Detecting structural errors in BPMN process models

Mohammed Oussama Kherbouche, Adeel Ahmad, Henri Basson

Detecting structural errors in BPMN process models

Oussama Mohammed Kherbouche, Adeel Ahmad, Henri Basson
Université Lille Nord de France
Laboratoire d'Informatique, Signal et Image de la Côte d'Opale
BP-719 62228 CALAIS Cedex FRANCE
Email: {kherbouche, ahmad, basson}@lisic.univ-littoral.fr

Abstract— **Business Process Modeling Notation (BPMN) has emerged as a standard notation to express the business process models. A lack of formal semantics in the BPMN can cause the syntactic and structural errors. The former requires less effort to be checked, while the later usually needs a complex state-space analysis to prove some properties, like the deadlock-freedom and the livelock-freedom. In this paper, we present an approach based on model checking for the automated verification of business process models. We illustrate the deadlocks, livelocks, and multiple termination problems, which can help the business modelers to avoid structural errors.**

Keywords-component; BPMN process models; Kripke structure LTL; Model checking.

I. INTRODUCTION

The Business Process Management (BPM) [1, 2] has been increasingly used for the identification, specification and modeling of business process. During the last decade, Business Process Modeling Notations (BPMN) [3] has become a standard, in this regard, for the modeling of business processes. It has been also used as a tool for expert analysis for decision making. This success is based on its simplicity of notations [4] and its exhaustive expressiveness. Nevertheless the widely used BPMN as modeling support for business process relies on the human expertise along with associated possible mistakes.

The major distinct possible errors can be either syntactical or structural. The syntactical errors may occur by mistaking the use of modeling elements i.e. an *AND-join*, *OR/XOR-join* or an event when it does not allow more than one outgoing arc, etc. The valid or invalid combinations to be used are usually prescribed by the corresponding standard. The syntactical correctness of models can be verified by using some modeling tools such as BizAgi [5], Intalio [6], or Bonita [7].

However, a syntactically correct process can exhibit unexpected behavior during its run-time, as a result of poorly controlled data or structural errors. The structural errors, such as wrong combination of the sequence of elements given by misaligned splits and joins are difficult to be detected due to lack of formal semantics of BPMN process models. Subsequently, the run-time behavior of a process should be analyzed to achieve a complete verification, showing whether the process model fulfills important structural criteria. These can be either deadlock-freedom or livelock-freedom to avoid the proper functioning of the process, which can cost financially expensive damages.

For a business process consistency, business process modelers should check the accuracy and compliance of adopted models after each applied change on an existing model. The objective is to reach a verified changed model, which must be also able to provide satisfying responses to the questions related to the model consistency, such as: 'Does the processes terminate?', 'Is there a possible deadlock?', 'Does every task *T* of a certain process *P* is reachable?', etc. The difficulty of responses depends on the scale of size and complexity of the evolving process which may cost more time and can involve a higher expertise level.

In this paper, we propose an approach to automate the checking of some structural errors such as deadlocks, livelocks, and multiple terminations in BPMN process models based on model checking. The approach has two major advantages. Firstly, we assume a computable polynomial time, i.e. most of the structural errors are actually detectable. Secondly, if an error is found, it provides a direct graphical path leading to the error. The main idea is to map the BPMN process model to Kripke structure, and then check the validity of major properties (e.g. absence of deadlocks, livelocks and multiple terminations) expressed in Linear Temporal Logic (LTL) [8, 9] formulae.

The rest of the paper is structured as follows. The section II briefly narrates the closely related work to the proposed approach. Section III, summarizes the preliminaries used to illustrate our approach. We describe the frequent structural errors in the section IV. Section V discusses, in detail, the proposed approach, along with pertinent examples. Later in Section VI, we conclude our contribution.

