Enhancing Robust Execution of BPMN Process Diagrams: A Practical Approach

Hodjat Soleimani Malekan, Mohammad Shafahi, Naser Ayat, Hamideh Afsarmanesh

Enhancing Robust Execution of BPMN Process Diagrams: A Practical Approach

Hodjat Soleimani Malekan, Mohammad Shafahi, Naser Ayat, Hamideh Afsarmanesh

Informatics Institute, University of Amsterdam, 1098XH, Amsterdam, the Netherlands
{h.soleimanimalekan, m.shafahi, h.afsarmanesh}@uva.nl; naser.ayat@gmail.com

Abstract. As a standard modeling language for definition of business processes and services, BPMN is used both within the organizations as well as for co-creation of joint services to run among the networked organizations. However, introducing certain constructs, such as OR-join and Complex-join in BPMN Process Diagrams (BPDs) can lead to execution problems, due to ambiguities inherent in these constructs in relation to their execution semantics. Although these constructs are often used to represent the real-world behaviors, none of the existing approaches applied by the BP management systems are practically tuned to disambiguate and support their proper execution. Rooted in workflow patterns concept, we first introduce a set of algorithms to automate the identification of ambiguous patterns (i.e., workflow patterns that include OR-join constructs). Then, we introduce a set of equivalent unambiguous BP fragments that can substitute those ambiguous patterns. To this end, we have conceptualized the identification of three OR-join ambiguous patterns by applying the RPST technique, represent a set of unambiguous solutions for their substitution, and implemented our method as the proof of concept for our approach.

Keywords: Business Process Execution, BPMN, OR-join, Workflow Patterns, Ambiguous Patterns.

1 Introduction

Business Process Model and Notation (BPMN) [1] is a standard modeling language for describing Business Processes (BPs). The primary objective of BPMN is to provide a standard notation that is intuitive enough to be understood by business users (e.g., domain experts). In addition, it is expressive enough to be executed by technical users (e.g., BP analysts) [3]. To this end, BPMN Process Diagram (BPD) is an effective means for designing and executing BPs by domain experts and BP analysts, respectively.

Through its semi-concise definition, besides supporting the actors internal to each organization, BPMN adequately supports Collaborative Networked Organizations (CNO) [19] in consolidating their BPDs [20] and co-creating business services [17] that can jointly run among different organizations. In particular, the provision of

relevant features, as well as the expressiveness, and understandability of the BPMN notations assist the BPD designers in CNOs to accomplish their mission [2]. The need for definition of joint business services in CNOs is addressed in the literature in the past [17, 18]. We have also studied and reported some real-world cases in [16], which evaluates the suitability of BPMN for CNOs. In this real application case, in order to construct Eco-friendly buildings, a general contractor establishes a CNO together with its collaborators. Then the established CNO successfully defines, aligns, and consolidates independent BPs from different organizations toward delivering a number of shared business services, all represented by BPDs. During this exercise; however, we discovered several executability concerns about those defined BPDs that used "OR-join" to combine the BPs from different CNO members, as also reported in some other literature [9, 10]. In this paper, we address a number of executability problems related to BPDs.

To generate an executable BPD, the use of some problematic BPMN constructs, such as OR-join [3, 4] and Complex-join [5] makes run-time problems, because of ambiguities in their execution semantics. For instance, the existence of OR-join gateway in BPDs, which is used to capture the synchronization behaviors, causes the BP management systems (e.g., Activiti) have no idea of for how many paths of the join gateway should wait before proceeding to the next step [3, 4].

Nevertheless, one challenge to resolve the problematic constructs is well understanding the molder's intention behind the applied constructs. This, in turn, helps with offering equivalent executable BP fragments, to be substituted with the existing BP model. In this regard, we have applied the concept of *workflow patterns* [6, 7], which is designed to provide a rigorous basis for different aspects of BP technologies (e.g., BP execution). Hence, we are aimed at identifying and resolving the certain OR-join included workflow patterns, so-called *ambiguous patterns* in this paper. In practice, the ambiguous patterns are not properly executed by any of the many BP management systems that support BPMN [3].

To deal with the problem of ambiguous patterns in BPDs, we design a solution that: (i) identifies three ambiguous patterns that include OR-join (e.g., general synchronizing merge), (ii) suggests unambiguous equivalent workarounds for disambiguating by the users, and (iii) introduces a Disambiguation Support System tool to interact with business users on identifying and resolving ambiguities.

Consequently, for the rest of this paper, we first explain the BPMN notation, briefly. Second, we explain why OR-join is ambiguous, and define and exemplify the ambiguous workflow patterns. Next, the algorithm for the identification of ambiguous patterns, as well as the unambiguous workaround BP fragments are proposed. Then, the architecture of our developed Disambiguation Support System is explained. After discussing the results of our research, the conclusion is presented.

2 Business Process Modeling and Notation

BPMN (ver. 2.0.2) is a popular BP modeling language standard that targets both appropriate visualization and executive semantics [1]. Because of its rich graphical

notation and XML exchange definitions, it is used pervasively by organizations [2]. To represent BP, BPDs are produced by using BPMN elements.

