure |
|---|---|
|  |  $\{Pre(B), P_{LinkRef_i}\}$<br>start<br>In_B<br>Out_B<br>In_C<br>Out_C<br>$\{Post(C), P_{LinkRef_i}\}$ |
| $P_{Ref_i} \hat{=} (Pre(A) = Pre(B) \wedge Pre(B)$<br>$\rightarrow (Post(C) \wedge Post(C) = Post(A))$ | |

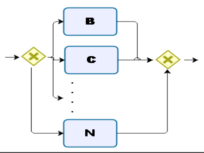
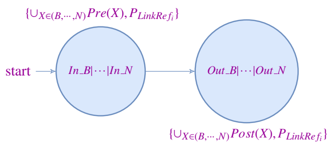
III ) The formal semantic using Kripke structure for a parallel refinement pattern $(\|, \{B, \cdots, N\})$ where A is the refined activity:
$\mathscr{L}_{\|}(\cap_{X \in (B, \cdots, N)} in\_X) = \{\cap_{X \in (B, \cdots, N)} Pre(X), P_{linkRef_i} | \nexists s \in S \wedge s \neq in\_B; (s, in\_B) \in T \wedge P_{LinkRef_i} \hat{=} \cap_{X \in (B, \cdots, N)} Pre(X) = pre(A)\}$
$\mathscr{L}_{\|}(\cap_{X \in (B, \cdots, N)} out\_X) = \{\cap_{X \in (B, \cdots, N)} Post(X), P_{LinkRef_i} | \forall s \in S; (out\_N, s) \notin T \wedge P_{LinkRef_i} \hat{=} \cap_{X \in (B, \cdots, N)} Post(X) = post(A)\}$ Table 3 gives a semantic description of a BPMN parallel refinement pattern $(\|, \{B, \cdots, N\})$ to the Kripke structure.

Table 3: Adopted Semantics for a parallel refinement pattern.

| BPMN Object | Kripke Structure |
|---|---|
|  |  $\{\cap_{X \in (B, \cdots, N)} Pre(X), P_{LinkRef_i}\}$<br>start<br>In_B,⋯,In_N<br>Out_B,⋯,Out_N<br>$\{\cap_{X \in (B, \cdots, N)} Post(X), P_{LinkRef_i}\}$ |
| $P_{Ref_i} \hat{=} Pre(A) = \cap_{X \in (B, \cdots, N)} Pre(X) \wedge \cap_{X \in (B, \cdots, N)} Pre(X) \rightarrow (\cap_{X \in (B, \cdots, N)} Post(X) \wedge$<br>$\cap_{X \in (B, \cdots, N)} Pre(X) \rightarrow (\cap_{X \in (B, \cdots, N)} Post(X) \wedge$<br>$\cap_{X \in (B, \cdots, N)} Post(X) = Post(A))$ | |

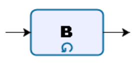
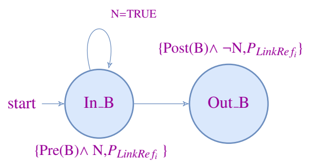
IV ) The formal semantic using Kripke structure for an exclusive refinement pattern $([], \{B, \cdots, N\})$ where A is the refined activity :
$\mathscr{L}_{[]}(\cup_{X \in B \cdots X} in\_X) = \{\cup_{X \in B \cdots N} Pre(X), P_{LinkRef_i} | \nexists s \in S \wedge s \neq in\_B; (s, in\_B) \in T \wedge P_{LinkRef_i} \hat{=} \cup_{X \in (B, \cdots, N)} Pre(X) = pre(A)\}$
$\mathscr{L}_{[]}(\cup_{X \in B \cdots X} out\_X) = \{\cup_{X \in B \cdots N} Post(X), P_{LinkRef_i} | \forall s \in S; (out\_N, s) \notin T \wedge P_{LinkRef_i} \hat{=} \cup_{X \in (B, \cdots, N)} Post(X) = post(A)\}$ Table 4 gives a semantic description of a BPMN exclusive refinement pattern $([], \{B, \cdots, N\})$ to the Kripke structure.