II. RELATED WORK

The structural errors can interrupt the execution of business process models. We intend by the term structural errors as the deadlocks, livelocks, or multiple terminations. The motivation, of the current work, has been to provide a means for automated verification of absence of structural errors. The proposed approach allows the better identification of the non-compliant business processes, before their execution. During the last decade, there are many research works focused on detecting the structural errors in the business processes. In [10], an approach has been proposed for detecting deadlocks and multiple termination patterns in SAP reference model. It is intended to be applied on two popular modeling languages i.e. Event-Process Chain (EPC) and PetriNet. Dijkman *et al.* [11] also propose PetriNet-based method to verify BPMN process

models. However, it advocates the human judgment to detect and assess the correctness of structural errors. Their semantics does not strictly conform to the multiple instances of model, exception handling, and message flows. Furthermore, Awad *et al*. [12, 13] present an approach to detect deadlocks using a method in continuation of their previous work concerning BPMN-Q [14]. They use *business process querying* to detect the common structural errors. They propose a customized language to graphically interpret the *deadlock patterns* whose occurrence in process models lead to deadlocks. Such ad-hoc queries based on quantifications of based on structural properties may enhance the space and time complexities. In [15] Van der Aalst proposes soundness criterion to guide the modeling regarding the specification of EPC. They propose to map EPCs (without connectors of type V) onto Petri nets. As a result it gains the advantages of formal semantics and analyzing techniques available for Petri nets. However, their semantics does not properly model multiple instances, exception handling, and message flow using Petri nets. Another approach [16] proposes finite-state automata to detect deadlocks and multiple terminations. They transform BPMN model to an automata-based formalism to verify the compatibility of transition function. The correctness of the BPMN model is assessed through the existence of process sequence, which could be accepted by the process automata. Although, their approach suites the *deadlock* and *multiple termination* problems, but it has attached inconveniences of complexities of automata-based formalism. Also, it is difficult to analyze the *live-lock* situations.

Model checking can be used for the better detection of some structural errors. Our approach is aimed to provide an automated assistance to verify the soundness process model. It can be implemented by abundant model checker tools, e.g SPIN [17], NuSMV [18], etc.

III. PRELIMINARIES

The following sections, we briefly explain the concepts and technical terms used in the proposed approach.

A. Business Process Modeling Notation

The Business Process Modeling Notation (BPMN), adopted by OMG, has been specified since February 2006 (3). It is used as a standard notation set. The primary goal of BPMN is to provide the notations which are readily understandable by all business users. BPMN creates a standardized bridge for the gap between the business process design and process implementation.

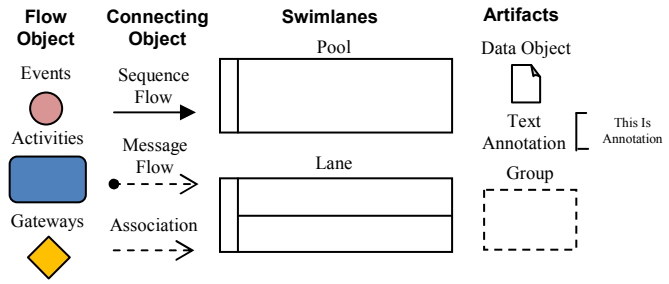

Figure 1. A core of BPMN elements

As shown in Fig.1, BPMN process diagrams provide a number of graphical notations for business process. These can be categorized as below:

- Flow Objects: are the main graphical elements to define the behavior of a business process. There are three kinds of flow objects, which are event, activity, and gateway.
- Connectors: are the graphical elements to connect the Flow Objects to each other. There are three kinds of Connecting Objects, which are Sequence Flow, Message Flow, and Association.
- Swimlanes: are the graphical elements to group the modeling elements. There are two ways of grouping the primary modeling elements, which are pools and lanes.
- Artifacts: are used to provide additional information about the Process. There are two standardized Artifacts, which are Group and Text Annotation.

B. Model Checking

The model-checking [19] method is based on three phases:

- The system modeling phase: The objective of this phase is to provide the formal semantics representation of the system. It represents transition systems where the nodes are system states and transitions describe the possible reachability of one state to another. It includes the Kripke structures, Petri nets, finite automata, timed automata, etc.
- The specification phase of the property: It is a translation phase to specify the property to check, in a formal language, which was formerly written in natural language. Among the many formalisms proposed, there is a variation of temporal logic of linear time or branching time (LTL, PLTL, CTL, CTL*, TCTL, etc.), or a μ-calcul.
- The verification phase: In this phase data are applied in an algorithm to check if the system model satisfies or not the specification model. This algorithm depends on the nature of the models chosen for the system and the property.

Among the possible models to describe a system and a given property, the choice is often a compromise between expressiveness and ease of analysis. There exist many tools which widely used such as SPIN [17] and NuSMV [18] to achieve this goal.