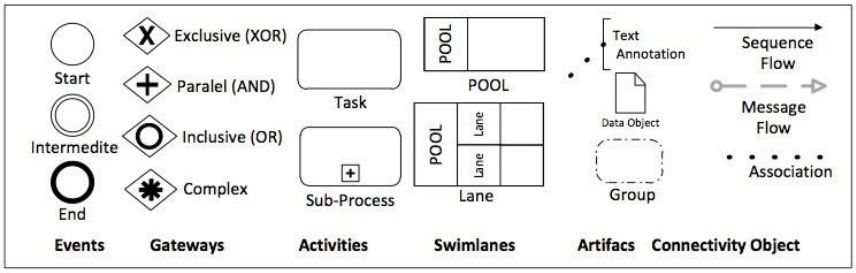

Fig. 1. A set of BPMN Elements

In Fig. 1, the main elements of BPMN are illustrated into six categories of "Events", "Activities", "Gateways", "Connectivity objects", "Swimlanes", and "Artifacts". In principle, gateways control the *split* and *join* in the flow of processes. In the *split* construct, several paths diverge from the gateway, while in the *join* construct, several paths converge into one path. Similar to XOR logical operator in programming languages, the continuation of the process flow in an XOR (exclusive) gateway is mutually exclusive. Also, in the case of having multiple incoming paths into an exclusive gateway, the flow continues only if one of the incoming paths is activated. The AND (parallel) gateway denotes the concurrent trigger of all paths that either come to or go out of the gateway.

The OR (inclusive) gateway combines the behaviors of both the exclusive gateways and the parallel gateways. Thus, the trigger of one, a certain number, or all of the outgoing path(s) is possible in a split condition. Similarly, in a merge condition, process flow continues after receiving one, a certain number, or all of the incoming activated path(s). The Complex decision gateways are suitable for expressing decision points by the BP rules (e.g., when *n* paths out of *m* existing paths can be chosen).

Activities show the tasks that are being carried out in a process. Modelers can localize a set of related activities of a process in one sub-process. Selecting the level of details about activities depends on modeler's perception [3]. Events can appear at the beginning, in the middle, and at the end of processes, called start, intermediate, and end events, respectively. Circles in BPMN notation depict event constructs (see Fig. 1). Artifacts in BPMN notation show some extra necessary explanation about a process. Depicting text annotations, data object, and group, in turn, facilitates the perception of BPDs.

Connectivity objects represent the three main flow streams in BP diagrams: "sequence flow", "message flow", and "association". Sequence flows show the execution order of activities. The direction of the exchanged message is illustrated by message flow. Association links depict the direct interactions between BPMN elements. Swimlanes denote resources such as process participants and roles in processes. Furthermore, swimlanes represent either one organizational role in a Lane or a set of business participants in a Pool.

In spite of its rich set of elements, BPMN executions face difficulties, because of unclear execution semantics. The semantics of BPMN is described textually and is not formally standardized [4]. This issue can result in ambiguities, because of either failure in formalizing some BPMN constructs (e.g., OR-join construct), or different potential interpretations of the standard (e.g., Complex-join construct). As such, the robust execution of BPDs that include OR-join [3] and Complex-join [5] can lead to inconsistencies in executability of a BPD. The BP management systems also lack the execution features for supporting the mentioned ambiguous constructs [3, 8].

Although those two ambiguous constructs are difficult to elaborate, in certain cases applying OR gateways or Complex gateways are inevitable. For instance, the OR-join is appropriate for combining two different behaviors in merging two BP models. Similarly, the complex synchronization behavior, which is captured by Complex-join gateway has no substitution with split and join constructs of other BPMN gateways [5]. In this paper, we focus on OR-join ambiguity resolution. In the next section, we take a closer look at the reasons behind the OR-join ambiguity in BPDs.

3 Why is the OR-join Construct Ambiguous?

In order to represent the synchronization, the OR-join constructs are used. More precisely, the OR-join synchronizes some of its incoming paths that their precise number is unknown until the execution time of a BPD.

Note that, during the execution of a BPD, the current point of execution is indicated by the concept of *the thread of control*. The execution starts, when the thread of control is located at the start point. Accordingly, the thread of control traverses from the *start* point to the *endpoint*. In the meanwhile, a thread of control can split into several distinct threads after a diverging point (see Si in Fig. 2), or multiple distinct threads are joined into a single thread of control in a converging point (see Ji in Fig. 2). Borrowed from Petri net notation [3], the BPMN standard document [1] uses the notion of "token" (the black circles on gateways in Fig. 2) to explain the states, which are traversed by different threads of control.

The OR-join serves as a converging point that waits for all tokens produced *upstream* on all its incoming paths to arrive. By reaching the state that no active token can come along a particular incoming path, the OR-join is enabled [1]. Yet, it is not clear what is considered within the range of the term "*upstream*" associated with an OR-join [9]. We describe the threads of control in an OR-join by an example, illustrated in Fig 2.

