On Avoiding Erroneous Synchronization in BPMN Processes

Flavio Corradini, Fabrizio Fornari, Chiara Muzi(✉), Andrea Polini, Barbara Re, and Francesco Tiezzi

University of Camerino, 62032 Camerino, Italy
{flavio.corradini,fabrizio.fornari,chiara.muzi, andrea.polini,barbara.re,francesco.tiezzi}@unicam.it

Abstract. BPMN has acquired a clear predominance in the modeling of organization processes. Since it is a fairly complex modeling language, in some cases it is important to clarify the behavior of a modeled process, especially when concurrency comes into play. We consider unsafe process models with arbitrary topology, and we focus on the effects of concurrent control flows activated within single process instances. We use text annotations to clarify the concurrent behavior, and tokens with identity to regulate the synchronizations. We illustrate the benefits of our approach by a simple, yet realistic, scenario about paper reviewing.

Keywords: Modeling · Concurrency · BPMN · Erroneous synchronization.

1 Introduction

Concurrent issues in modeling and programming has been discussed for years [1]. With the growing number of distributed applications run by complex organizations, a proper management of concurrency became more and more important. Focusing on enterprise architecture Zachman identified the different dimensions to consider in order to reason on, and understand, the dynamics of a complex organization [2]. Among the others process modeling describes how an organization structures its activities in order to achieve its goals [3]. Concurrency results to be an issue with regards to the arrangement of these activities. Indeed, some of them "*can be performed simultaneously by several autonomous workers that may coordinate their work by means of communication*" [4]. Resolving concurrency issues positively impacts on the organization performance.

To describe a process, de facto standard is BPMN 2.0 [5] provided by OMG. It adopts a semi-formal approach combining a precisely specified syntax with a token-based semantics given in natural language. A semi-formal description is useful to allow different stakeholders to easily communicate and share ideas so that BPMN can play the role of a bridge between business analysts and IT developers [6].

© Springer International Publishing AG 2017
W. Abramowicz (Ed.): BIS 2017, LNBIP 288, pp. 106–119, 2017.
DOI: 10.1007/978-3-319-59336-4_8

Concerning the management of concurrency in BPMN, it is worth noticing that multiple process instances is explicitly addressed in the specification, while the effects of concurrent control flows within a single instance is underspecified. This can easily occur due to concurrent control flows initiated through the use of AND-split and OR-split gateways. The relevance of such an issue is pointed out also by studies stating that an increase in the level of concurrency for BPMN models implies an increase in modeling error probability [7,8]. Here, we focus on the management of concurrent behavior in a single process instance.

Imposing well-structured rules contributes to control and minimize the level of this form of concurrency by guaranteeing some correctness properties [9,10]. However, such restrictions are not easily applicable by all model designers. Indeed, on the one hand, designers with limited modeling experiences are prone to model spaghetti processes. On the other hand, more expert designers should be free to express their creativity in modeling the process according to the reality they feel [4]. In addition, not all process models with an arbitrary topology can be transformed into equivalent well-structured one [11,12]. Summing up, advantages of the structured process modeling style over the unstructured one (and vice versa) have been a topic of active debates for decades [4]. Hence, processes with arbitrary topology are still very common in practice.

In this work we do not impose any restriction on the usage of the modeling notation. We refer here to process models with an arbitrary topology including concurrent behavior, which may lead to the occurrence of erroneous synchronizations. Such a kind of processes generally include sequence flows that can be activated more than once at the same time, referred as *unsafe* processes. These processes are typically discarded by the modeling approaches proposed in the literature, as they are over suspected of carrying bugs. Unfortunately, this attitude significantly limits the use of concurrency in business process modeling, which is an important feature in modern systems and organizations. Instead, we believe that in these cases the designer could keep the 'offending' model and solve the issue by better clarifying the intended behavior. In fact, the problem typically is not in the model itself but it is due to the underspecification of the BPMN standard in dealing with concurrency issues within a single process instance.

Our work is thus mainly motivated by the need of achieving synchronization correctness in unstructured processes, which is still an open challenge. More specifically, the contribution of the paper is an advanced use of BPMN text annotations to enrich the model with information suitable to deal with concurrent execution of control flows. We also contribute by refining the process execution semantics by taking into account token identities to avoid erroneous synchronizations. The major benefit of our contribution is having the possibility to fully explore the modeling potentialities of BPMN notation in case of processes with arbitrary topology and concurrent behavior.

