Communication Papers of the 18th Conference on Computer Science and Intelligence Systems pp. 65–74
DOI: 10.15439/2023F8794
ISSN 2300-5963 ACSIS, Vol. 37

Formal verification of BPMN diagrams in Integrated Model of Distributed Systems (IMDS)

Jakub Jałowiec
Institute of Computer Science,
Warsaw University of Technology
Nowowiejska Str. 15/19, 00-665 Warsaw, Poland
Email: kuba.jal@gmail.com

Wiktor B. Daszczuk
Institute of Computer Science,
Warsaw University of Technology
Nowowiejska Str. 15/19, 00-665 Warsaw, Poland
Email: wiktor.daszczuk@pw.edu.pl

Abstract— **Business process model and notation (BPMN) is a way of describing business processes using convenient diagrams. In the last decade, it became a de-facto industry standard, widely used by software architects and business analysts to describe business requirements and the overall structure of a designed information system. Ensuring that diagrams model their intended behavior is of utmost importance for notation users. This article deals with the definition of BPMN through the conversion to the Integrated Model of Distributed Systems (IMDS) and automated verification of BPMN diagrams. The translation of a subset of BPMN preserves information about the processes in the formal model. This allows finding partial deadlocks and checking partial termination (concerning a subset of processes), verification in terms of BPMN processes, and mapping found errors onto source BPMN definition. Moreover, IMDS is tailored to model distributed systems, which is the very nature of business processes. A tool for automated translation of BPMN diagrams to IMDS, automated verification, and visualization of results is developed.**

I. Introduction

An EXAMPLE of business processes in everyday life is ordering a product in an online shop. Appropriate steps of such a business process include several steps, like filling up the online form (ordering the product), preparation of money transfer form, payment, shipping the order, delivery and final confirmation.

Business processes involve several steps and should preserve logical relationships, be efficient, reliable, and coordinate work between stakeholders. Business processes have theoretical foundations in workflow nets [1], a class of Petri nets [2] used to model business behavior and formally analyzed mathematically.

Business Process Model and Notation (BPMN [3]), is a graphical framework used to model business processes with around 100 symbols representing various aspects of process execution, communication, and dataflow. It has found commercial use with numerous supporting tools. Proper modeling is important to save time and money during implementation and maintenance. The verification using model checking and temporal logic [4] ensures that a BPMN diagram follows its intended behavior.

BPMN's rich syntax presents challenges in formalizing its semantics [5], making it difficult to verify properties of a given model. This article presents a method for verifying BPMN diagrams using the Integrated Model of Distributed Systems (IMDS [6]), which is a formalism for describing and verifying distributed computational systems. IMDS is capable of detecting partial deadlocks, which involve only a subset of processes. The proposed method maps BPMN into IMDS, giving the diagrams formal semantics, and allowing for partial (or total) deadlocks detection. The Dedan tool [7] is used to automatically detect deadlocks and check the termination of the entire model system or set of processes.

The contributions of this article are:

1. Normalization of BPMN Process and Collaboration diagrams to an intermediate representation. It provides explicit semantics for all elements and helps to avoid ambiguity in interpretation.
2. Translation rules of BPMN to IMDS. All implemented BPMN elements have been reduced to 9 constructions for which implementation in IMDS was given. These constructs can be thought of as an "intermediate language" defining the semantics of BPMN elements through their corresponding IMDS structures.
3. Verification in IMDS for the location of deadlocks (total and partial) or checking the termination (total and partial). Checking of partial properties, not present in other tools, allows the designer to find errors in diagrams with their local cooperation, and even individual diagrams. Moreover, our verification tool Dedan uses a fair verification algorithm that prevents discovering false deadlocks [8].
4. Mapping verification traces back to BPMN specification to observe errors in the original specification rather than in the verification model. The verification methods described in the literature give the result of the check in a form specific to them, leaving the mapping of the resulting trace on the source BPMN diagram for a human. It is possible because the semantics of BPMN and its IMDS translation is the same.

©2023, PTI

Thematic track: Information Systems Management

5. Animation of counterexamples/witnesses traces on BPMN diagrams to observe the behavior of model components leading to a deadlock or successful termination/lack of termination.

There are many publications about verifying BPMN using different kinds of formalisms. However, few present a working solution that can be used to verify real-life use cases of BPMN diagrams in an automated way and view the verification results on the source BPMN diagram. However, the most important is automatic checking of partial deadlocks, which is rare in BPMN verification. A partial deadlock can occur even in a single process or in two communicating processes, while other processes perform their work.

II. Related work

Multiple techniques of BPMN formalization have been proposed. A common way of doing it is to map BPMN into Petri nets. Dijkman et al. [9] propose a mapping into classic Petri nets. It deals with a concise subset of BPMN, containing BPMN elements with local semantics, including Pools, Sub-processes, Exclusive Gateways, Parallel Gateways, Event-Based Gateways, Tasks, Events, and exception handling, whereas Tasks with only a single outgoing or incoming Message Flow are considered. There is also presented a tool to transform BPMN diagrams into a Petri Net Markup Language specification [10] that Petri net verifier can further process. Although the article deals with translation only of BPMN 1.x diagrams, it gives a general idea of BPMN 2.0.x translation.