C. Kripke structure

Kripke structure [20] is used to provide semantics, which allow the checking, whether a specific property holds or not. The semantics are based on temporal logics for most of the widely used specification languages for reactive systems.

Let us assume, AP as a set of labels i.e., a set of atomic proposition such as variables, constants and predicate symbols.

A Kripke structure is a 4-tuple M = (S, I, R, $\mathcal{L}$) where:

- S is a finite non-empty set of states
- I □ S is a set of initial states
- R □ S × S associates with each state s □ S its possible successors are, $\forall s \in S, \exists s' \in S$ such that $(s, s') \in R$
- $\mathcal{L}$: S → 2^{AP} , associates with each state s □ S the set of atomic propositions $\mathcal{L}$(s) holds in s.

D. Linear Temporal Logic (LTL)

LTL [21] is the most commonly used language for specifying temporal properties of software or hardware designs.

The set of alphabets of LTL is composed of:

- Atomic proposition symbols: p, q, r …
- Boolean connectives : ⊤ (true), □ (false), ¬ (not), □ (or), □ (and), → (imply), ↔ (one-to-one)
- Temporal connectives : G, X, F, U

The set of LTL formulae is as follows:

- Any atomic proposition i.e. p, q is a formula.
- If □ and ψ are formulae, then ¬ □, □ □ ψ, □ □ ψ, □ → ψ and □ ↔ ψ are also formulae.
- If □ and ψ are formulae, then G□, X□, F□ and □ U ψ are formulae.

The four temporal connectives X, F, G, and U as shown in Fig.2, are explained as below:

- G ('always'): is read always in the future (in all future states of path). Graphically, it can be denoted as : □
- X ('next time'): is read at the next time (in the next state of path), and denoted as: ○
- F ('eventually'): is read eventually (in some future state of path, and denoted as: ◊
- U ('until'): is read until, which can be denoted as: **U**

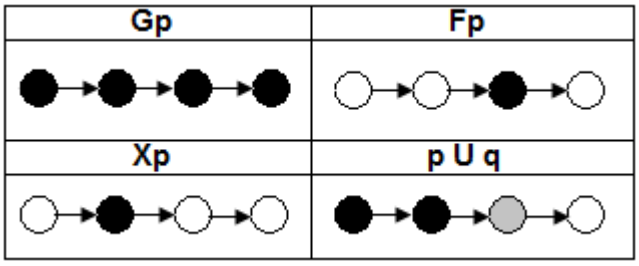

Figure 2. LTL Temporal connectives representation

1) Semantics of LTL: Let π = s_0, s_1, s_2…s_n be a sequence of states and L such as: $\forall i \geq 0, L(q_i)$ □ AP.

The sequence π satisfies □. It is denoted by π |= □. This relation can be defined inductively and gives semantics of LTL formulas as below:

- p □ AP, π |= p ↔ p □ L (q_0)
- π |= p □ q ↔ π |= p or π |= q
- π |= ¬ p ↔ π |≠ p
- π |= Xp ↔ π_1 |= p
- π |= p U q ↔ ∃ j ≥ 0, π_j |= q □ (∀k < j, π_k |= p).

IV. STRUCTURAL ERRORS

This section, presents some structural errors which can occur during run-time of BPMN process models. These are used to illustrate the proposed approach to ensure verification of the business process models.

A. Deadlock patterns

A deadlock in a process model is given if a certain instance of this model cannot continue working, while it has not reached the process end (i.e. deadlock is a condition used to describe a process that cannot be completed). According to Onada *et al.* [22] there are two complementary concepts, reachability and absolute transferability. Primarily, the reachability between process P_1 and process P_2, which means, there is at least one occurrence sequence from P_1 to P_2. Secondly, the absolute transferability, which means, it is a much stronger concept to state that a token can always be transferred from P_1 to all input points of process P_2. This makes absolute transferability to reduce reachability between two nodes, because of the existence of routing control nodes in between. The presentation of the business process by using only the reachability (without absolute transferability), can cause a deadlock.