The rest of the paper is organized as follows. Section 2 introduces a motivating scenario, while Sect. 3 discusses on unsafe processes. Section 4 provides details on our methodology, and Sect. 5 reports the works found in literature that inspired our work. Finally, Sect. 6 closes the paper with some conclusions and future work.

2 A Motivating Scenario

To better clarify the issues we want to address, we introduce a scenario concerning the management of the paper reviewing process of a scientific conference. We use this scenario to motivate our approach, and throughout the paper to illustrate its technicalities. We rely therefore on a simplified version of the scenario, as in [3, Sect. 4.7.2].

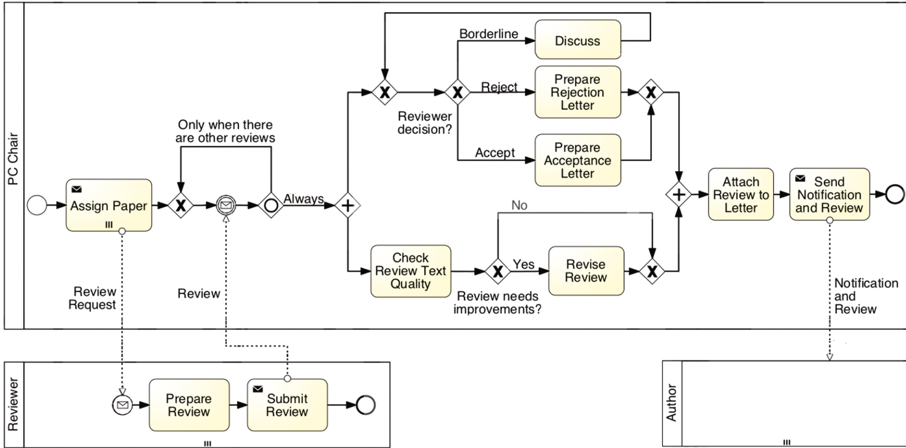

Fig. 1. Paper review process collaboration.

This is modeled in BPMN as the collaboration in Fig. 1. The participants are: **Program Committee (PC) Chair**, the organizer of the reviewing activities. For the sake of presentation, we assume that the considered conference has only one chair, whose behavior is represented by the process within the PC Chair pool; **Reviewer**, a person with knowledge in some of the conference topics. This role is modeled as a multi-instance pool. Each process instance describes the tasks that a reviewer has to accomplish to complete her/his assignment. For the sake of simplicity, we choose to assign only one paper to each reviewer; **Author**, who submitted a paper to the conference and acts on behalf of the other authors (contact author). This role is modeled as a multi-instance blackbox pool since details on the author behavior are not relevant to our purposes.

The reviewing process is started by the chair, who assigns (via a parallel multi-instance activity) each submitted paper to a reviewer. Then, the chair receives the reviews and evaluates them. In particular, as soon as a review is received, the chair starts its evaluation and is immediately ready to receive and process another review. This behavior is rendered in BPMN by means of a loop, realized via an OR split gateway and a XOR join gateway, whose single iteration consists of receiving a paper review and starting its evaluation. The evaluation of each review is modeled by the process fragment enclosed by

the AND split gateway and the AND join one. Indeed, the evaluation proceeds along two concurrent control flows: *(bottom branch)* the chair checks the quality of the received review and, if necessary, he/she revises it to improve and *(top branch)*, according to the reviewer decision, the chair prepares the acceptance/rejection letter or, if the paper requires further discussion, the decision is postponed. In the last part of the process (after the AND join gateway), the chair attaches the review to the notification letter, and sends it to the contact author.

The model described so far represents in an intuitive and compact way the paper reviewing scenario. However, despite its simplicity, it hides some subtleties that may affect its correct enactment. For instance, it may happen that an author of a paper will receive a notification with attached the review of another paper. We describe below how this kind of situations may occur, by making use of the concept of *token*, thought of as a means to indicate the process elements that are active during the execution.

Let us consider the reception of the review for a paper, say $paper1$. This event produces a token that activates the OR split; assuming that other reviews are waited, the OR gateway produces in its own turn two tokens: one is used to reactivate the receiving message intermediate event, while the other to activate the evaluation of $paper1$'s review. This latter token is split into two tokens for activating the two evaluation branches described above. Then, a review for another paper, say $paper2$, is received and dealt with in a similar way. The evaluations of the two reviews proceed, hence, along two concurrent control flows. After some steps, we may have the current situation: *(i)* $paper1$'s review has been revised by the chair and a corresponding token reached the AND join gateway from the bottom incoming flow, while the other $paper1$ token is still marking the Discuss task, as the paper received a borderline score; *(ii)* $paper2$ received a reject score, thus, while the chair is still checking the review quality, a $paper2$ token reached the AND join gateway from the top incoming flow.