Rachdi [11] proposes mapping into Time Petri Nets (TPN). The considered subset is the same as in [9], but the mapping introduces time constraints on executing BPMN elements. The authors propose an algorithm for the reachability analysis of the resulting TPN but do not provide an automatic tool. Other approaches to model time in BPMN are described in [12].

Authors of [13] propose mapping BPMN into Colored Petri Nets (CPN). Additionally, they introduce a method to divide a given BPMN model into partitions and verify them hierarchically, which reduces the complexity of the resulting CPN model. Unfortunately, the method works only for BPMN model that is well-defined, that is, only for a relatively small subset of BPMN, which limits its expressiveness. Additionally, the authors have implemented a tool that automatically translates BPMN diagrams into the corresponding CPN.

Li and Die [14] propose another method of mapping BPMN into classic Petri Nets. The method is rather poorly described and can be applied to only a small subset of BPMN, but it introduces the concept of preprocessing of BPMN diagrams. A similar concept, called normalization, is proposed in this article. Other attempts to formalize BPMN include for example transformation of BPMN into Pi-Calculus [15], PROMELA [16], COWS [17], Alvis [18] and YAWL [19] with the formalization through Graph Transformation Systems [20].

Automated BPMN diagrams verification is conducted in the BProVe program [21][22]. It verifies the diagrams in terms of safeness, proper termination, dead activities, and other properties. Its underlying logic is based on translating BPMN into MAUDE – a re-writing logic implementation. Another automatic tool VBPMN [23], uses a translation of BPMN into LOTOS NT process algebra formal notation and verification using the CDAP verifier. Its authors also provide a set of benchmarks [24] that are used during tests of the proposed automatic tool. Work [25] proposes verification using process automata that can be compared to our IMDS graphical view [26]; however, this technique concerns checking a single BPMN pool. In [20], the Bogor LTL checker is used to verify the workflows. As in other approaches, only total deadlocks are caught automatically; partial deadlocks require the specification of temporal formulas.

Some verification techniques concern only a limited set of BPMN elements, for example, in [25], communication between pools and boundary links are not considered.

Modeling in rules and processes is proposed in [27], in a spreadsheet in [28], Free-Choice Nets [29], Function-Behaviour-Structure Diagram [30], and Linked Data [31], but without a verification.

The detailed comparison of verification techniques and subsets of BPMN elements served in individual methods cannot be presented due to size limitation of the article. Some overviews of the verification tools and approaches can be found in [32], [33].

III. IMDS and BPMN

A. Overview of IMDS

IMDS formalism is addressed to distributed systems modeling. Its main idea is to show interactions between the two basic concepts: servers and agents. Servers $S=\{s_1,\dots,s_n\}$ are distributed computing nodes offering some services. Agents $A=\{a_1,\dots,a_k\}$ are distributed computations modeled as sequences of messages invoking servers' services. A system configuration T is a set of current servers' *states*=(*server, value*) – one state per server – and *messages*=(*agent, server, service*) of all agents – one message per agent. An interaction between servers and agents takes the form of *actions* in set F. Action is the execution of a service on a server by an agent message. The action transforms one configuration into another one, in which the server state and the agent message are replaced by new ones: ((*agent message, server state*), (*next agent message, next server state*)). There are also agent-terminating actions that do not produce a new. The system starts with an initial configuration T_0 containing initial states of all servers M_{init} and starting messages of all agents R_{init}. The formal definition of IMDS can be found in [6]. For an IMDS system we will use the notation $(S, A, M_{init}, R_{init}, F)$. IMDS semantics is defined by a Labeled Transition System

(LTS) where the nodes represent system configurations, and the arcs represent IMDS actions. The LTS root is the initial configuration.

B. Business Process Model and Notation (BPMN)

The BPMN standard includes four diagram types: Process, Collaboration, Choreography, and Conversation [3]. This article focuses on verifying Process and Collaboration diagrams, which describe the control. The elements used are Pools, Swimlines, Flow Objects, Data Objects, Connecting Objects, and Artifacts. Data Objects are not considered, because they do not have any semantic meaning when verifying the behavior [5]. BPMN lacks a formal definition framework, necessitating the need for a formal verification method.

C. BPMN Process Diagram syntax

BPMN Process Diagram is a graph $PD=(N,F)$, where N are nodes: *Activities* A, *Gateways* G (*Exclusive* G_X and *Parallel* G_P), *Events* E (*Start* E_s, *End* E_e and *Interrupting Boundary Intermediate* E_i). F are *Flows*: *Sequence* F_s, *Message* F_m and *Boundary Links* F_b. They are $F \subseteq N \times N$, $F_s \subseteq N \times N$, $F_m \subseteq A \times A$, $F_b \subseteq A \times E_i$.

In the proposed syntax, *nodes* correspond to *Flow Objects*. They play the role of building blocks of BPMN diagrams. Fig. 1 shows all major subtypes of *Flow Object* that are in the scope of this article and their graphical notation.