The OR-split Si produces tokens for all or some of its outgoing paths, i.e., either one task, such as "Task A", or two tasks, e.g., "Task A and Task B", or all three tasks (i.e., A, B, and C) are enabled. The corresponding OR-join, Ji, is supposed to synchronize the paths, which are enabled by Si. In brief, enabling Ji depends on information of the enabled paths that lead to the OR-join, called the non-local information. Various situations of this issue are shown in parts (a) and (b) of Fig. 2.

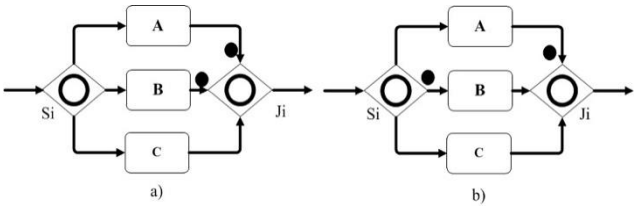

Fig. 2. An example of OR-join enablement

In part (a), Ji is enabled. Yet, in part (b), Ji is supposed to wait for the upstream token until it arrives after passing the "Task B". Both parts of Fig. 2 have the same incoming paths of Ji; nevertheless, the position of remained tokens determines the activation of Ji [10]. In summary, enabling decision of OR-join requires the knowledge of the current state as well as possible future states of the process. In practice, determining all possible states of the BP becomes very complicated when other constructs, such as multiple OR-joins, cycles, or cancellation exist in a BPD [9].

The OR-join enabling decision in different proposed approaches, namely [9], is based on the information of the current state as well as the possible future states of the BP model. The formal techniques introduced; hence, seek to determine all possible states of the BP model during the execution which are computationally expensive. Accordingly, introducing a "clean semantic" for the OR-join construct is almost impossible, due to numerous configurations in BP models, e.g., vicious circle [3,4].

Removing the OR-join poses three challenges. First, how to preserve the behavior of the OR-join included BPD, which is initially intended by the modeler. The OR-join construct is used to show the synchronized behavior; thus, the substitution of this construct should represent both the synchronization and flexible behavior of OR (i.e., the combined behavior of both AND/ XOR gateways) [10]. Second, the executability of the BPD with OR-join constructs is a concern, while, no fully automated solution has been implemented [3]. Finally, the solutions should be understandable by BP modelers [10]. Consequently, we intend to identify the ambiguous fragments (i.e., OR-join included fragments) and provide equivalent workaround solutions to be substituted with thereof. Moreover, the usability of solution by the domain experts and BP analysts matters for us in offering our disambiguating solution. To provide a systematic solution for both identifying and resolving the ambiguous BP models, we investigate the occurrence of ambiguities in the context of workflow patterns [6, 7].

4 Ambiguous Workflow Patterns

The workflow pattern initiative, introduced by van der Aalst et al., is an attempt to identify the recurring patterns to provide a basis for enhancing BP modeling languages and evaluating commercial tools [6]. To support different behaviors and functions (so-called control-flow) in BPs, 43 patterns has introduced in [6], which are recently delineated in [7]. Supporting more patterns indicates the higher level of

expressiveness of a modeling language. BPMN is more expressive in comparison with most of the graphical modeling languages [6], such as EPC or UML. Nevertheless, three of the BPMN-represented workflow patterns use the problematic OR-join construct to represent the *synchronizing* behavior.

The three ambiguous patterns are the variants of "synchronizing merge" control-flow pattern. Generally, in a *synchronizing merge* pattern, two or more incoming paths, which are previously diverged from a unique split point, converge into one particular join construct [6]. In BPMN, the OR-join construct controls the convergence point and passes the thread of control to succeeding element [7]. Beyond the general mentioned structure, each of the three synchronizing merge variants is distinguished by its specifications and the way it provides information for OR-join construct to pass the thread of control to the next step. Here, we explain and exemplify the three variants of: "*structured*", "*local*" and "*general*" synchronizing merge [6].

First, the *Structured synchronizing merge* pattern is a balanced BP fragment, and no further split and join can appear in the way from divergence point (i.e., OR-split) towards convergence point (OR-join).

For example, the product line of a company is audited according to a quality management plan. In case of nonconformity, a set of preventive actions and/or corrective actions are taken to eliminate the nonconformity. The annual audit ceremony is closed when neither any nonconformity is identified, nor any preventive and/or corrective actions are left. The BPD of this example is illustrated in Fig. 3 where the structured synchronizing merge area is enclosed by dashed-line.