In [22], the authors have also identified several potential causes of deadlocks, as follows:

- Loop deadlock: as shown in Fig.3.a, occurs when there is an execution path from the output of an *AND-join* back to its input points. If this path contains an *XOR-Split*, deadlock can occur if the branch leading to the loop is chosen. In case there is a path that does not contain *XOR-Splits*, deadlock occurrence is certain.
- Multiple sources: as shown in Fig.3.b, the multiple sources occur when two different sources lead at the input points of *AND-join* gateway. Assuming that none of the source nodes is the *AND-Join* itself, it can be observed that the multiple source patterns can occur when one of the process structure is as follows:
  - Any of the two sources is an *XOR-split* gateway.
  - The process has multiple start points that will be synchronized later. In case of models specified in BPMN, multiple starts are permissible. Actually, multiple start points resemble an *AND-split* gateway between the start events; hence we can deduce that there is reachability between two or more sources (start events) to the *AND-join* node.
- Improper structuring: as shown in Fig.3.c, occurs when an *AND-join* gateway receives input which contains *XOR-split*.

B. Livelock patterns

Livelock can be defined as a state from which it is possible to proceed, but it may be impossible to reach the desired final state.

As shown in Fig.3.d, the livelock can result an infinite execution of process. In this case some of the process may run successfully but some may trap in an endless loop of execution. This can happen when an *AND-split* is used instead of an *XOR-split* for modeling an existing loop.

C. Multiple termination patterns

The multiple terminations correspond to the situations where exists an *AND-split* before an *XOR-join* gateway, as shown in Fig.3.e

In this case, only one sequence is traversed when the exclusive gateway is executed. This case leads to the violation of soundness criterion. Thus, the BPMN process model does not terminate in the predefined (expected) terminate processes.

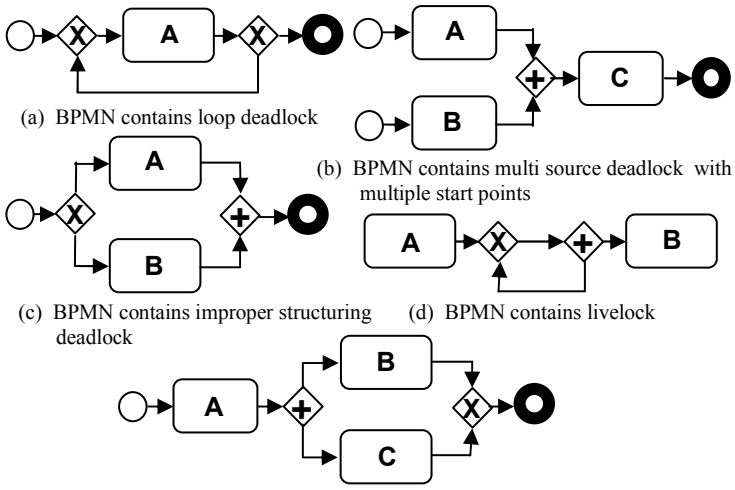

Figure 3. Structural error in BPMN

V. PROPOSED APPROACH

This section explains the detection of prominent structural errors which can occur in a BPMN process. Our approach is broadly described in Fig. 4. The main idea is to map BPMN process model into a finite-state model (Kripke structure) for specifying the system behavior. We also attempt to provide the means for LTL formulae for the compliance checking, that may lead to verify the existence of structural errors and ensure the soundness of process model. Several LTL properties can be defined simultaneously for a Kripke structure. The model checker provides, as a result, a counterexample, which verifies, in turn, the existence of structural errors in the BPMN process model. The verification steps are detailed, as follows:

A. Finite state generator

The business process models can be transformed or reduced to states and transitions between the states [23]. Furthermore, such automaton models may be subjected to an automated checking. Two typical approaches for such transformations can be found in [15] and [24]. In [15], the authors transform the business processes to Petri nets, followed by a transformation into Kripke structures which are then checked. Whereas, in [24], the author, transform the business processes directly into Kripke structures.

We translate BPMN process models directly into Kripke structure to express the behavior of the process models. The states of a Kripke structure represent the behavior of the process model. This translation facilitates the better verification of the desired temporal properties such as: $M \models \square$ iff $M, \pi \models \square$ for all paths π in a Kripke Structure *M*.