Table 4: Adopted Semantics for an exclusive refinement pattern.

| BPMN Object | Kripke Structure |
|---|---|
|  |  $\{\cup_{X \in (B, \cdots, N)} Pre(X), P_{LinkRef_i}\}$<br>start<br>In_B\|⋯\|In_N<br>Out_B\|⋯\|Out_N<br>$\{\cup_{X \in (B, \cdots, N)} Post(X), P_{LinkRef_i}\}$ |
| $P_{Ref_i} \hat{=} Pre(A) = \cup_{X \in (B, \cdots, N)} Pre(X) \wedge \cup_{X \in (B, \cdots, N)} Pre(X) \rightarrow (\cup_{X \in (B, \cdots, N)} Post(X) \wedge$<br>$\cup_{X \in (B, \cdots, N)} Pre(X) \rightarrow (\cup_{X \in (B, \cdots, N)} Post(X) \wedge$<br>$\cup_{X \in (B, \cdots, N)} Post(X) = Post(A))$ | |

V ) The formal semantic using Kripke structure for a loop refinement pattern $(\curvearrowright N, B)$ where A is the refined activity:
$\mathscr{L}_{\curvearrowright}(in\_B) = \{Pre(B) \wedge N, P_{LinkRef_i} | \nexists s \in S \wedge s \neq in\_B; (s, in\_B) \in T \wedge P_{LinkRef_i} \hat{=} Pre(B) \wedge N = Pre(A)\}$
$\mathscr{L}_{\curvearrowright}(out\_B) = \{Post(B) \wedge \neg N, P_{LinkRef_i} | \forall s \in S; (out\_N, s) \notin T \wedge P_{LinkRef_i} \hat{=} Post(B) \wedge \neg N = Post(A)\}$
Table 5 gives a semantic description of a BPMN loop refinement pattern $(\curvearrowright N, B)$ to the Kripke structure.

Table 5: Adopted Semantics for a loop refinement pattern.

| BPMN Object | Kripke Structure |
|---|---|
|  |  N=TRUE<br>{Post(B)∧ ¬N,$P_{LinkRef_i}$}<br>start<br>In_B<br>Out_B<br>{Pre(B)∧ N,$P_{LinkRef_i}$ } |
| $P_{Ref_i} \hat{=} (Pre(B) \wedge N) = Pre(A) \wedge (Pre(B) \wedge N) \rightarrow (Post(B) \wedge \neg N$ $(Pre(B) \wedge N) \rightarrow (Post(B) \wedge \neg N$ $\wedge (Post(B) \wedge \neg N) = Post(A))$ | |

The soundness of BPMN process model to verify the safety and the refinement can be ensured by satisfying the properties described above.

## 8 CASE STUDY

In this section we will describe our system and then try to apply the refinement verification for this system.

### 8.1 Description

A Compose Email sub-process presented in figure 8 is a message which can be text or voice. The text message can be sent by adding the address of the receiver, the message title which is called the subject, typing the email message and sending this text.

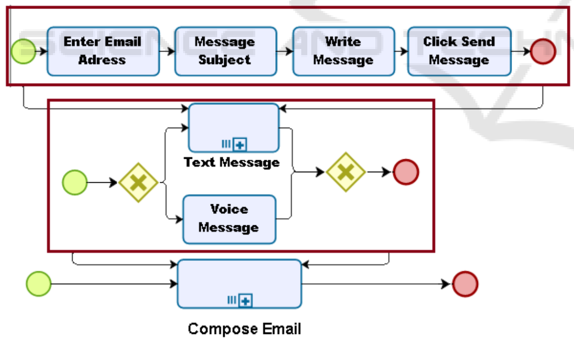


Figure 8: Compose Email Sub-process.