In this situation, the two incoming flows of the AND join carry a token. Thus, according to the standard semantics of BPMN, the AND gateway triggers the flow through its outgoing sequence flow. In fact, the semantics does not distinguish tokens related to the evaluation of the $paper1$'s review from those related to the $paper2$'s one. This **erroneous synchronization** of tokens allows the process execution to continue with the notification task, using the revised review of $paper1$ and the rejection letter of $paper2$.

To address this problem, we advocate the use of tokens with identity. This enables the AND join gateway to distinguish the two incoming flows, hence avoiding the erroneous synchronization. In fact, only tokens with the same identity, i.e. referring to the same paper review, synchronize. When synchronization cannot take place, the incoming tokens just wait for the arrival of 'brother' tokens.

Notably, in order to have the situation described above, during the execution of the considered process more than one token must concurrently transit along the same sequence flow. In the reviewing scenario this happens each time a

review is assigned as result of the *OR Join* behavior specification. Moreover, the other condition leading to situations of erroneous synchronization is the presence of *concurrent control flows*, where the generated multiple tokens are split and then have to be synchronized. In our scenario, we have that the concurrent control flows correspond to the two evaluation activities performed by the chair. We present in Sect. 4 our approach to avoid erroneous synchronizations to take place when the above conditions are met.

Other possible solutions have been proposed in order to overcome the concurrency issues addressed by our approach. With reference to the introduced reviewing scenario, a first solution proposes to model the PC Chair by means of two processes: one that assigns papers, collects reviews and instantiates the other (multi-instance) process, whose instances separately deal with the evaluation of paper reviews. As no interaction can take place among these instances, erroneous synchronizations cannot emerge. A second solution suggests to put in sequence the various evaluation activities performed by the chair. This avoids concurrent flows and, hence, the possibility of erroneous synchronizations. Compared with our solution, where the chair behavior is modeled as a single process instance, the first alternative does not fit well with the reality, as the behavior of a single human person is split into two separate processes, one of which is multi-instance. Missing to represent concurrency aspects can be dangerous when the model is intended to model activities to be automated by information systems. The second one, instead, imposes to put in sequence a set of activities that originally were parallel. Most of all, the two alternative solutions require an alteration of the original structure, as well as of the semantics, of the designed process. This requires the designer to be expert enough to identify the concurrency issue in his model and, then, to solve it by properly restructuring the model. Moreover, these are ad-hoc application-specific solutions. Instead, our approach provides a general solution to the problem, without altering the structure of the process. In fact, we acts on the level of abstraction, which is lowered in order to distinguish token identities.

3 On Unsafe Processes

Unsafe processes emerge only when the control flow is organized in such a way that tokens can be dynamically generated during the process instance execution. In this section, we clarify how multiple tokens are generated, and how to recognize processes that do that and hence may be subject to erroneous synchronization problems.

First, we set the scene by introducing the necessary background notions. The first key concept is that of *token*. The BPMN specification states that "a token is a theoretical concept that is used as an aid to define the behavior of a process that is being performed" [5, Sect. 7.1.1]. A token is commonly generated by a start event, traverses the sequence flows of the process and passes through its elements (enabling in this way their execution), and is consumed by an end event when the execution terminates. Besides, tokens can be generated and consumed

by gateways. The distribution of tokens in the process elements, called *marking*, defines a state of the process, as it indicates which activities are enabled and which sequence flows have been selected. The *process execution* is therefore defined in terms of marking evolution (i.e., changes of state).

Now, by relying on the above notions, we define when a process is *unsafe*.

Definition 1 (Unsafeness). *A process is* unsafe *if and only if during its execution it can reach a marking where more than one token marks the same sequence flow.*

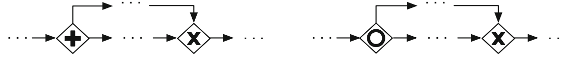

Fig. 2. Token generator structures (bounded number of tokens).

Fig. 3. Token generator structures (unbounded number of tokens).