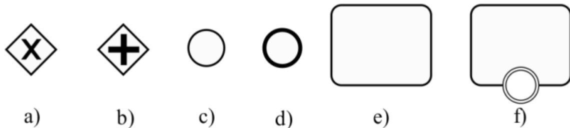


Fig. 1. The subset of BPMN elements formalized in this article: a) *Exclusive Gateway*, b) *Parallel Gateway*, c) *Start Event*, d) *End Event*, e) Activity f) Activity with attached Interrupting Boundary Inte*rmediate Event*

F is a non-symmetrical relation which means that for a given element $f=(n_1,n_2) \in F$, there is a directed connection from n_1 to n_2, $source(f)=n_1$ and $target(f)=n_2$. In addition to *Sequence Flows* and *Message Flows,* which are explicitly defined in the BPMN specification, we allow *Boundary Links*. Additionally, *Sequence Flows*, *Message Flows* and *Boundary Links* are subject to the following syntactic rules: *Message Flows* are between different *Pools*, *Sequence Flows* are inside a *Pool*, *Boundary Links* connect an *Activity* to an *Interrupting Boundary Intermediate Event*. We will also use the following notation: *in* are incoming flows, *out* – outgoing, *seq* concerns *Sequence Flows* and *mes* concerns *Message Flows*. For a given element E we have $in(E)$, $out(E)$, $in_{seq}(E)$, $out_{seq}(E)$, $is_{mes}(E)$, $out_{mes}(E)$.

Each maximal connected component of the underlying undirected graph of $P=(N,F_s \cup F_b)$ is called a BPMN *Pool* (an element for grouping nodes). Every node in N must be contained in some *Pool*. Additionally, we intentionally omit BPMN *Swimlanes* as the authors of the notation left its meaning to the modeler, which is ambiguous [5].

There are additional syntactical constraints that the graph (N,F) must fulfil: *Sequence Flows* cannot have the same node as source and target, *Start Events* must not have incoming *Sequence Flows*, *End Events* must not have outgoing *Sequence Flows*, and *Interrupting Boundary Intermediate Events* must have exactly one incoming flow: a *Boundary Link*.

D. BPMN Process Diagram semantics

The proposed BPMN semantics is based on the labeled transition system, called *state space*. For BPMN nodes we use the term *internal state* to distinguish it from IMDS *server state*.

While the general structure of a diagram – its nodes and flows – constitutes the static characteristics of a diagram, the concept of *tokens* and their interactions with nodes is introduced to describe the semantics of BPMN [3]. In contrast to [3], we model BPMN messages using tokens, which simplifies the semantics of *Message Flows*.

A *marking* is a pair $S = (D_{flows}, D_{nodes})$, where: D_{flows} is a distribution of tokens on flows of the given diagram, and D_{nodes} is a distribution of symbols {*neutral, activated, 1, 2, …*} on the nodes of the given diagram. The symbol of the given node N in D_{nodes} is the *internal state of N*. In the Initial Marking, the *Start Events* are in the *activated* state, all other nodes are in *neutral*, and all D_{flows} have 0 tokens. The *End Events* play the role of sinks for tokens.

The lifecycle of a BPMN element, is just an automaton: the loop of transitions between the states *neutral, activated, 1, 2,…,* and back to *neutral*. If there exists a *Boundary Link*, then a transition from every state to *neutral* is triggered by an *Interrupting Boundary Intermediate Event*.

In order to formalize BPMN execution semantics in a concise way, the authors propose to split the dynamics of a BPMN element into 3 phases: a*ctivation*, *execution pattern* and *completion. Activation* and *completion* are atomic transitions fired at the *beginning* and at the *ending* of BPMN element execution, respectively. *Execution pattern* in turn is a sub-automaton which simulates the stateful execution semantics of a given BPMN element.

Activation can be fired only if the required number of tokens were put on the incoming *Sequence Flows* or *Boundary Links* of a given BPMN element, and the node has been the *neutral* state. During *activation* of the BPMN element, two things happen atomically: the number of tokens from its incoming flows that triggered the activation is consumed (i.e. the tokens disappear) and the element's changes its state to *activated*.

Activation it is followed by the *execution pattern* which simulates arbitrary stateful interactions that the BPMN element may be subject to. *Execution pattern* is required to mark its ending with a *completion* transition to the *neutral* state of the BPMN element. This article focuses primary on BPMN elements that follow an *execution pattern* involving: empty automata, handling of *Message Flows* in *Activities* (i.e. inter-process communication) and handling of

Boundary Events in *Activities* (i.e. exception handling) which will be discussed later in this subsection.

Completion finalizes the execution of a BPMN element. Analogously to *activation*, the following two things happen atomically at the same time during it: a specified number of tokens is emitted along a subset of its outgoing Sequence Flows and the element changes its state back to *neutral*.

1. Activation patterns

We interpret the behavior of BPMN diagram as a token game: the tokens on arrows incoming to an element represent preconditions, and the tokens on arrows outgoing from an element represent postconditions. Each BPMN element follows either of the two activation patterns: XOR or AND. If a BPMN element follows the *XOR activation pattern*, a token on any of its incoming *Sequence Flows* or *Boundary Links* will enable activation of the element. In case of the *AND activation pattern*, a token will be put on each of its incoming *Sequence Flows* to enable its activation. If the number of tokens on any incoming Sequence Flow exceeds the number of tokens required for its activation, only the required number of tokens is consumed.

2. Execution patterns

Empty execution pattern. This pattern represents the trivial case when the element does not involve a complex stateful synchronization logic. That includes elements such as *Parallel* and *Exclusive Gateways* (control flow elements) as well as normalized *Activities* that do not have incident Message Flows or incident *Interrupting Boundary Intermediate Events*.