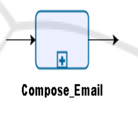
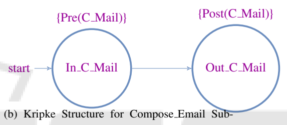
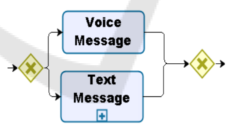
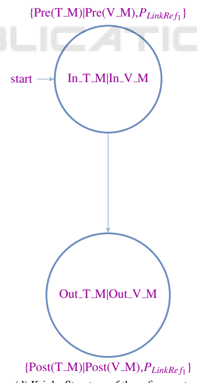
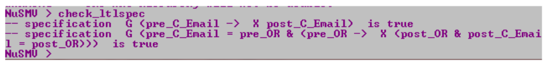
### 8.2 Experimental Result

We discuss thereafter the first refinement of the sub-process Compose Email. Table 6 shows the state diagram of the sub-process compose Email. The translation from the sub-process shown in table 6(a) to a Kripke structure shown in table 6(b) is the instantiate of a sub-process. The refined sub-process compose Email defined by the refinement sub-process Text message or the task Voice message is a choice which can be made between the two activities is described in table 6(c). The translation from this sub-process to a kripke structure is shown in table 6(d). A generated specification expressed in LTL language checked for safety abstract property and refinement property was the following:

- $P_0 \hat{=} G(pre(C\_Mail) \rightarrow X post(C\_Mail)$.
- $P_{Ref_1} \hat{=} G((pre(C_M ail) \;=\; pre(OR))\&pre(OR) \;\rightarrow\; X(post(OR)\&post(C\_Mail) = post(OR)))$.

With : pre(OR) = pre(T_M)— pre(V_M)and post(OR) = post(T_M)— post(V_M).
We remind that:
pre(C_Mail) (post(C_Mail)) is the precondition (post-condition) of the activity Compose Email, pre(T_M) (post(T_M)) is the precondition (post-condition) of the activity text message and pre(V_M) (post(V_M)) is the precondition (post-condition) of the activity Voice Message.
This requirement is obviously true for the discussed example cited above according to table 7(e).

Table 6: Modeling and verification of Compose_Email sub-process.

| | |
|---|---|
|  (a) Compose_Email Sub-process |  {Pre(C_Mail)}<br>{Post(C_Mail)}<br>start<br>In_C_Mail<br>Out_C_Mail (b) Kripke Structure for Compose_Email Sub-process |
|  (c) Refinement of sub-process Compose_Email |  {Pre(T_M)\|Pre(V_M),$P_{LinkRef_1}$}<br>start<br>In_T_M\|In_V_M<br>Out_T_M\|Out_V_M<br>{Post(T_M)\|Post(V_M),$P_{LinkRef_1}$} (d) Kripke Structure of the refinement |
|  (e) Result of the refinement and the abstract requirements checking for Compose_Email | |

```
NuSMV > check_ltlspec
-- specification  G (pre_C_Email ->  X post_C_Email)  is true
-- specification  G (pre_C_Email = pre_OR & (pre_OR ->  X (post_OR & post_C_Emai
l = post_OR)))  is true
NuSMV >
```

## 9 IMPLEMENTATION

The figure 9 and 10 shows the screen-shots of the the abstract process Compose_Email and the refinement in the main edit view of the application. Users can drag elements from the element palette in the top of the Graphical User Interface (GUI). The GUI of the abstract model is developed by our team (Hlaoui and Ayed, 2009). After saving or importing a configuration of both the abstract and the refined matching model defining the behavior of one of the refinement patterns in an XML file, the tool converts them to a Kripke structure each of them. Once this is done it applies the safety and the refinement properties in order to check the flexibility of the process. The two Kripke structures are convert to the input format of NuSMV model checker and the properties are demonstrated.