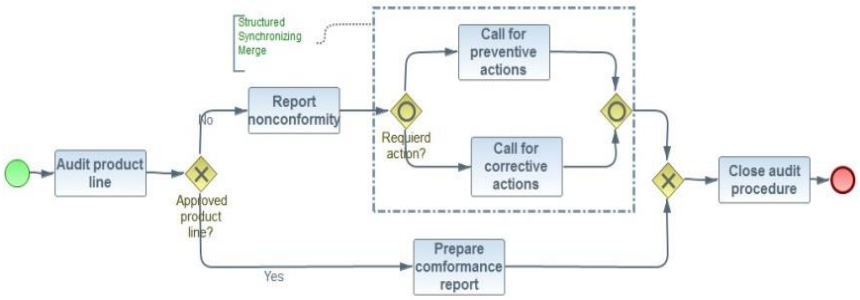

Fig. 3. Structured synchronizing merge in BPMN

Second, the local synchronizing merge pattern can decide on synchronization based on the available local information. Unlike the structured synchronizing merge, the local synchronizing merge fragments can include cycles. Although the cycle within the BP model, must either completely locate on a single path coming from the split point to the join point, or the entire OR region should be contained in a cycle [7].

For example, consider in the previous example, taking the preventive actions can is followed by a review task. This cycle is illustrated by adding two XOR-gateway in the dashed-line area in Fig. 4. In this case, the control XOR- gateway can take the thread of control out of cycle, it is decidable whether the OR-join will be enabled or not.

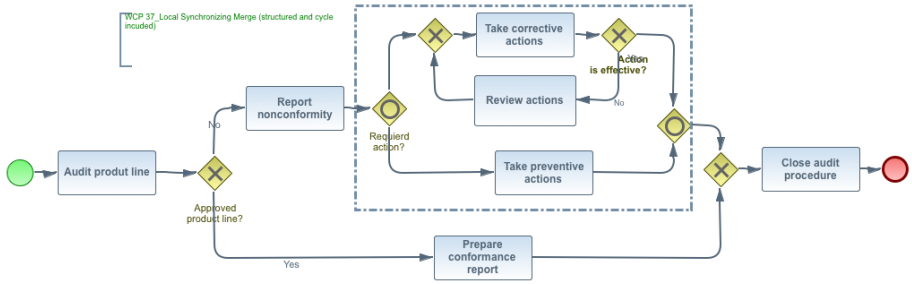

Fig. 4. Local Synchronizing Merge in BPMN

Finally, the general synchronizing merge is neither structured nor can be enabled without analyzing all possible future states. Although it covers more OR-join included BP models, it faces serious problems during the execution time. Besides, any cycle can exist in this pattern.

For example, regarding the quality auditing scenario, considering that for checking the efficiency of the preventive actions, the outsourcing is continuously checked. If it needs an external third-party for outsourcing, the "Plan outsourcing" task is performed. Otherwise, the efficiency of the preventive task is confirmed by the committee. In Fig. 5, the XOR-gateway to "required external resource" can take the thread of control out of OR-region.

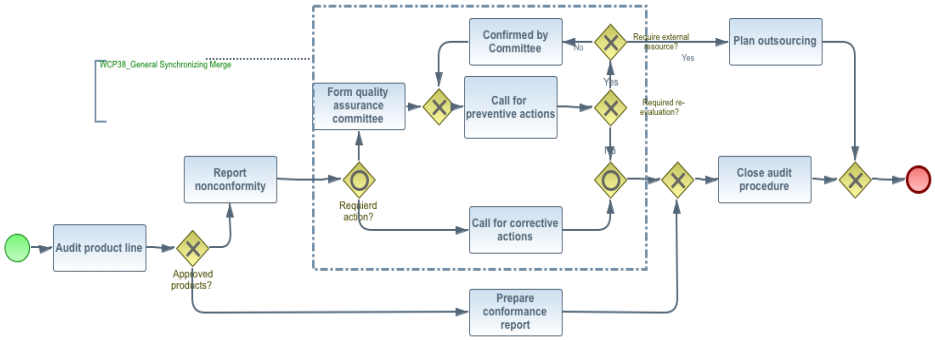

Fig. 5. General Synchronizing Merge in BPMN

None of the presented solutions for OR-join semantics, such as in [9, 10] are implemented and fully supported by main BP management systems, such as jBPM[1], Activiti[2], and Camunda[3] [8].

[1] https://www.jbpm.org
[2] https://www.activiti.org
[3] https://www.camunda.org

5 Algorithm for Ambiguous Pattern Identification

Let us, first, explain briefly the intuition behind our algorithm. To identify the mentioned ambiguous patterns in a given BPD, the Refined Process Structured Tree (RPST) technique is used [11]. The RPST technique hierarchically decomposes a BP model into a set of fine-grained fragments, which have "single-entry-single-exit" (SESE) boundary nodes. The SESE fragments provide a sound basis for recognizing an ambiguous pattern in BP models. Because, the single-exit node of the fragment facilitates the identification of OR-join constructs. Comparably, the corresponding entry point of the fragment can enclose the ambiguous pattern's region. Thus, we can precisely determine ambiguous fragments, regardless of the BP models' elements between the split and join points. Furthermore, the processing time for making the RPST is linear and the implementation of the algorithms is not computationally expensive [11]. Fig. 6 shows an example of RPST decomposition.