Handling of Message Flows. We define *handling of Message Flows of a given BPMN element E* as sequential generation of tokens on the outgoing *Message Flows*, and consumption of tokens from the incoming *Message Flows* incident to *E*, provided that the element was *activated*. The *Message Flows* are processed sequentially, in a left-to-right and then upper-to-lower order implied by graphical representation of the BPMN diagram. If the given incoming *Message Flow* does not contain a token, the message exchange will be blocked until such a token appears on it. Each numeric internal state (*1, 2, ...*) denotes how many *Message Flows* have been processed by the *Activity*.

Handling of Boundary Events (Exception handling). By *exception handling,* we mean the ability of *BPMN Activities* with *Boundary Events* to change their state to *neutral* at any point of their execution and generate a token on their incident *Boundary Link*. This definition effectively treats *Boundary Events* as *interrupting exceptions*.

3. Completion patterns

Analogously to *nodes activation*, each BPMN element follows either of the two *completion patterns*: XOR and AND. If a BPMN element follows the *XOR completion pattern*, a token will be generated along exactly one arbitrary outgoing *Sequence Flows*. If the BPMN element follows the *AND activation pattern*, a token will be generated on each of its outgoing *Sequence Flows*.

4. Final remarks on execution semantics

The activation and completion patterns for individual BPMN elements are: *Parallel Gateway* (AND, AND), *Exclusive Gateway* (XOR, XOR), *Activity* (XOR, AND), *Event* (XOR, AND).

Additionally, we assume throughout this article that BPMN uses interleaving semantics. It means that if many transformations are enabled in the diagram, one of them is executed, chosen in a non-deterministic manner. Choosing other semantics is also possible, but interleaving matches the semantics of IMDS used for verification. As shown in [34], every coincidence-based system can be transformed into an interleaved system.

The concepts of reachability, initial marking, reachable markings, and marking space are introduced to complement the semantics of BPMN.

Consider marking *A*. *Reachable markings* are all markings that can be reached from *A* by executing a sequence of transformations (nodes activation, message exchange, nodes completion, *Boundary Event* handling). The initial marking sets all Start Events to the *activated* state and all others to the *neutral*.

The transformation of a BPMN diagram include:

1. Node completion changes the state of the node to *neutral* and inserts tokens to the output *Sequence Flows* following the appropriate *Completion Pattern*.
2. Node activation removes the tokens from input *Sequence Flows* following the appropriate *Activation Pattern* and changes the state of the node from activated to the next state according to its *Execution Pattern*.
3. Message sending changes the state of the node to the next state and inserts a token to the *Message Flow*.
4. Message receiving removes the token from the *Message Flow* and changes the node state to the next state.
5. Executing a *Boundary Link* resets the state of the node to *neutral* and inserts a token into the *Sequence Flow* incident to the link.

The initial marking of the diagram is implied by its structure. Namely, all nodes with no incoming *Sequence Flows* that are not *Boundary Events* are initially in the activated state, i.e., they contain tokens.

The *marking space* of the BPMN diagram is the graph $G = (S_0, S, R)$, where S_0 is initial marking (position of tokens and internal states of nodes), S is a set of all reachable markings and R is a transformation relation moving the tokens and changing states of the nodes.

To sum up, vertices of *marking space* are markings of the given BPMN diagram, that are reachable from its initial marking. Initially, the tokens are present in all *Sequence Flows* outgoing from *Start Events*, as they are *activated*. Every transformation moves activation to the output of the *Sequence Flows*. Some transformations are executed with non-deterministic choices, like *Exclusive Gateways* and *Activities* with *Boundary Events*. If $|in_{mes}(n)|+|out_{mes}(n)|>0$, the enabled transformations contain sending and receiving

messages. Also, *Parallel Gateways* move the tokens to all their outgoing *Sequence Flows* atomically. All those rules are adopted in target IMDS models in a slightly different, but equivalent way. The difference lies in breaking atomicity with interleaving; see next section. The formal semantics of BPMN *marking space* is given by translation rules to IMDS system.

Apart from the braking atomicity of BPMN, the marking space of the BPMN diagram and the LTS of its translation to IMDS are the same. So both descriptions share the same semantics.

IV. Translation of BPMN into IMDS

A. Normalization

This is a preliminary step for translation to IMDS. Normalization strips the diagram of ambiguity:

- appending *Start Events* to all elements e which are not *Events* and for which $|in_{seq}(e)|=0$,
- appending *End Events* to all elements e for which are not *Events* and for which $|out_{seq}(e)|=0$,
- refactoring all BPMN elements that are not *Exclusive Gateways* and *Parallel Gateways* into semantically

Table I.