Intuitively, a process is unsafe if it contains either a process fragment capable of generating a bounded number of tokens (Fig. 2), or an unbounded number of tokens, by resorting to a loop (Fig. 3). Notably, it is evident that *well-structured* processes do not contain token generator fragments [10]. Looking at the structure of the process in Fig. 1 referring our running example, we can identify a pair of gateways, namely the first XOR and the OR, that form a fragment corresponding to the structure in Fig. 3. To establish if a process is unsafe, we can translate the BPMN model into a Petri Net [10] and resort to techniques for verifying safeness properties of Petri Nets. For an account of these techniques we refer to [13,14].

4 An Approach to Erroneous Synchronizations Avoidance

To manage unsafe BPMN models, we need a fine-grained view on the tokens flow within process instances. This allows us to distinguish tokens referring to different concurrent control flows; e.g., in our motivating scenario we want to distinguish the tokens referring to concurrent evaluations of different papers. We achieve this by relying on the use of *tokens with identity*. Such identity can evolve during the process execution, as the token can have different meanings in different parts of a process. Thus, the token can be identified by means of different (unique) identifiers, whose scope can be limited to the part of interest in the process. Such scope is application specific, hence it must be the designer in charge of explicitly specifying this information on the process model.

The proposed approach includes ingredients allowing to: (i) enrich BPMN models with additional information, via specific text annotations on sequence

flows, called *check-in* and *check-out*, that enclose the part of process defining the scope of a token identifier; and (ii) refine the BPMN semantics by taking into account token identities to control synchronizations. In particular, such ingredients extend the modeling phase of a BPMN process with the following steps: **Step 1** - The designer controls if the designed process is unsafe (see Sect. 3). If the process is unsafe, the designer goes to step 2, otherwise the model can be safely implemented; **Step 2** - The designer introduces check-in and check-out annotations in the process; **Step 3** - The designer analyzes the process execution by means of a refined semantics that takes into account token identities, thus avoiding erroneous synchronizations between distinguished tokens. If the desired behavior is achieved, the model can be safely implemented; otherwise the designer either goes back to step 2 to revise the positions of check-in and check-out annotations, or redesigns the process model and reapplies the approach from step 1.

Our choice of using text annotations for defining the scopes of token identifiers, rather than introducing new modeling elements, is due to the intention of avoiding a syntactic extension of the BPMN notation. This allows us to easily apply our approach to existing BPMN models and, most of all, to use the whole plethora of tools already available for BPMN. These are indeed the usual benefits of approaches based on annotations, which nowadays are very common in the field of programming languages.

In the following, we describe the proposed use of annotations and how they permit refining the BPMN semantics. Then, we show how the approach works into practice.

4.1 Check-in and Check-out

Check-in and check-out annotations are used to explicitly specify the scope of token identifiers. In particular, **Check-in** represents a point of the process from where the identity of the traversing tokens is enriched with a fresh identifier; **Check-out** represents a point of the process where the identifiers created by the corresponding check-in are no longer needed and, hence, are removed from the identity of the traversing tokens.

Check-ins and check-outs are identified by their names, ranged over by n. Each check-out must be correlated with one check-in, i.e. there is a check-in in the process model with the same name; on the other hand, each check-in is correlated with zero or more check-outs. Graphically, see Fig. 4, a check-in (resp. check-out) is a standard BPMN text annotation with the peculiarity of being attached to a sequence flow and of enclosing a text of the form Check-in (n) (resp. Check-out (n)).

Fig. 4. Graphical notation of check-in and check-out annotations.

As already mentioned, a token can have different meanings in the process. This can be achieved by means of more check-ins. Notably, a check-in can occur inside a check-in/check-out block. Therefore, the identity of a token is defined as a set T of pairs of the form (n, id), where n is the name of a check-in traversed by the token and id is an identifier freshly[1] generated by the check-in n. When a token is generated by the activation of a start event, it is initialized with a default identity represented by the set $\{(init, 0)\}$, where $init$ is a reserved check-in name and 0 is an identifier. The identity of a token changes only when it traverses check-in or check-out points, while its flow during the process execution is regulated by the standard BPMN execution semantics unless when it meets a synchronization point (i.e., an AND or an OR join gateway). We explain below the resulting refined BPMN semantics.