TABLE I. MAPPING OF BPMN OBJECTS TO KRIPKE STRUCTURE

BPMN Object	Kripke Structure
Start s	Start
End e	Final
Message M	E_M, M, exM
Task T	E_T, T, exT
T, Ex, T_x	E_T, T, exT; E_T_x, T, exT_x
A, B, C	E_A, A, exA; E_B, B, exB; E_C, C, exC
A, B, C	E_A, A; E_B, B; exA, exB; E_C, C, exC
A, B, C	exA; E_B, B, exB; E_C, C, exC
A, B, C	E_A, A, exA; E_B, B, exB; E_C

Figure 4. illustrates the translation of BPMN process model example to a Kripke structure

The finite non-empty set of states *S* of the Kripke structure represents the nodes *N* of the process model. *N* is a finite set of

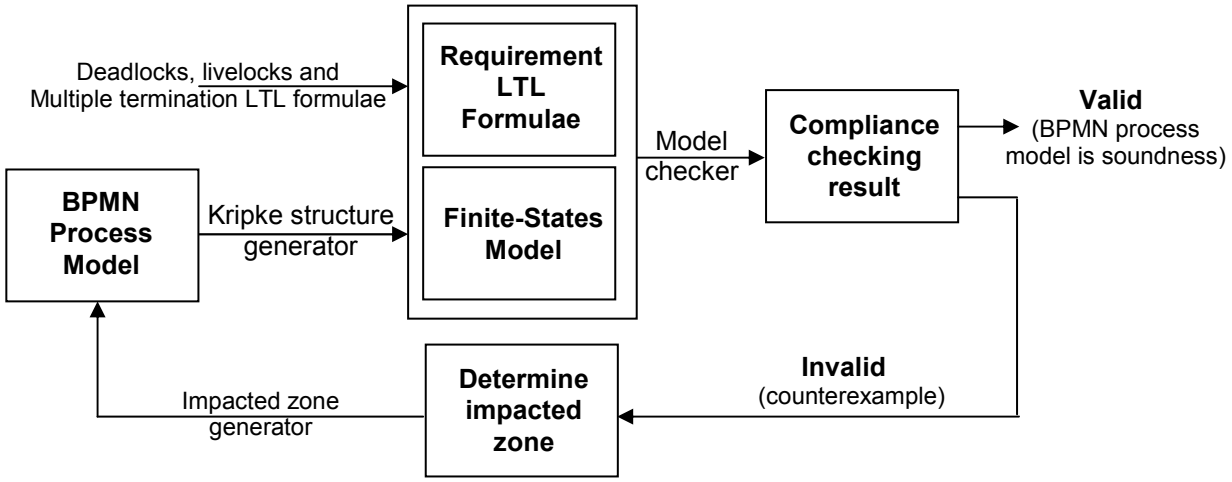

Figure 5. Compliance checking approach

flow objects in BPMN process which can be partitioned into events *E*, activities *A* and gateways *G*. The transition relations R represent the edge relations *T* (where, Transition $T \subseteq F \times F$ is a finite set of sequence flows connecting objects).

To obtain a Kripke structure, we define *AP* as the set of Atomic Propositions and assign labels to states. The defined set of atomic propositions *AP* is associated with each state $s \in S$ such as $\mathcal{L}(s)$ holds in *s* ($\mathcal{L}$ is the labeling function of Kripke Structure *M*). It expresses all properties of a given state.