BPMN to IMDS translation rules

Group	Element type	Graphical symbol	Translation rules
1	*Pool* $\mathcal{S}$	$\mathcal{S}$	set of agents $A_{\mathcal{S}} \subseteq A$, server *pool*($\mathcal{S}$) = ($\mathcal{S}$, $V_{\mathcal{S}}$, $Q_{\mathcal{S}}$) , $V_{\mathcal{S}}$ = {*ready*} initial state (*pool*($\mathcal{S}$), *ready*)
2	*Sequence Flow f*	f	service $f \in Q_{\mathcal{S}}$
3	*Message Flow f* between *Activity A* in *Pool X* and *Activity B* in *Pool Y*,	A, Pool X, f, Pool Y, B	service $f \in Q_{and(A)}$, service $f \in Q_{and(B)}$, set *agents(f)* = $\{agent_f^j \mid j \in \{1,\dots,K\}\} \subseteq A$ initial messages for agents in *agents(f)*: $M_f = \{m_f^j \mid (agent_f^j, f) \wedge j \in \{1,\dots,K\}\} \subseteq M_{init}$
4	*Start Event* $\mathcal{E}$,	f	agent $a_{\mathcal{E}} \in agents(\mathcal{S})$, initial message $(a_{\mathcal{E}}, pool(\mathcal{S}), f) \in M_{init}$
5	*End Event* $\mathcal{E}$,	f	action $\{(agents(\mathcal{S}), pool(\mathcal{S}), f), (pool(\mathcal{S}), ready)\}$ $\rightarrow \{-, (pool(\mathcal{S}), ready)\}$ (agent-terminating action)
6	Nodes with 1 incoming & 1 outgoing *Sequence Flows*, no incident *Message Flows*,	f_1, f_2	action $\{(agents(\mathcal{S}), pool(\mathcal{S}), f_1), (pool(\mathcal{S}), ready)\}$ $\rightarrow \{(agents(\mathcal{S}), pool(\mathcal{S}), f_2), (pool(\mathcal{S}), ready)\}$
7	*XOR activation* / *XOR execution*, (nondeterministic choice)	in_1, in_k, out_1, out_l	set of actions: $\{(agents(\mathcal{S}), pool(\mathcal{S}), in_i), (pool(\mathcal{S}), ready)\}$ $\rightarrow \{(agents(\mathcal{S}), pool(\mathcal{S}), out_j), pool(\mathcal{S}), ready)\}$ $i \in \{1,\dots,n\}, j \in \{1,\dots,m\}$
8	*Interrupting Boundary Intermediate Event* $\mathcal{E}$ bound to *Activity* $\mathcal{T}$,	in, $\mathcal{T}$, out, $\mathcal{E}$, f	2 actions in *Pool* ($\mathcal{S}$): $\{(agents(\mathcal{S}), pool(\mathcal{S}), in), (pool(\mathcal{S}), ready)\}$ $\rightarrow \{(agents(\mathcal{S}), pool(\mathcal{S}), \varepsilon), (pool(\mathcal{S}), ready)\}$, $\{(agents(\mathcal{S}), pool(\mathcal{S}), in), (pool(\mathcal{S}), ready)\}$ $\rightarrow \{(agents(\mathcal{S}), pool(\mathcal{S}), out), (pool(\mathcal{S}), ready)\}$
9a	*AND activation* / *AND execution*,	in_1, in_k, out_1, out_l	Tuple $\begin{pmatrix} ANDserver, andActions, \\ poolActions, agents, initialMessages \end{pmatrix}$ = *createAND(n,* $\mathcal{S}$*, K)*
9b	*Activity* with incident *Message Flows* and optional *Interrupting Boundary Intermediate Event*,	m_1, m_n, in, out, $\mathcal{E}$, f	

equivalent constructs in which $|out_{seq}(e)|=1$ *and* $|in_{seq}(e)|=1$.

The first two rules follow directly from the BPMN specification of nodes that do not have incoming or outgoing *Sequence Flows* [3]. The third rule guarantees that elements that follow mixed XOR activation and AND completion semantics are transformed into equivalent groups of elements, following either XOR activation and XOR completion semantics or AND activation and AND completion semantics.

B. Translation of normalized graph into IMDS

The model is formally the tuple ($S, A, M_{init}, R_{init}, F$), The actions have the form ((*agent, server, service*), (*server, state*))/((*agent, other server, other service*), (*server, next state*)). The proposed BPMN-to-IMDS translation reflects the behavior of BPMN elements. *Server states* mimic the BPMN *internal states* while *agent messages* are tokens. Node activation, execution and completion are all mimicked by IMDS *actions*.

Elements of a BPMN diagram can be divided into ten groups with respect to the way that they are mapped into IMDS, as described in Table 1.

For Group (9) the authors propose a special procedure of translation called *createAND*(n, $\mathcal{P}$, K), transforming the given BPMN element e that is contained within a *Pool* $\mathcal{S}$ into a tuple (*ANDserver, andActions, poolActions, agents, initialMessages*) consisting of: a new *ANDserver*, its set of *andActions*, a set of *poolActions* in the corresponding *Pool* $\mathcal{S}$, new agents in *agents*($\mathcal{S}$), that become agents of the *Pool* $\mathcal{S}$, and *initialMessages* of the new agents. The new server mimics the atomic consumption and generation of multiple tokens, and to mimic consumption and generation of tokens. As proved in [34], coincident actions are equivalent to interleaved actions at the cost of adding new states between them.

Because IMDS agents cannot be created dynamically, they must be preallocated during the translation of BPMN into IMDS. Let us introduce a constant K to define how many agents are preallocated on each *Message Flow* or group (9) and used in the function, described in detail in [35].

C. Limitations of the proposed translation

The proposed translation method has some major issues which should be taken into account. The first is that IMDS agents cannot be created dynamically. In order to simulate the dynamic creation of tokens, the concept of *preallocation of agents* is introduced. The translation method cannot work for diagrams whose execution may generate infinitely many tokens (for example in a loop containing a *Parallel Gateway*). We refer to such diagrams as *unbounded*. The translation method proposed in this article can preallocate more IMDS agents than are needed. This is because the translation parameter, K, can be arbitrarily large. If K is allowed to be infinite, then the resulting IMDS system would not be static.