## 10 IMPACT ANALYSIS OF CHANGE (REFINEMENT)

The flexibility of a process is defined as a property who allow to this process the adaptation of change without causing the instability of the system where it is used. Indeed, in today's dynamic business world, the economic success of an enterprise increasingly depends on its ability to react to changes within its environmental in a quick and flexible way (Fdhila et al., 2011). This paper presents a new methodology for adaptation to a special change called refinement. According to (Dahman et al., 2013), the changes can be described by basic operations that express atomic modifications which are noted by $U_\delta$. This changes can be made on process according to this three operations :

- Create(fragment): Insert a new fragment into the process.
- Destroy(fragment): Delete a fragment from the process.
- Update(fragment,$\mu$,$\vartheta$): Associate the property $\mu$ of the fragment with the value $\vartheta$.
- Undo(fragment,$\mu$,$\vartheta$): Dissociate the property $\mu$ of the fragment with the value $\vartheta$.

$\mu$ can be a labeling property, a type of object flow, a target or a source.
The model obtained by applying of a sequence of operations $U_{\delta_f}$ in the fragment f is apply(f,$U_{\delta_f}$).

## Metrics

**Definition 7.** *(Normalized size:) (Dahman, 2012)*
*The size of a sequence of operations size(Q)=||Q|| is defined by the number of operations of creation and destruction of object flows. The normalization simplifies all reverse or dependent operations.*

Figure 11 shows the change of the sub-process *Compose_Email* by the exclusive refinement($[],\{Text\_M, Voice\_M\}$) described in the previous section. The result of this refinement is:
$Comp\_Email' = apply(Copmose\_Email, U_{\delta_{Compose\_Email}})$.

1. Size (*compose_E*) = 1.
   ⇒ create(*Comp_Email*).
2. Size ($[],\{Text\_M, Voice\_M\}$) = Size ($U_{\delta_{comp\_E}}$) = 4.
   ⇒ create(Comp_Email) , destroy(Comp_Email), create(Text_M), create(Voice_M),create(G1), create(G2). (G1 and G2 are the gateways).
3. Size (comp_Email)' = 4

**Definition 8.** *(Normalized semantics:)(Dahman, 2012)*
*The semantics of a sequence of operations Q = ||Q|| is defined by the update and the undo operations. The normalization simplifies all reverse or dependent operations.*

1. Sem(compose_E) = 2.
   ⇒ update(compse_E,lab,in(compose_E)), update(compose_Email,lab,out(compose_E)).
2. Sem($[],\{Text\_M, Voice\_M\}$)= 10.
   ⇒ update(Text_M,lab,in(Text_M)),update(Text_M, lab,out(Text_M)), update(G1,typ,objectFlow), update(G1,tar,Text_M),update(G1,tar,Voice_M), update(G2,typ,objectFlow),update(Text_M, tar,G2),update(Voice_M,tar,G2),update(Voice_M ,lab,in(Voice_M)),update(Voice_M ,lab,out(Voice_M)).
3. Sem(Comp_E') = 10.
   ⇒ update(compse_E ,lab,in(compose_E)), update(compose_Email, lab,out(compose_E)), undo(compse_E ,lab,in(compose_E)), undo(compose_Email ,lab,out(compose_E)) update(Text_M ,lab,in(Text_M)), update(Text_M ,lab,out(Text_M)),update(G1,typ,objectFlow), update(G1,tar,Text_M),update(G1,tar,Voice_M), update(G2,typ,objectFlow),update(Text_M ,tar,G2), update(Voice_M ,tar,G2), update(Voice_M ,lab,in(Voice_M)), update(Voice_M ,lab,out(Voice_M)).