When a token traverses a check-in point n, its identity is not altered if it already contains an identifier generated by n, otherwise the token identity is enriched with a new identifier pair. Formally, the token identity evolution determined by a check-in is defined by function *TraverseCheckIn* that, given as input a check-in name and the identity set of an incoming token, it returns as output the identity of the outgoing token

$$TraverseCheckIn(n, T) = \begin{cases} T & \text{if } (n, id) \in T \\ T \cup \{(n, fresh(n))\} & \text{otherwise} \end{cases}$$

where $fresh(n)$ is a function that returns a fresh identifier for the check-in n (notably, this function can be straightforwardly implemented by relying on a counter local to each check-in). As an example, consider that during the execution of a process a token with identity $T_0 = \{(init, 0)\}$ passes through the check-in named *first*. The token identity set evolves to $T_1 = \{(init, 0), (first, 3)\}$, assuming that at the time of the check-in crossing $fresh(first)$ returns 3. Then, the token identity does not change until another check-in is reached. In particular, if the token then passes through the check-in *second*, the identity set becomes $T_2 = \{(init, 0), (first, 3), (second, 7)\}$, as $fresh(second)$ returns 7. If the token passes through the same check-in more than once, nothing happens if the identifier produced by such check-in is still considered in the identity set. In the example, if the token passes again through check-in *first*, its identity set remains T_2.

Dually, when a token traverses a check-out point n, its identity is not altered if it does not contain an identifier generated by n, otherwise the corresponding identifier pair is removed from the identity set of the token. Formally, the token identity evolution determined by a check-out is defined by the following function.

$$TraverseCheckOut(n, T) = \begin{cases} T \backslash (n, id) & \text{if } (n, id) \in T \\ T & \text{otherwise} \end{cases}$$

[1] An identifier, generated by a check-in n, is called *fresh* if it is different from all other identifiers previously generated by the check-in n.

Let us consider again the example previously discussed, where the token currently has identity T_2. Now, if during the execution of the process the check-out *second* is reached, then the identity set T_2 changes into T_1. Instead, if the token reaches a check-out named *first*, then the identity set T_2 becomes $T_3 = \{(init, 0), (second, 7)\}$, while nothing happens if the token reaches a check-out named *third*.

Finally, as already said, when a token with identity traverses any element of the process model different from a check-in, a check-out or a synchronization point, the effect on the token and on the process execution is the one prescribed by the standard semantics of BPMN. For example, if a token with identity set T_1 traverses an AND split gateway, a token with the same identity T_1 is produced for each outgoing sequence flow. Instead, when a token with identity traverses a synchronization point, the BPMN semantics synchronizes tokens with the same identity, i.e., tokens whose identity sets coincide. In this way, erroneous synchronizations, which mix up different concurrent control flows, are avoided. Notably, the synchronization requires a complete match of identities among tokens, which means that the identity sets must have the same pairs; thus, for example, $\{(init, 0), (third, 3)\}$ and $\{(init, 0), (first, 5)\}$ do not match with T_1. It is also worth noticing that, in case of synchronization of tokens whose identity is given by the default value $\{(init, 0)\}$, our refined semantics coincides with the one prescribed by the BPMN standard. In other words, our semantics is conservative with respect to the standard one, i.e., if no check-in and check-out annotations are introduced in the model then the two semantics coincide.

To sum up, once the BPMN model under design is enriched with check-ins and check-outs, during its execution we can observe the evolution of token identities. In this way we are able to track the behavior of the process considering the paths traversed by the tokens and, most of all, their synchronizations (ensured to be non-erroneous).

4.2 The Approach at Work

In this section we illustrate how our approach can be applied in practice. Figure 5 shows how check-ins and check-outs are used to specify in which part of the process, within the PC Chair pool, tokens represent the control flows of the paper evaluation. For the sake of presentation, we identified three relevant parts of the process named A, B and C. Moreover, to show the flow of each token, we mark the corresponding path in the process with token identities (curly brackets and the default identifier are omitted).