We consider a BPMN Process Diagram (BPD) as the input for our pattern identification purpose. The corresponding definition of the BPD model is captured, as follows:

Definition 1 (BPMN Process Diagram). A BPMN Process Diagram is a graph-based representation of a business process. BPD is a tuple of **BPD = (T, GX, GA, GO, GC, E)**; where T is a non-empty set of tasks, and G is the superset of gateways, such that $\mathbf{G = GX \cup GA \cup GO \cup GC}$, where GX is the set of XOR gateways, G is the set of AND gateways, GO is the set of OR gateways, and GC is the set of Complex gateways, these sets are all disjoint. All nodes in a BPD are defined by $N = G \cup T$. The control flow, which is the set of direct edges between two nodes in a BPD, is defined by $E \subseteq N \times N$. Each fragment (say F) is a non-empty directed sub-graph of a BPD, **F = (NF, EF)**, where, $NF \subseteq N$ and $\mathbf{EF = E \cap (NF \times NF)}$.

Definition 2 (Well-structured BPD). Two ordered sets of predecessor and successor nodes are defined for each node (i.e., a task or a gateway). For instance, regarding a task $\tau \in T$, $\bullet\tau$ stands for the set of immediate predecessor nodes, i.e., $\mathbf{\bullet\tau = \{\eta \in N \mid (\eta, \tau) \in E\}}$, and $\tau\bullet$ is the set of immediate successor nodes of τ; $\mathbf{\tau\bullet = \{\eta \in N \mid (\tau, \eta) \in E\}}$.

In a well-structured BPD, there is at most one incoming sequence path (i.e., edge) and one outgoing sequence path for every task τ, such that $| \bullet\tau | \leq 1 \wedge | \tau\bullet | \leq 1$. There exists only one start node, such that $\eta \in T$ and $| \bullet\eta | = 0$, and only one end node, where $\eta \in T$, if $| \eta\bullet | = 0$. A gateway can be either a split or a join. A gateway "g_s" represents a split gateway, if $| \bullet g | = 1 \wedge | g\bullet | > 1$. In a similar way, "g_j" signifies a join gateway, if $| \bullet g | > 1 \wedge | g\bullet | = 1$; thus "go_j " is an example of OR-join gateway.

Definition 3 (Paths). There exists a path between η and η', if an ordered set of nodes are in sequence, such that $\langle (\eta 1, \eta 2), (\eta 2, \eta 3), \ldots, (\eta x{-}1, \eta x) \rangle$, where $(\eta = \eta 1) \wedge (\eta' = \eta x) \wedge \forall 1 \leq i < x\text{-}1\ (\eta i, \eta i{+}1) \in E$; $i, x \in N$. The path is signified by $\mathbf{\eta \rightarrow \eta'}$.

There is a cyclic path, if a path exists that starts and ends at the same node. Hence, a BPD has a cycle, if $\exists\ \eta \in N$, $\mathbf{\eta \rightarrow \eta}$.

For η, γ ∈ BPD, the node η is dominate γ, if all paths from a start node to γ include η. For example, in a fragment of a BPD, if all paths from the start to γ includes the node η, then "η" dominates "γ", denoted by **η Dom γ**. Comparably, a node η post-dominates node γ, if all paths from γ to an end, include η, which is signified by **η PDom γ.**

Definition 4 (Refined Process Structure Tree). Decomposition techniques, by definition, produce a non-empty set of fragments from a graph, such as a BPD. The refined process structured tree introduced in [11], decomposes a BPD into SESE fragments, which are connected to the rest the BP model via two nodes. A fragment is SESE, if an ordered edge pair (η, γ) meets three conditions [11]: (i) η **Dom** γ, (ii) γ **PDom** η, and (iii) every cycle that contains η also includes γ and vice versa.

A process tree is comprised of canonical fragments. If the fragment F is canonical, $\forall F' \in N, F \neq F' \Rightarrow (EF \cap EF' = \emptyset) \vee (EF \subset EF') \vee (EF' \subset EF)$. Hence, the canonical fragments, called *fragments* for the rest of this paper, either follow a hierarchy or a sequence relation. These relations are represented by a unique tree, called RPST, where the parent of each fragment is the smallest fragment that contains it [11].

The SESE region is connected to the other fragments, with two boundary nodes as the entry/exit nodes, such that no incoming/outgoing edge of them are in F. A boundary node "η" is an entry, where $\boldsymbol{\varepsilon ntry(F) = E \cap ((N \backslash NF) \times NF)}$, or can be an exit point, where $\boldsymbol{\varepsilon xit(F) = E \cap (NF \times (N \backslash NF)}$.

Definition 5 (RPST fragments' classes). Let F be a fragment of a BPD. It can appear in four classes of: "*trivial*", "*polygon*", "*bond*", and "*rigid*" [11], as follows:

F is a *trivial* fragment (T), if F is a singleton that includes only a single sequence arc, such as a sequence flow in a BPD.

F is a *polygon* fragment (P), if there is a sequence of fragments of (z1, z2, . . ., zn), and n ∈ ℕ, where the entry of z1 is the entry of F, the exit of zn is the exit of F, and for all i that 1 ≤ i ≤ n the exit of zi, is the entry of zi+1, therefore, $\mathbf{F = \bigcup_{mi=1} zi}$. In Fig. 6, "P2", is a polygon that includes the sequence of: "Task A" and "B1 fragment".