If the translation method is parametrized using $K = 1$, then only a single agent will be created for the *Sequence Flow f*. First execution of the *Parallel Gateway G* is correctly simulated by the translated IMDS model. The problem arises when the gateway is executed for the second time. In this case, there is no agent whose message can mimic the behavior of the token generated for the second time on the *Sequence Flow f*. Using parametrization *K=2* solves the problem, because there is a second preallocated agent, whose messages simulate the second execution. Thus, K has to be chosen carefully. It stems from the nature of IMDS. The choice of K value belong to the designer, it depends on how many token can come to a node “splitting” the behavior.

In the future, we plan to extend the plan to enrich IMDS to cover dynamic process creation, which will substitute agent preallocation.

The second major problem with the proposed translation method concerns BPMN elements with non-local semantics, like Inclusive Gateways. They were excluded from the proposed syntax, because the authors could not propose correct execution semantics for such elements. Thus, they are also not considered in the translation method. We don't plan to support this feature as non-local behavior is incompatible with distributed system.

V. EXAMPLES

A. AND activation and AND execution pattern

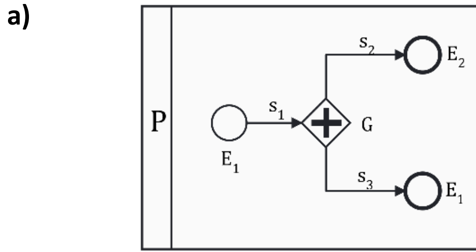
Here we use the graphical view of IMDS [26]. The example in Fig. 2 contains a *Pool P*, *Start Event E1*, two *End Events E2* and *E3*, *Parallel Gateway G*, and three *Sequence Flows*: *s1, s2, s3*. Those flows names are included in both *Pool server* and *AND server* definitions. We add the suffix *P* to *Pool server* services and *G* to *AND server* services, to differentiate between the two servers' services. Red dashed arrows show the transfers of agents between the servers, the arrows point to the states expected by the agents to perform their next actions.

Let $K = 2$ be the parametrization used in the translation. It can be the result of receiving more than one token acquired from the subdiagram represented by *E1* (for example tokens produced by a *Parallel Gateway*). The translation results in the following IMDS system:

1. (S, A, Minit, Rinit, F) = (
2. S = {(pool(P), {ready}, {s1P, s2P, s3P}), (and(G), {0,1,2}, {s1G, s2G, s3G})},
3. A = {a_{s1}, a_{s3}^{1}, a_{s3}^{2}},
4. Minit = {(pool(P), ready), (and(G), 0)},
5. Rinit = {(pool(P), a_{s1}, s1P), (and(G), a_{s3}^{1}, s3G), (and(G), a_{s3}^{2}, s3G)},
6. F = {
7. ((agents(P),pool(P),s1P),(pool(P),ready))→((agents(P),and(G),s1G),(pool(P),ready)),
8. ((agents(P),and(G),s1G), (and(G),0)) → ((agents(P),and(G),s2G),(and(G),1)),
9. ((agents(P),and(G),s3G), (and(G),1)) → ((agents(P),pool(P),s3P), (and(G),2)),

```
10. ((agents(P),and(G),s2G), (and(G),2)) →
    ((agents(P),pool(P),s2P), (and(G),0)),
11. ((agents(P),pool(P),s2P), (pool(P),ready)) →
    ((pool(P),ready)),
12. ((agents(P),pool(P),s3P), (pool(P),ready)) →
    ((pool(P),ready))})
```

a)



b)

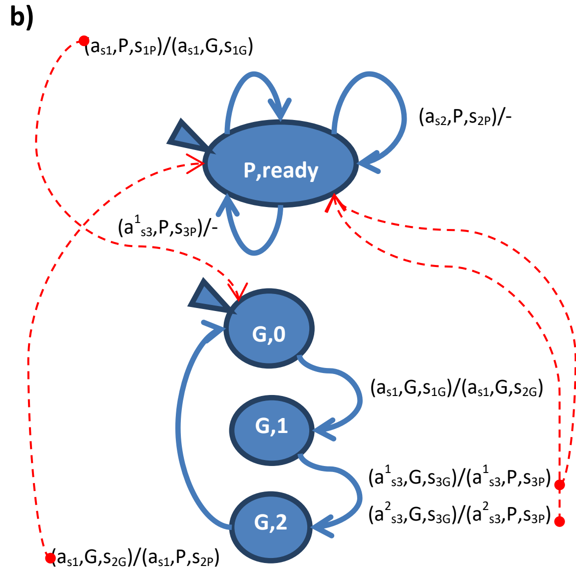


Fig. 2. a) *Exclusive Gateway* b) its translation to the IMDS automaton

Individual actions are responsible for: Action 7. starts the operation of the *AND server* by transferring the agent a_{s1} from the *Pool server* to *AND server*. Action 8. Transfers the incoming agent to the outgoing Sequence Flow s_2. Action 9. activates the preallocated agent a^1_{s3}, (or a^2_{s3}), transfers it to *the Pool server*, and changes the *AND server* state to *2*. Action 10. finishes counting the flows, transfers agent a_{s1} to the *Pool server*, and changes the state of *AND server* to *0*. Actions 11. and 12. terminate both agents as they are simulating behavior of the two *End Events*.