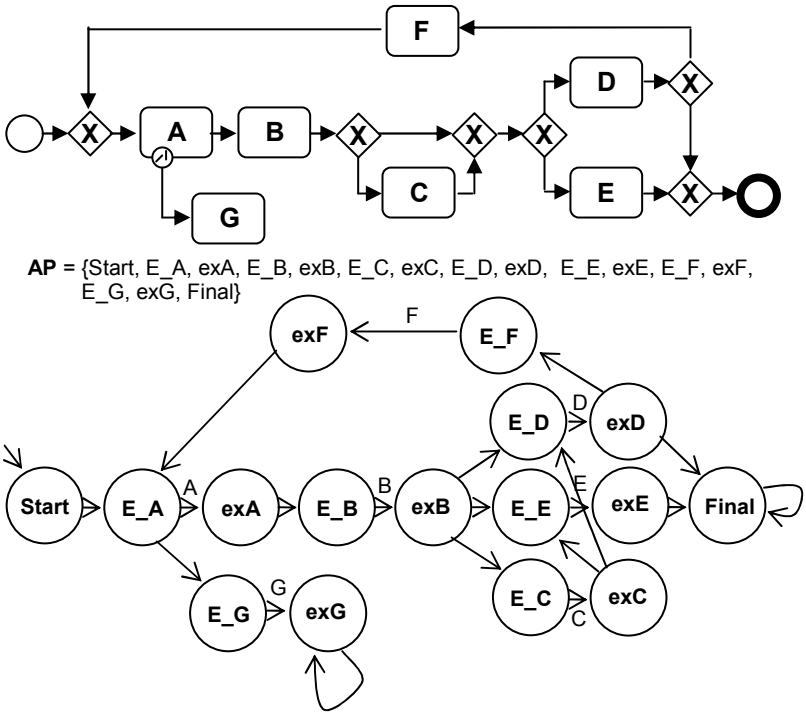

Figure 6. Mapping BPMN process model to Kripke structure

The initial state $I \subseteq S$ is the start point E^s (start event) of the process model. Each state s is labeled with enabled and executed transitions. Where, *E_A* signifies that the transition A is enabled and *ex_A* signifies that the transition A is executed (completed). A brief description of the mapping from a set of BPMN tasks, events, and gateways to Kripke structure is given in Tab. I.

Once the Kripke structure is obtained, we then proceed to define the desired correctness temporal properties.

B. LTL formulae generator

The soundness of BPMN process model to avoid structural errors can be ensured by satisfying the following temporal properties:

1) Detect absence of deadlocks

As mentioned above a deadlock situation can occur when no final state is reached, or some part of the process cannot be executed.

A Kripke structure is said to be deadlock-free if it does not contain any computation that can lead to a deadlock. The deadlock freedom is a safety property (i.e. something bad never happens). Let us assume a temporal formula (final), which represents the set of final states. In such a case, we can express deadlock-freedom by the following LTL formula:

$$\Box\ (\bigcirc\Box \rightarrow \mathit{Final})$$

This formula must be satisfied as valid on every path. The formula $\bigcirc\bot$ (means that "there is no next state") is easy to deduce, i.e. no transition is possible. Likewise, we can express reachability of a given deadlock state as the existence of a state with the dual property.

$$\Diamond\ (\bigcirc\bot \rightarrow \neg \mathit{Final})$$

2) Detect absence of livelocks

As described in section IV, the livelock state is a state from which it is impossible to reach the desired final state. A property which expresses the non-existence of livelock is a liveness property (i.e. something good eventually happens). A typical LTL formula is shown below:

$$\Diamond\Box\phi \rightarrow \Box\Diamond\psi$$

If a task tries to run infinitely, then it will be always in the execution state. This simplifies to ¬ ◊□□ (i.e. it will not succeed 'at run' forever). In A counterexample of these properties is an infinite execution according to which any of the expected behavior does not happens (i.e. the process does not terminate). Detection of a livelock (as explained in section IV Fig.3.d), can be expressed in the LTL formula, as shown below:

$$\Diamond\Box exA \rightarrow \Box\Diamond Final$$

3) Detect absence of multiple termination

To recall, the multiple termination is a situation in which there exists an AND-split before an XOR-join gateway. Detection of this case is based on checking the safety property of LTL (i.e. something bad never happens). It can be verified by the following formula:

$$\Box \neg (\Diamond (\phi \wedge \bigcirc\psi) \rightarrow Final)$$

A counterexample of these properties is a finite execution which leads to unexpected behavior.

C. Model Checking

The finite state machines and the temporal logic formulae are presented as input to a model checker. The model checker verifies whether the temporal logic formulae are respected by the given finite state machines or not. As a result, it confirms the soundness of the process models. Otherwise, it returns a counterexample in cases of structural errors.

D. Determine impacted zone

Ensuring the unsoundness of business process is necessary to help modelers to avoid structural errors. Formal approaches such as model checking have the capability of providing counterexamples when the temporal properties to be checked are not satisfied by the process model [25, 26]. Mostly, these counterexamples are given in terms of internal state transitions rather than in terms of process models that are difficult to understand by a non-technical user. To benefit from these counterexamples, the output of the model checker should be translated in the visual notation, which is easier for the user to understand.

To map a counterexample to the source BPMN process model supports the better determination of the impacted zone (by structural errors). We use model checker dependency, it contain a tool chain that translates the output of the model checker back to the process model notations. This allows us to map each state to the original BPMN process model element and colored as red to highlight the errors to business modelers, in order to correct them.

To illustrate further, we take the example of multi source deadlock with multiple start sources (as depicted in section IV). The verification process is described in Fig.6.