F is a *bond* fragment (B), if it includes all fragments "z" that share the same entry and exit nodes with F, such that $\mathbf{F = \bigcup_{z \in C} Z}$. Where, "C" is the set of the canonical fragments with common boundary nodes (i.e., entry and exit) with "F". For instance, the two OR-gateways before and after task "C" and task "D" in Fig. 6 are the entry and exit of the bond fragment "B2". Bond fragments are thus balanced by definition.

F is a *rigid* fragment (R), if it is none of the all above classes of fragments (i.e., trivial, polygon, and bond). For instance, "R1" is a rigid fragment, shown in Fig. 6. The rigid fragments are arbitrary and unstructured [11].

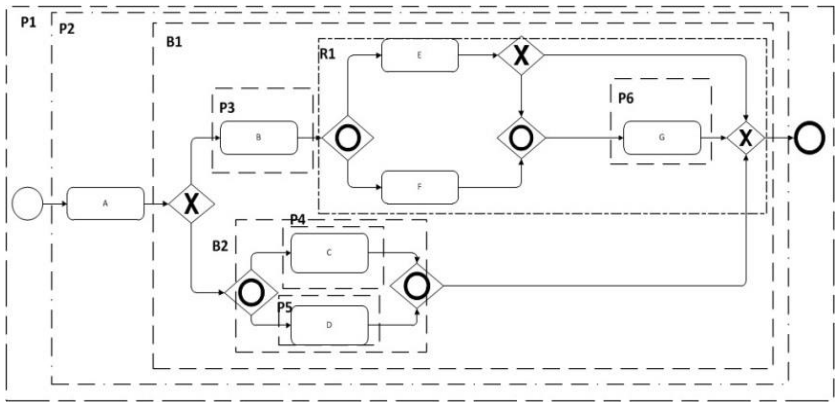

Fig. 6. An Example of RPST decomposition of a BP

The Algorithm 1 summarizes our ambiguous pattern identification approach.

Algorithm 1 Identifying OR-join included patterns

Require: BPD: A BPMN Process Diagram
Ensure: S: A set S of OR-join included patterns in BPD
S ←∅
Construct the RPST of BPD in accordance with [11]
Let F be the set of all fragments in the RPST
Let B be the set of bond fragments in F
Let R be the set of rigid fragments in F
for all b ∈ B do
 if (εxit(b) ∈ GO_J) ∧ (εntry(b) ∈ GO_S) then
  if ((∃ η → η) ∧ (⟨η → η⟩ ⊂ ⟨εntry(b) → εxit(b)⟩)
   S ← S ∪ (b, "Local synchronizing merge")
  else if (∄ η → η) then
   S ← S ∪ (b, "Structured synchronizing merge")
 end if
end for
for all r ∈ R do
 if (go_j ∈ GO_J)∧(go_s ∈ GO_S)∧(go_s **Dom** go_j)then
  if (∃ γ → γ) ∧ (⟨γ → γ⟩ ⊂ ⟨go_s → go_j⟩) then
   S ←S ∪ (r, "Local synchronizing merge")
  else
   S ← S ∪ (r, "General synchronizing merge")
 end if
end for
return S

6 Ambiguous Pattern Resolution

To substitute the ambiguous OR-join included patterns, we suggest three equivalent BP fragment represented in [12, 13], illustrated in Fig. 7. The workaround solutions are composed by "AND-"and "XOR-" gateways; thus, they are executable. Furthermore, they are structured and more understandable by users [3]. Besides, users are able to focus on specific BP fragment (selected as the ambiguous pattern) for restructuring a BPD, which greatly assists users when they work with the arbitrary structures of local and general synchronizing merge patterns.

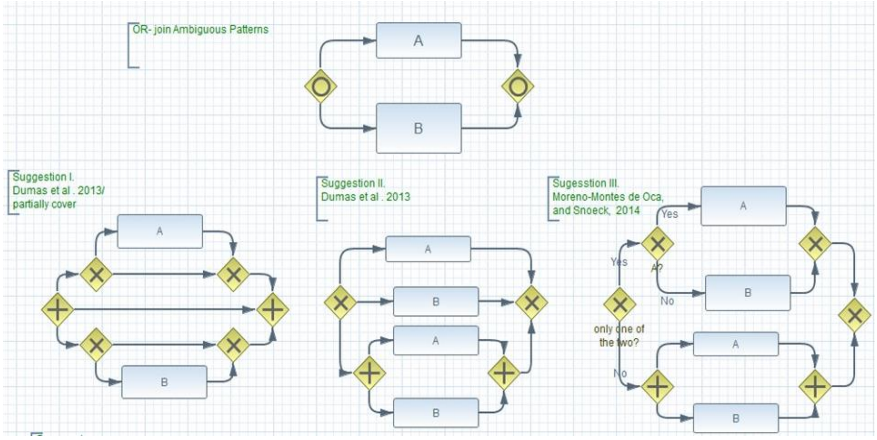

Fig. 7. The OR-join workaround solutions

In Fig. 7, the "suggestion I" represents the behavior of OR-join, while it also lets the situation of neither "A" nor "B". The behavior of OR-join can be also captured by using suggestions "II and III"; however, they use one task, such as "A", more than once in a BP model. Besides, the main problem of these two workarounds (i.e., II and III) is the scalability problem, in case that they are more than two tasks in the OR-region the number of paths and combinations grows exponentially.