Note that action 9. can be executed by both agents a^1_{s3} and a^2_{s3}. However, any of those actions changes the state of *AND server* to *2*, prohibiting the other agent to execute its action. As a result, only one on the agents executes its action when the gate *G* is activated. Actions 7.-10. are created by the *createAND* procedure.

B. *partial deadlock*

Fig. 3 presents an example of a BPMN diagram that contains a partial deadlock. It consists of two *Pools*: *X* and *Y*. *Pool X* consists of one *Start Event*, one *End Event,* two *Exclusive Gateways*, and two *Activities*, one of which is connected through a *Message Flow* to the other *Pool*. *Pool Y* consists of one *Start Event*, one *End Event*, two *Parallel Gateways* connected in a fork manner, and two *Activities*, one of which is connected to an *Exclusive Gateway, Message Flow*, and a *Interrupting Boundary Intermediate Event* of type *Timer*.

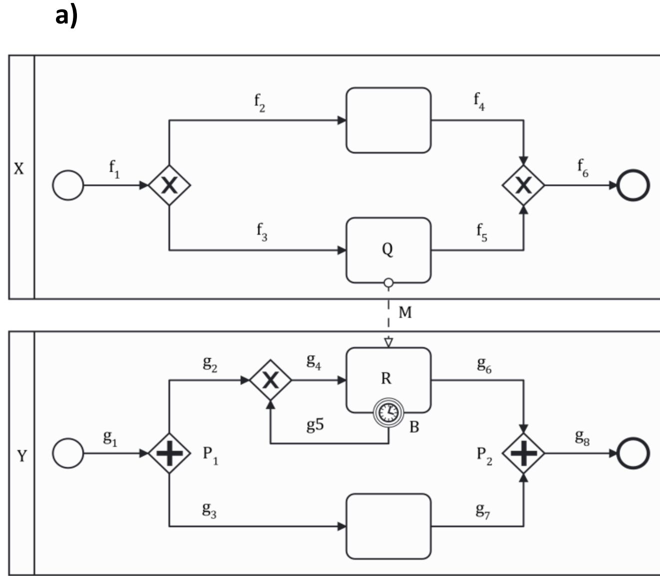
This BPMN diagram falls into partial deadlock if *Pool X* follows this execution path: f_1, f_2, f_4, f_6. In this case, *Activity R* in *Pool Y* will keep throwing timeout exceptions because the *Message Flow M* will not be fired. The proposed translation of BPMN diagrams into IMDS lets the designer keep track of such partial deadlocks using the Dedan tool, and backward way to *bpmn2imds* tool for observing the deadlock is source diagram. The agent in *Pool X* is not deadlocked since it terminates. The agent of *M* is technically deadlocked (its starting message will never cause an action), but we do not treat such situation as a real deadlock (however, the designer can draw some conclusions from this fact). One of the agents in *Pool Y* – a_{g1} is looping (g_4, g_5, g_4, g_5, ...); it does not deadlock but the question of its termination gives *false*. The question of possible termination gives *true* with the witness of the agent *X* following f_1, f_3, f_5, f_6, the "upper" agent *Y* – a_{g1} following g_1, g_2, g_4, g_6, g_8, the "lower" agent *Y* - a_{P1g3} following g_3, g_7, and the agent a_M following *Message Flow M*.

Let $K = 1$ be the parametrization used in the translation. The translation results in the following IMDS system; we omit the upper index in the services of *P* and *Q Activities*; we add the *X*, *Y*, *Q* and *R*, P_1 and P_2 suffixes to IMDS services for readability.

Let $K = 1$ be the parametrization used in the translation. The translation results in the following IMDS system; we omit the upper index in the services of *P* and *Q Activities*; we add the *X*, *Y*, *Q* and *R*, P_1 and P_2 suffixes to IMDS services for readability:

```
1.  (S, A, Minit, Rinit, F) = (
2.  S = {
3.  (pool(X), {ready}, {f1, f2, f3, f4, f5, f6}),
4.  (pool(Y), {ready}, {g1, g2, g3, g4, g5, g6, g7,
    g8}),
5.  (and(P1), {0, 1, 2}, {g1, g2, g3}),
6.  (and(P2), {0, 1, 2}, {g6, g7, g8}),
7.  (and(R), {0, 1, 2}, {g4, M, g6}),
8.  (and(Q), {0, 1, 2}, {f3, M, f5})},
9.  A = {af1, ag1, aM, aP1g3},
10. Minit = {(pool(X), ready), (pool(Y), ready),
    (and(Q), 0), (and(R), 0), (and(P1), 0),
    (and(P2), 0)},
11. Rinit = {(pool(X), af1, f1), (pool(Y), ag1, g1),
    (and(Q), aM, M),(and(P1), aP1g3, g3)},
12. F = {
13. ((agents(X),pool(X),f1),
    (pool(X),ready))→((agents(X),pool(X),f2),
    (pool(X),ready)),
14. ((agents(X),pool(X),f1),
    (pool(X),ready))→((agents(X),pool(X),f3),
    (pool(X),ready)),
15. ((agents(X),pool(X),f2),
    (pool(X),ready))→((agents(X),pool(X),f4),
    (pool(X),ready)),
```

```
16. ((agents(X),pool(X),f3),
    (pool(X),ready))→((agents(X),and(Q),f3),
    (pool(X),ready)),
17. ((agents(X),pool(X),f4),
    (pool(X),ready))→((agents(X),pool(X),f6),
    (pool(X),ready)),
```