**Definition 9.** *(Normalized complexity:(Dahman, 2012)) The normalized complexity is a complexity of a sequence of operations defined by:*

$$Comp(Q) = Sem(Q)/Size(Q)$$

For the example illustrated before we have:

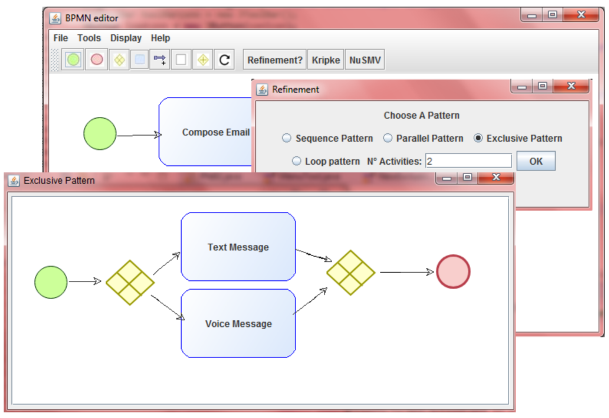
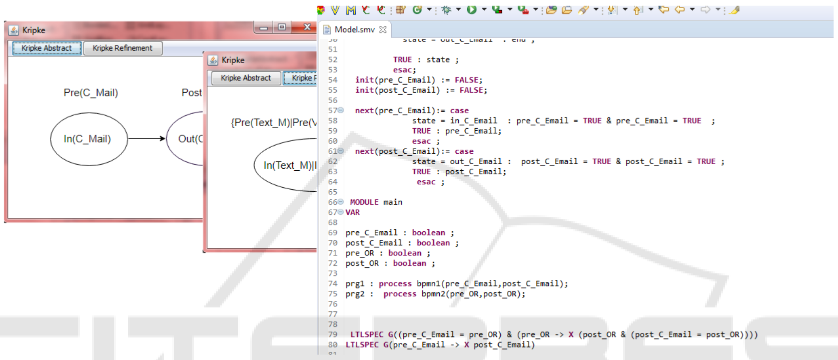


Figure 9: Screen Shots for the abstract and the refinement processes.



```
        TRUE : state ;
        esac;
  init(pre_C_Email) := FALSE;
  init(post_C_Email) := FALSE;

  next(pre_C_Email):= case
            state = in_C_Email  : pre_C_Email = TRUE & pre_C_Email = TRUE  ;
            TRUE : pre_C_Email;
            esac ;
  next(post_C_Email):= case
            state = out_C_Email :  post_C_Email = TRUE & post_C_Email = TRUE ;
            TRUE : post_C_Email;
             esac ;

 MODULE main
VAR

pre_C_Email : boolean ;
post_C_Email : boolean ;
pre_OR : boolean ;
post_OR : boolean ;

prg1 : process bpmn1(pre_C_Email,post_C_Email);
prg2 :  process bpmn2(pre_OR,post_OR);



 LTLSPEC G((pre_C_Email = pre_OR) & (pre_OR -> X (post_OR & (post_C_Email = post_OR))))
LTLSPEC G(pre_C_Email -> X post_C_Email)
```

Figure 10: Screen Shots for the the mapping to Kripke structure and then to NuSMV.

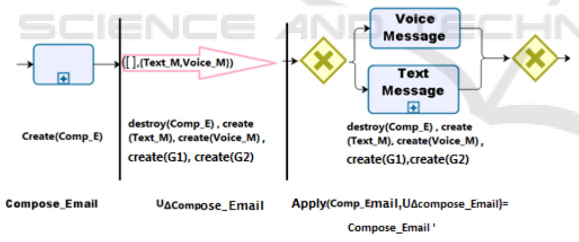


Figure 11: Applying an operation with a first refinement change of *Compose_Email*.

Table 7: Results of complexity for the example.

| | |
|---|---|
| Comp(Compose_Email | 2/1 = 1 |
| Comp(([], {$Text\_M, Voice\_M$}) | 10/4 = 2.5 |
| Comp(comp_Email' | 10/4 = 2.5 |

Based on this results, authors in (Dahman, 2012) define a *Metric of a correct automated transformation*. They allow to compare transformations and synchronization.

$$size(M') = \lambda * size(M) \&$$

$$sem(M') = \phi * sem(M) \&$$

$$comp(M') = \phi / \lambda \; comp(M).$$

where M' is the transformation of M, $\lambda$ and $\phi$ are factors who depend on the mapping of the operations that are present in the source models.