If we observe execution sequences of above process model, the following execution traces can be obtained:

σ^1_1 = ({S_1}, {E_A}, {exA, exB})

σ_2 = ({S_2}, {E_B}, {exA, exB})

σ_3 = ({S_1}, {S_2}, {E_A}, {E_B}, {exA, exB}, {E_C}, {exC}, {Final})

If execution sequences σ1 and σ2 refers that model checker return a counterexample where S_1 and S_2 (start points) did not start synchronously, that causes a deadlock because $\mathcal{L}$ (*exA, exB*) cannot hold in {*exA, exB*} state, which is not a final state. So, the LTL formula to detect the absence of deadlock is not satisfied. The only condition for the process model to work correctly and to satisfy the LTL formula (to detect the absence of deadlock) is that the S_1, S_2, *E_A, E_B* start and run synchronously in corresponding branches (as is the case of σ3).

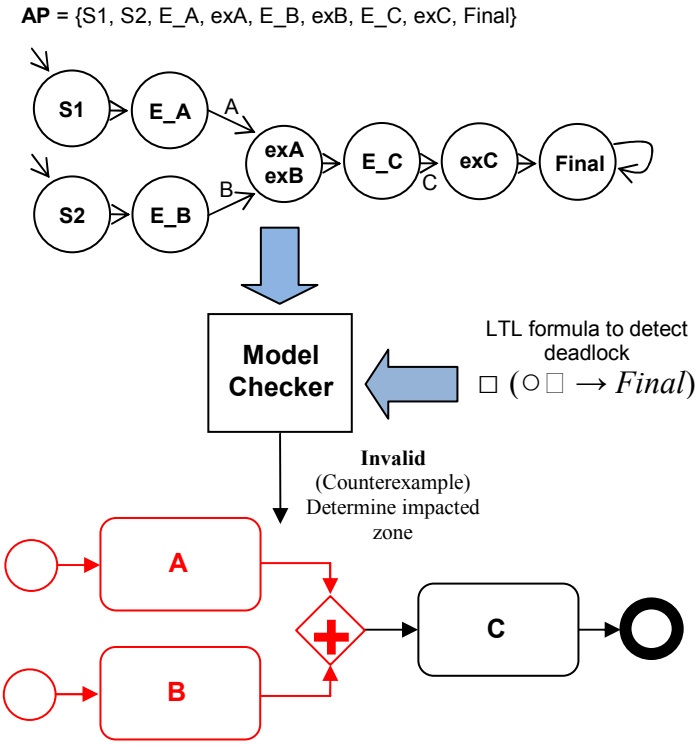

Figure 7. Detection of multiple source deadlock with multiple start points

Currently, we implement a prototype tool to validate the presented approach for detecting structural errors in BPMN. It is developed as a set of Eclipse IDE plug-ins. We make use of Eclipse BPMN 2.0 Modeler[1] plug-in. As model checker, we opted for EpiSpin[2] plug-in. The objective is to transform automatically the BPMN process model to Promela model. This is, then, provided as input along with pre-defined LTL formulae to the EpiSpin model checker to detect the deadlocks, live-locks, and multiple terminations.

VI. CONCLUSION

Using the model checking techniques can help to better detect structural errors in process models. The automated

[1]svn+ssh://svn.java.net/bpmn-modeler~source-code-repository/
[2]http://epispin.ewi.tudelft.nl/

checking of such errors has two basic advantages i.e. to compute the polynomial time and the error traceability.

The presented approach emphasizes to map the BPMN process model to Kripke structures to express the behavior of the process models. The resulted mapping is used to satisfy the temporal properties (e.g. absence of deadlocks, livelocks and multiple terminations), which are expressed using the Linear Temporal Logic (LTL) formulae.

We discuss, in detail, the error cases and the properties to be checked to validate the verification of structural errors via model checking. The generated finite states (Kripke structures) are validated in conformance with LTL formulae. The resulting compliance checking can verify the soundness of the process model, otherwise it return a counterexample, which can facilitate to determine the impacted zone.

The objective is to provide assistance to the process modelers for the better detection of errors and their correction. In the future, we intend to continue this approach and particularly to focus the compliance checking rules for the post change scenarios.

REFERENCES

[1] R.Lu "Constraint-Based Flexible Business Process Management," in School of Information Technology and Electrical Engineering, University of Queensland, 2008.