7 Architecture of Disambiguation Support System (DSS)

To facilitate the process of disambiguation a Disambiguation Support System (DSS) is designed and partially developed. The DSS Architecture is illustrated in Fig. 8.

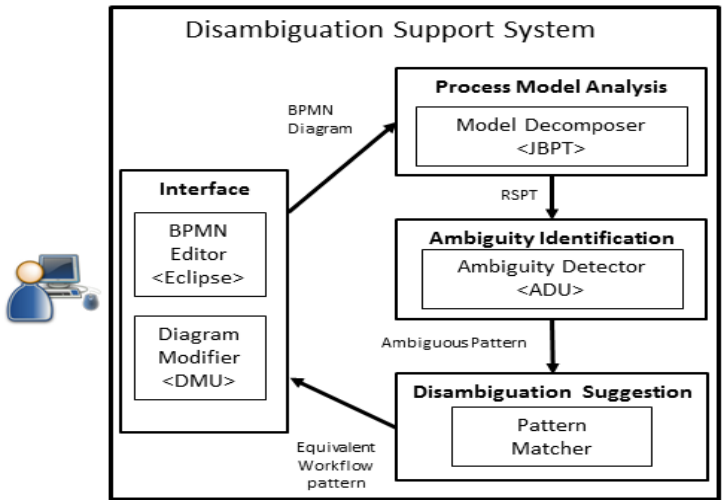

Fig. 8. Disambiguation Support System Architecture

The DSS units are as follows:

i) *Model Pre-processor.* This unit is responsible for designing and illustrating the BPMN Process Diagrams. To develop this unit, we have used Eclipse[4] BPMN 2.0 editor. Moreover, the syntactical correctness of developed BPMN models is checked in accordance with the Eclipse verification rules.

ii) *Model Decomposer*: The RPST builder creates a structured tree from *.bpmn files. Hence, we can recognize the types of fragments (i.e., bond or rigid). For this purpose, we have used jBPT[5] and BPT-NH[6], which are two Java libraries that apply graph structures to provide a hierarchy in form of RPST for BPMN process diagrams.

iii) *Ambiguity Identifier*: Following our proposed algorithm, we identify the OR-join ambiguous patterns. Accordingly, the ambiguous patterns their type are returned.

iv) *Disambiguation Recommender*: According to the identified ambiguous pattern, unambiguous solutions are recommended to modelers.

v) *Visual Representor*: This unit manages the user interactions and graphical interface, and it is also the subject of further development.

Fig. 9 depicts a screenshot of ambiguity identifier unit of DSS, which its source code is available on online[7]. The algorithm performs well on the identification of OR-join patterns based on the test of different examples of ambiguous patterns.

[4] Eclipse : www.eclipse.org/bpmn2-modeler
[5] Java Business Process Technology (JBPT): www. code.google.com/jbpt
[6] BPT-NH: www. github.com/BPT-NH
[7] BPCon : https://bitbucket.org/Hodjat/disambiguationsupportsystem

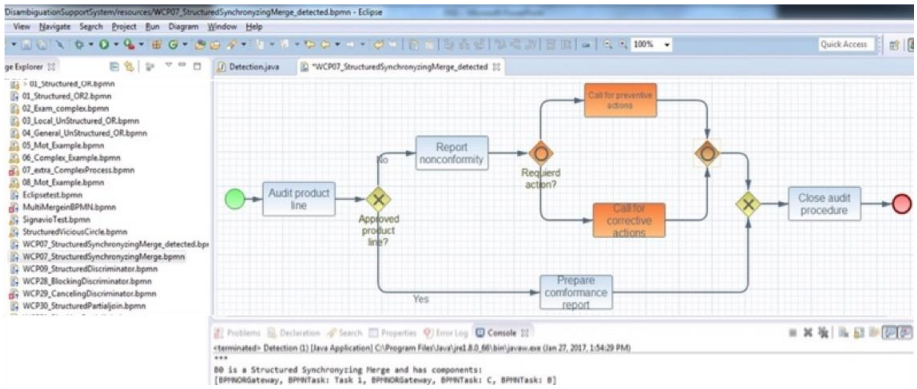

Fig. 9. A screenshot of the ambiguity identifier feature in DSS system

8 Discussion and Conclusion

In this paper, we tackled the problem of certain ambiguous and non-executable BPDs that include the OR-join constructs. Nevertheless, this construct is vastly used when combining and consolidating different BPs by independent organizations, such as those that are members of a CNO that wish to share their BP repositories to deliver a set of consolidated business services [16].