1. $\lambda = size(Comp\_Email')/size(Comp\_Email) = 4/1 = 4$.
2. $\phi = sem(comp_{E}mail')/sem(comp_{E}mail) = 10/2 = 5$.
3. $\phi/\lambda = 5/4 = 1.25 \approx 1 \Rightarrow Comp\_Email' \approx Comp\_Email$. Complexity is preserved during this transformation.

We can also conclude that even with incremental semantic enrichment (here refinement) the complexity is not high which favors our methodology.

## 11 CONCLUSIONS

To reduce the complexity of business process modeling, we introduced in this paper refinement technique in modeling with BPMN. This allows developers to introduce gradually system requirements and to prove at each level that the model preserves the refined one. Refinement patterns have been introduced allowing rigorous refinement. Also, with proofs related to these patterns, developer can introduce grading requirements in the development process and can verify that the refinement preserves requirements of the refined model. Throughout this paper, we proposed new methodology for the specification and the verification of business processes based on BPMN and refinement, and using NuSMV model checker for the verification. This allows the developer to guarantee that the properties of a business process are conserved by the different refinement patterns. We won't make an automatic refinement because until now it's an interactive step of our approach. Also, it's important to propose a solution allowing developer to discover the origin of an eventual error on the model in case of non verified LTL formulas in the checking step. In the future we will implement a quality management plug-in to manage the quality of a business process after each change.

## REFERENCES

Allweyer, T. (2010). *BPMN 2.0: Introduction to the Standard for Business Process Modeling*. Books on Demand. https://books.google.tn/books?id=fdlC7K_3dzEC.

Behrmann, G., David, A., and Larsen, K. G. (2004). A tutorial on uppaal. In *Formal methods for the design of real-time systems*, pages 200–236. Springer.

Bonitasoft (2009). Bonita open solution.

Bulanov, P., Groefsema, H., and Aiello, M. (2011). Business process variability: A tool for declarative template design. In *International Conference on Service-Oriented Computing*, pages 241–242. Springer.

Capel, M. I. and Mendoza, L. E. (2012). Automating the transformation from bpmn models to csp+ t specifications. In *Software Engineering Workshop (SEW), 2012 35th Annual IEEE*, pages 100–109. IEEE.

Choi, B. W. (1994). Petri net approaches for modeling, controlling, and validating flexible manufacturing systems.

Cimatti, A., Clarke, E., Giunchiglia, F., and Roveri, M. (2000). Nusmv: a new symbolic model checker. *International Journal on Software Tools for Technology Transfer*, 2(4):410–425.

Cortés, L. A., Eles, P., and Peng, Z. (2003). Modeling and formal verification of embedded systems based on a petri net representation. *Journal of Systems Architecture*, 49(12-15):571–598.

Dahman, K. (2012). *Gouvernance et étude de l'impact du changement des processus métiers sur les architectures orientées services : une approche dirigée par les modèles. (Governance and Analysis of Business Processes Change Impact on Service Oriented Architectures: A Model-Driven Approach)*. PhD thesis, University of Lorraine, Nancy, France.

Dahman, K., Charoy, F., and Godart, C. (2013). Alignment and change propagation between business processes and service-oriented architectures. In *Services Computing (SCC), 2013 IEEE International Conference on*, pages 168–175. IEEE.

Dijkman, R. M., Dumas, M., and Ouyang, C. (2008). Semantics and analysis of business process models in bpmn. *Information and Software technology*, 50(12):1281–1294.

Fdhila, W., Baouab, A., Dahman, K., Godart, C., Perrin, O., and Charoy, F. (2011). Change propagation in decentralized composite web services. In *Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), 2011 7th International Conference on*, pages 508–511. IEEE.

Groote, J. F., Mathijssen, A., van Weerdenburg, M., and Usenko, Y. (2005). From $\mu$crl to mcrl2. *Algebraic Process Calculi: The First Twenty Five Years and Beyond*, page 126.

Hat, R. (2017). Java business process model. Technical report.