At the beginning of the execution, the token placed on the start event has the default identity, represented by the set $\{(init, 0)\}$. Then, the token enters into, and hence activates, Part A of the process, which is a token generator. Thus, for each received review, a new token identity has to be generated. To this aim, the designer introduced a check-in named n so that, as soon as a token traverses the check-in, its identity is enriched with the new identifier $(n, 1)$, denoting that the token is related to review of *paper*1. The OR split gateway then splits the token

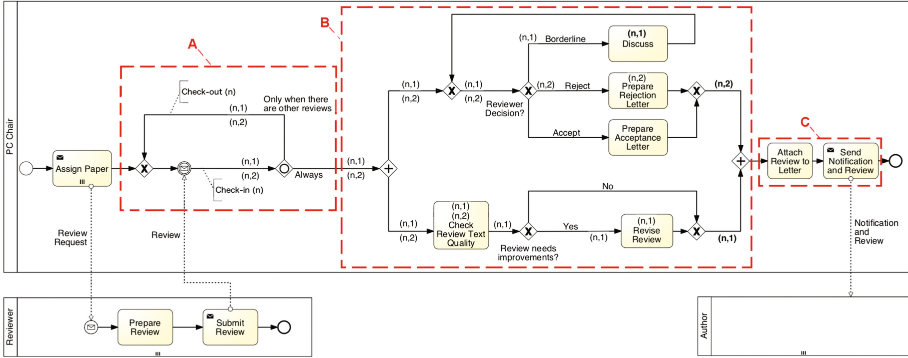

Fig. 5. Paper reviewing process collaboration simulation with tokens id.

into two tokens with the same identity $\{(init, 0), (n, 1)\}$. One of the generated tokens will go back into the loop, traversing the check-out point n and hence loosing its $paper1$ identity. This shows the usefulness of the check-out annotation in our approach: sometimes it is necessary that a token looses its identity as, e.g., it enters in a path where it is merely used as a control flow signal. In our case, the only purpose of the considered token is to activate a new iteration of the loop; in fact, without loosing its identity the token would fail in doing this. Instead, the other $paper1$ token will go into Part B of the process. This token will cross the AND split gateway and the evaluation of the $paper1$'s review will start. From this point, the execution proceeds as described in Sect. 2, thus a new token with identity $\{(init, 0), (n, 2)\}$ enters in the game, and the marking represented by the tokens whose identity is written in bold in Fig. 5 is reached. Now, the AND join gateway has two incoming tokens, one per each incoming edge, and thus evaluates their synchronization. Anyway, according to the refined semantics, the synchronization does not take place, as the two incoming tokens have different identities. Therefore, they remain in the edges waiting for their brothers. In this way, the appropriate synchronization will take place, the tokens will go through Part C of the process, and a notification to the author with attached the review of the corresponding paper will be sent.

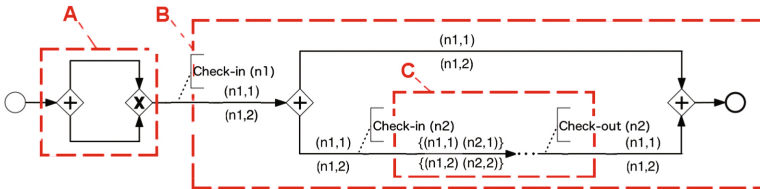

Fig. 6. Process structure combining check-ins and check-outs.

Now, in order to explain the use of more than one check-in and check-out in a process, we show also how the approach applies to another example. For the sake of readability, we show just the structure of the process, i.e. we omit its task elements. The process structure in Fig. 6 is divided in three parts: A is a token generator that produces a token for each of the outgoing edges of the AND split; B corresponds to a token identity scope[2]; and C is a sub-part of B where tokens identity must be further specialized. The figure shows the flow of tokens in the structure, and in particular how token identities evolve in case of nested scopes. Let us consider now a variant of this process, shown in Fig. 7. In this case the structure of the process is enriched with a path from the inner scope to an element of the enclosing one. In particular, this path is then merged with a path of the enclosing scope (AND join gateway in Part A). In order to allow the synchronization of tokens coming from these two paths it is necessary to remove the identifiers created by check-in $n2$. This is properly done via a second check-out $n2$. This example thus shows why we may need to associate two or more check-outs to a single check-in.

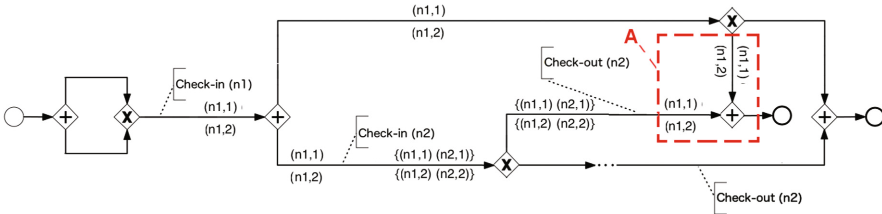

Fig. 7. A variant of the process structure in Fig. 6.