To provide our solution, the concept of workflow patterns is applied, in order to identify the ambiguous patterns systematically and suggest equivalent unambiguous BP fragments for substituting with them. To this end, a set of formalized definitions are provided for ambiguous patterns and the corresponding algorithm is designed, implemented, and verified. No similar BPMN disambiguation solution based on ambiguous workflow identification has been addressed to the best of our knowledge, while the identification of workflow patterns is partially studies in two research works.

The study introduced in [14] proposes definitions to recognize a number of workflow, however they have excluded the OR-join included workflow patterns. Another study by Elaasar et al. in [15] introduces a MOF-based modeling language for the purpose of identifying patterns in BPMN that has taken the workflow patterns as a test case. However, they have reported difficulties in identifying the aimed patterns (i.e., structured synchronizing merge), due to the sensitivity of the method to natural language notes on gateways and the lack of provided information of gateways (e.g., conditions to follow each path emanated from a gateway). In comparison, our approach not only provided disambiguation suggestions, but also is not limited by the information of the gateway variables of a BPD. As the future work, we will focus on Complex-join included patterns to identify and resolve.

References

1. OMG/BPMI: Business Process Model and Notation (BPMN) version 2.0.2. *OMG Specification, Object Management Group* (2014)
2. Malekan, H. S., Afsarmanesh, H.: Overview of business process modeling languages supporting enterprise collaboration. In International Symposium on Business Modeling and Software Design (pp. 24-45) Springer (2013)
3. Van der Aalst, W. M.: Business process management: a comprehensive survey. ISRN Software Engineering (2013)
4. Börger, E.: Approaches to modeling business processes: a critical analysis of BPMN, workflow patterns and YAWL. Software & Systems Modeling, 11(3) (2012)
5. Kossak, F., Illibauer, C., Geist, V., Kubovy, J., Natschläger, C., Ziebermayr, T., Kopetzky, T., Freudenthaler, B. and Schewe, K.D.: A Rigorous Semantics for BPMN 2.0 Process Diagrams. In *A Rigorous Semantics for BPMN 2.0 Process Diagrams* (pp. 29-152) Springer (2014)
6. van Der Aalst, W.M., Ter Hofstede, A.H., Kiepuszewski, B. and Barros, A.P.: Workflow patterns. Distributed and parallel databases 14.1. 5-51 (2003)
7. Russell, N., van der Aalst, W.M., ter Hofstede, A.H., Workflow patterns: the definitive guide MIT Press (2016)
8. Geiger, M., Harrer, S., Lenhard, J. and Wirtz, G.: BPMN 2.0: The state of support and implementation. Future Generation Computer Systems, 80, pp.250-262 (2018)
9. Wynn, M.T., Edmond, D., van der Aalst, W.M. and ter Hofstede, A.H.: Achieving a General, Formal and Decidable Approach to the OR-join in Workflow using Reset nets. Application and Theory of Petri Nets (pp. 423-443) Springer (2005)
10. Völzer H.: A new semantics for the inclusive converging gateway in safe processes. In BPM conference (pp. 294-309) Springer, Berlin, Heidelberg (2010).
11. Polyvyanyy, A., Vanhatalo, J., Völzer, H.: Simplified computation and generalization of the refined process structure tree. WSFM. (pp. 25-41) (2010)
12. de Oca, I. M. M., & Snoeck, M. Pragmatic guidelines for business process modeling. KU Leuven, Faculty of Economics and Business (2015)
13. Dumas, M., La Rosa, M., Mendling, J., & Reijers, H. A.: Fundamentals of business process management (Vol. 1, p. 2) Springer (2013)
14. Juan, YC, Yuan, KY.: Control flow pattern recognition for BPMN process models. International Journal of Electronic Business Management. 111(2):133 (2013)
15. Elaasar, M., Briand, L. C., Labiche, Y.: VPML: an approach to detect design patterns of MOF-based modeling languages. SoSyM, 14(2), 735-764 (2015)
16. Malekan, H.S., Adamiak, K., Afsarmanesh, H.: A systematic approach for business service consolidation in virtual organization. SOCA 12: 41 (2018)
17. Camarinha-Matos, L. M., Afsarmanesh, H., Oliveira, A. I., & Ferrada, F.: Cloud-based collaborative business services provision. In: Enterprise Information Systems, Lecture Notes on Business Information Processing, Volume 190, pp 366-384, Springer (2014)
18. Afsarmanesh, H., Sargolzaei, M., Shadi, M.: Semi-automated software service integration in virtual organisations. Enterprise Information Systems, 9(5-6), 528-555. (2015)
19. Camarinha-Matos, L. M., Afsarmanesh, H., Galeano, N., Molina, A.: Collaborative networked organizations– Concepts and practice in manufacturing enterprises. *Computers & Industrial Engineering*, 57(1), 46-60 (2009)
20. Malekan, H. S., Mehrizi, M. H. R., & Afsarmanesh, H.: Positioning collaboration in business process model consolidation in VOs. In Working Conference on Virtual Enterprises Springer. (pp. 110-123) (2016)