Hlaoui, Y. B. and Ayed, L. J. B. (2009). Patterns for modeling and composing workflows from grid services. In *Enterprise Information Systems, 11th International Conference, ICEIS 2009, Milan, Italy, May 6-10, 2009. Proceedings*, pages 615–626.

Hlaoui, Y. B. and Ayed, L. J. B. (2010). Symbolic model checking supporting formal verification of grid service workflow models specified by UML activity diagrams. In *NOTERE 2010, Annual International Conference on New Technologies of Distributed Systems, Touzeur, Tunisia, May 31 - June 2, 2010, Proceedings*, pages 255–260.

Kherbouche, O. M., Ahmad, A., and Basson, H. (2012). Detecting structural errors in bpmn process models. In *Multitopic Conference (INMIC), 2012 15th International*, pages 425–431. IEEE.

Kluza, K., Nalepa, G. J., Szpyrka, M., and Ligeza, A. (2011). Proposal of a hierarchical approach to formal verification of bpmn models using alvis and xtt2 methods. In *7th Workshop on Knowledge Engineering and Software Engineering (KESE 2011) at the Conference of the Spanish Association for Artificial Intelligence (CAEPIA 2011), La Laguna (Tenerife), Spain, November*, volume 10, pages 15–23.

Mir, A. A., Balakrishnan, S., and Tahar, S. (2000). Modeling and verification of embedded systems using cadence smv. In *Electrical and Computer Engineering, 2000 Canadian Conference on*, volume 1, pages 179–183. IEEE.

Morales, L. E. M. (2013). Business process verification using a formal compositional approach and timed automata. In *Computing Conference (CLEI), 2013 XXXIX Latin American*, pages 1–10. IEEE.

Oliver, F. and Martin, L. (2008). Mlsolver.

OMG (2009). Bizagi modeler.

Petri, C. A. and Reisig, W. (2008). Petri net. *Scholarpedia*, 3(4):6477.

Pnueli, A. (1977). The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 46–57. IEEE.

Raedts, I., Petkovic, M., Usenko, Y. S., van der Werf, J. M. E., Groote, J. F., and Somers, L. J. (2007). Transformation of bpmn models for behaviour analysis. *MSVVEIS*, 2007:126–137.

Rajabi, B. A. and Lee, S. P. (2010). Modeling and analysis of change management in dynamic business process. *International Journal of Computer and Electrical Engineering*, 2(1):181.

Ramadan, M., Elmongui, H. G., and Hassan, R. (2011). Bpmn formalisation using coloured petri nets. In *Proceedings of the 2nd GSTF Annual International Conference on Software Engineering & Applications (SEA 2011)*.

Read, S. (1999). A new introduction to modal logic.

Reijers, H. A. (2006). Implementing bpm systems: the role of process orientation. *Business Process Management Journal*, 12(4):389–409.

Ruys, T. C. (1999). Xspin/project-integrated validation management for xspin. In *International SPIN Workshop on Model Checking of Software*, pages 108–119. Springer.

Sam, v. D. (2014). Model checking business processes the search for the most compatible model checker.

Sember, J. (2005). Mcheck: A model checker for ltl and ctl formulas.

Van Der Aalst, W. M. and Ter Hofstede, A. H. (2005). Yawl: yet another workflow language. *Information systems*, 30(4):245–275.

Von Stackelberg, S., Putze, S., Mülle, J., and Böhm, K. (2014). Detecting data-flow errors in bpmn 2.0. *Open Journal of Information Systems (OJIS)*, 1(2):1–19.

Younes, A. B., Hlaoui, Y. B., Ayed, L. J. B., and Jlassi, R. (2013). Refinement based modeling of workflow applications using UML activity diagrams. In *IEEE 37th Annual Computer Software and Applications Conference, COMPSAC Workshops 2013, Kyoto, Japan, July 22-26, 2013*, pages 187–192.

Zamfir, M. (2011). Evaluation of Intalio BPM Tool. *Revista Romänä de Informaticä şi Automaticä*, 21(1):59–70.