5 Related Works

Several techniques have been developed and applied to specify and reason on issues introduced by concurrency in software systems [15,16]. Concurrency is recognized as an important aspect of processes [4], in particular when processes have to be simulated and/or executed [17]. With reference to processes, three different kinds of concurrency have been highlighted [18]: concurrent processes, concurrent control flows inside a single process, and concurrent events/tasks.

The issues observed in managing concurrent control flows and related synchronizations have been already considered by some workflow patterns [19]. Among the others, the "And-join generalization" pattern corresponds to the general notion of AND-join where several execution paths are synchronized and merged together. The pattern supports situations, such as those non-safe, where one or more incoming branches may receive multiple tokens for the same process instance. The intended semantics for the pattern tends to be unclear in situations involving non-safe behavior. Our paper aims to contribute to close this

[2] Notably, no check-out is defined for $n1$, meaning that identifiers of the form $(n1, id)$ must be keep on token identities until the end of the execution.

gap. In the BPMN specification we can observe a similar issue. The BPMN standard uses the concept of token to facilitate the discussion about a process execution flow, however it does not impose conditions on how to keep track of tokens propagation.

Tokens with identity have been used in other research works. Nevertheless, they are mainly used to manage concurrent processes rather than, as we propose, to manage concurrent control flows. For instance Börger et al. discuss about the use of tokens identity represented as hierarchy sets for tracing the sequence flow of a process instance [20]. The notion of token identity has been also discussed with reference to the characterization of the OR-join behavior [21]. Also Colored Petri Nets, where tokens have identity (colors), have been used to represent concurrent processes [22]. Finally, token identifiers have been also used to control process execution in the interaction with database transactions enabled by the represented process [23].

Multiple instance management raises problems when passing from design time to run-time [24]. The problem of the run-time synchronization evaluation is also introduced with regard to the OR-join by Dumas et al. [25]. In this regard we believe that postponing the issues from the design to the implementation is not a general solution. On one side because it is well known the importance of early defect detection to avoid loss of time and money. On the other side because the implementation of BPMN processes needs a transformation to executable languages that can introduce further issues (i.e. those introduced by BPEL [26–28]).

6 Conclusions and Future Work

In this paper we presented an approach to solve issues caused by the inherent underspecification of synchronizations statements in BPMN models, and that can emerge when unsafe processes with an arbitrary topology and concurrent control flows are considered. To solve the issue we rely on the introduction of text annotations, which allows the model designer to clarify the intended behavior in terms of tokens with identity.

As a future work, we plan to investigate on possible strategies to automatize the placement of check-in and check-out annotations, which would help us to resolve issues regarding practical usage and scalability. Currently, this step is completely manual and requires some efforts from the model designer, who has to carefully arrange the annotations in the BPMN model. This could also help to evaluate our approach and make the proposed BPMN extension easy-to-use, useful and less prone to errors. Moreover, we plan to extend our BPMN formalisation in [29] with the check-in and check-out notion. Finally, we plan to develop a software tool exploiting the potentialities of the approach to automatically generate code that is free from synchronization issues from (annotated) BPMN models. This will also enable a systematic validation of the proposal.