[2] W. van der Aalst, and al, "Business Process Management: A Survey," in Proceedings of Conference on Business Process Management (BPM 2003), Eindhoven, Netherlands 2003.

[3] Object Management Group. BPMN 2.0: OMG final adopted specification DOI= http://www.omg.org/spec/BPMN/2.0/PDF.

[4] I. Kitzmann, C. Konig, D. Lubke, and L. Singer, "A simple algorithm for automatic layout of bpmn processes," in CEC '09: Proceedings of the 2009 IEEE Conference on Commerce and Enterprise Computing. Washington, DC, USA: IEEE Computer Society, 2009, pp. 391–398.

[5] Bizagi Process Modeler DOI=http://www.bizagi.com/

[6] Intalio designer DOI=http://www.intalio.com/

[7] Bonita designer DOI=http://fr.bonitasoft.com/

[8] Kristin Y. Rozier. 2010. Linear Temporal Logic Symbolic Model Checking. NASA Ames Research Center,Moffett Field,CA 94035,USA.

[9] Temporal Logics and Model Checking DOI=http://users.encs.concordia.ca/~tahar/coen7501/notes/3-mc-02.05-4p.pdf

[10] B. F. van Dongen, M. H. Jansen-Vullers, H. M. W. Verbeek,and W. M. P. van der Aalst, "Verification of the sap reference models using epc reduction, state-space analysis, and invariants," Comput. Ind., vol. 58, no. 6, pp. 578–601, 2007.

[11] R. M. Dijkman, M. Dumas, and C. Ouyang, "Formal semantics and automated analysis of bpmn process models," Tech. Rep., 2007.

[12] A. Awad and F. Puhlmann, "Structural detection of deadlocks in business process models," in BIS, 2008, pp. 239–250.

[13] Ralf Laue, A. Awad, "Visual suggestions for improvements in business process diagrams," J. Vis. Lang. Comput. 22 (5): 385-399 (2011).

[14] Awad, A, "BPMN-Q: A Language to Query Business Processes," In: EMISA, pp.115–128 (2007).

[15] W. van der Aalst, "Formalization and verification of event driven process chains," Information and Software Technology, vol. 41, no. 10, pp. 639–650, July 1999. [Online]. DOI=http://dx.doi.org/10.1016/S0950-5849(99)00016-6

[16] N. Tantitharanukul and al, "Detecting deadlock and multiple termination in BPMN model using process automata" Electrical Engineering/Electronics Computer Telecommunications and Information Technology (ECTI-CON), 2010.

[17] Gerard J. Holzmann. 1997. The Model Checker SPIN. In Proceedings of IEEE Transactions on software Engineering – Special issue on formal methods in software practice. (IEEE Press Piscataway, NJ, USA).

[18] A. Cimatti and al, "NUSMV: a new symbolic model checker (2000) " International Journal on Software Tools for Technology Transfer.

[19] Tutorial Model Checking DOI=http://etr05.loria.fr/slides/mardi/slides.pdf

[20] M. C. Browne and al,."Characterizing finite Kripke structures in propositional temporal logic". Theoretical Computer Science - International Joint Conference on Theory and Practice of Software Development. Elsevier Science Publishers Ltd. Essex, UK.

[21] (Temporal) Logic Tutorial DOI=http://www.scss.tcd.ie/edsko.de.vries/talks/temporal_logic.pdf

[22] S. Onada and al, "Definition of deadlock patterns for business processes workflow models". In HICSS '99: Proceedings of the Thirtysecond Annual Hawaii International Conference on System Sciences-Volume 5, pages 50–65, Washington, DC, USA, 1999. IEEE Computer Society.

[23] Mahleko, B., Wombacher, A.: "Indexing Business Processes based on Annotated Finite State Automata," In: IEEE International Conference on Web Services (ICWS 2006). pp. 303–311. IEEE Computer Society, Los Alamitos, CA, USA (2006).

[24] Pulvermüller, E.: "Composition and correctness," Electronic Notes in Theoretical Computer Science (ENTCS) 65(4) (2002) .

[25] Y. Lui, S. Müller, and K. Xu.: "A static compliance-checking framework for business process models," IBM SYSTEMS JOURNAL, 46(2):335-362, 2007.

[26] A. F orster, G. Engels, and T. Schattkowsky.: "Activity diagram patterns for modeling quality constraints in business processes," In MoDELS, pages 2{16}, 2005.