References

1. Cleaveland, R., Smolka, S.A.: Strategic directions in concurrency research. ACM Comput. Surv. **28**(4), 607–625 (1996)
2. Zachman, J.A.: A framework for information systems architecture. IBM Syst. J. **26**(3), 276–292 (1987)
3. Weske, M.: Business Process Management: Concepts, Languages, Architectures. Springer (2007)
4. Polyvyanyy, A., Bussler, C.: The structured phase of concurrency. In: Bubenko, J., Krogstie, J., Pastor, O., Pernici, B., Rolland, C., Sølvberg, A. (eds.) Seminal Contributions to Information Systems Engineering, pp. 257–263. Springer, Heidelberg (2013)
5. OMG: Business Process Model and Notation (BPMN V 2.0). Technical report (2011)
6. Henderson, J.C., Venkatraman, N.: Strategic alignment: leveraging information technology for transforming organizations. IBM Syst. J. **32**(1), 4–16 (1993)
7. Mendling, J., Sanchez-Gonzalez, L., Garcia, F., La Rosa, M.: Thresholds for error probability measures of business process models. J. Syst. Softw. **85**(5), 1188–1197 (2012)
8. Moreno-Montes de Oca, I., Snoeck, M.: Pragmatic guidelines for business process modeling. Technical Report 2592983, KU Leuven, November 2014
9. Mendling, J., Reijers, H.A., van der Aalst, W.M.: Seven process modeling guidelines (7PMG). Inf. Softw. Technol. **52**(2), 127–136 (2010)
10. Dijkman, R.M., Dumas, M., Ouyang, C.: Semantics and analysis of business process models in BPMN. Inf. Softw. Technol. **50**(12), 1281–1294 (2008)
11. Polyvyanyy, A., García-Bañuelos, L., Dumas, M.: Structuring acyclic process models. Inf. Syst. **37**(6), 518–538 (2012)
12. Polyvyanyy, A., Garcia-Banuelos, L., Fahland, D., Weske, M.: Maximal structuring of acyclic process models. Comput. J. **57**(1), 12–35 (2014)
13. Murata, T.: Petri nets: properties, analysis and applications. Proc. IEEE **77**(4), 541–580 (1989)
14. Van Der Aalst, W.M.P.: Workflow verification: finding control-flow errors using petri-net-based techniques. In: Aalst, W., Desel, J., Oberweis, A. (eds.) Business Process Management. LNCS, vol. 1806, pp. 161–183. Springer, Heidelberg (2000). doi:10.1007/3-540-45594-9_11
15. Ramchandani, C.: Analysis of asynchronous concurrent systems by timed petri nets. Massachusetts Institute of Technology, Cambridge (1974)
16. Clarke, E.M., Emerson, E.A., Sistla, A.P.: Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans. Program. Lang. Syst. **8**(2), 244–263 (1986)
17. Vasilecas, O., Smaižys, A., Rima, A.: Business process modelling and simulation: hybrid method for concurrency aspect modelling. J. Mod. Comput. **1**(3–4), 228–243 (2013)
18. Sörensen, O.: Semantics of Joins in cyclic BPMN Workflows. Ph.D. thesis, Christian-Albrechts-University Kiel, Department of Computer Science (2009)
19. Russell, N., Ter Hofstede, A.H., Mulyar, N.: Workflow controlflow patterns: a revised view. Technical Report BPM-06-22, BPMcenter.org (2006)
20. Börger, E., Thalheim, B.: A method for verifiable and validatable business process modeling. In: Börger, E., Cisternino, A. (eds.) Advances in Software Engineering. LNCS, vol. 5316, pp. 59–115. Springer, Heidelberg (2008). doi:10.1007/978-3-540-89762-0_3

21. Thalheim, B., Sorensen, O., Borger, E.: On defining the behavior of OR-joins in business process models. J. UCS **15**(1), 3–32 (2009)
22. van Hee, K.M., Sidorova, N., van der Werf, J.M.: Business process modeling using petri nets. Trans. Petri Nets Other Models Concurrency VII, 116–161. Springer (2013)
23. Van Hee, K.M., Sidorova, N., Voorhoeve, M., others: Generation of database transactions with petri nets. Fundamenta Informaticae **93**(1–3), 171–184 (2009)
24. Barros, A.P., Grosskopf, A.: Multiple instance management for workflow process models. Google Patents US Patent 8,424,011, April 2013
25. Dumas, M.G., Grosskopf, A., Hettel, T., Wynn, M.T.: Evaluation of synchronization gateways in process models. Google Patents US Patent 8,418,178, April 2013
26. Recker, J.C., Mendling, J.: On the translation between BPMN and BPEL: conceptual mismatch between process modeling languages. In: CAISE, pp. 521–532 (2006)
27. Weidlich, M., Decker, G., Großkopf, A., Weske, M.: BPEL to BPMN: the myth of a straight-forward mapping. In: Meersman, R., Tari, Z. (eds.) OTM 2008. LNCS, vol. 5331, pp. 265–282. Springer, Heidelberg (2008). doi:10.1007/978-3-540-88871-0_19
28. Lapadula, A., Pugliese, R., Tiezzi, F.: Using formal methods to develop WS-BPEL applications. Sci. Comput. Program. **77**(3), 189–213 (2012)
29. Corradini, F., Polini, A., Re, B., Tiezzi, F.: An operational semantics of BPMN collaboration. In: Braga, C., Ölveczky, P.C. (eds.) FACS 2015. LNCS, vol. 9539, pp. 161–180. Springer, Cham (2016). doi:10.1007/978-3-319-28934-2_9