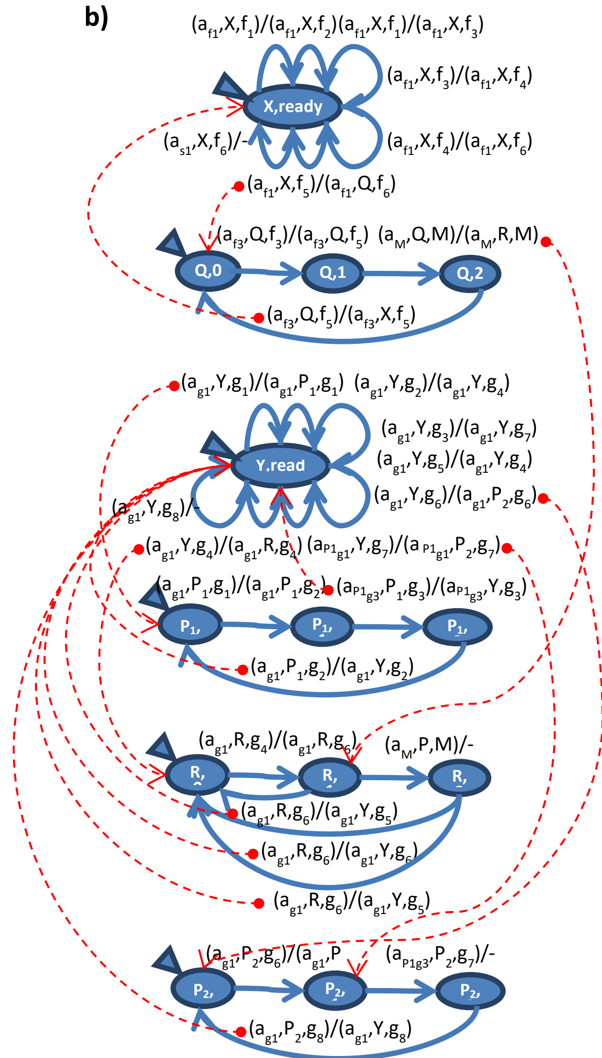


Fig. 3. Example BPMN (a) to IMDS (b) translation translation

```
18. ((agents(X),pool(X),f6),
    (pool(X),ready))→((pool(X),ready)),
19. ((agents(X),and(Q),f3), (and(Q),0)) →
    ((agents(X),and(Q),f5), (and(Q),1)),
20. ((agents(M),and(Q),M), (and(Q),1)) →
    ((agents(M),and(R),M), (and(Q),2)),
21. ((agents(X),and(Q),f5), (and(Q),2)) →
    ((agents(X),pool(X),f5), (and(Q),0)),
22. ((agents(Y),pool(Y),g1),
    (pool(Y),ready))→((agents(Y),and(P1),g1),
    (pool(Y),ready)),
23. ((agents(Y),and(P1),g1), (and(P1),0)) →
    ((agents(Y),and(P1),g2), (and(P1),1)),
24. ((agents(Y),and(P1),g2), (and(P1),1)) →
    ((agents(Y),pool(Y),g2), (and(P1),2)),
25. ((agents(Y),and(P1),g3), (and(P1),2)) →
    ((agents(Y),pool(Y),g3), (and(P1),0)),
26. ((agents(Y),pool(Y),g2),
    (pool(Y),ready))→((agents(Y),pool(Y),g4),
    (pool(Y),ready)),
27. ((agents(Y),pool(Y),g5),
    (pool(Y),ready))→((agents(Y),pool(Y),g4),
    (pool(Y),ready)),
28. ((agents(Y),pool(Y),g4),
    (pool(Y),ready))→((agents(Y),and(R),g4),
    (pool(Y),ready)),
29. ((agents(Y),pool(Y),g3),
    (pool(Y),ready))→((agents(Y),pool(Y),g7),
    (pool(Y),ready)),
30. ((agents(Y),and(R),g4), (and(Q),0)) →
    ((agents(Y),and(R),g6), (and (Q),1)),
31. ((agents(M),and(R),M), (and(Q),1)) → ((and
    (Q),2)),
32. ((agents(Y),and(R),g6), (and(Q),2)) →
    ((agents(Y),pool(Y),g6), (and (Q),0)),
33. ((agents(Y),pool(Y),g6),
    (pool(Y),ready))→((agents(Y),and(P2),g6),
    (pool(Y),ready)),
```

```
34. ((agents(Y),pool(Y),g7),
    (pool(Y),ready))→((agents(Y),and(P2),g7),
    (pool(Y),ready)),
35. ((agents(Y),and(P2),g6), (and(P1),0)) →
    ((agents(Y),and(P2),g8), (and(P1),1)),
36. ((agents(Y),and(P2),g7), (and(P1),1)) →
    ((and(P1),2)),
37. ((agents(Y),and(P2),g8), (and(P1),2)) →
    ((agents(Y),pool(Y),g8), (and(P1),0)),
38. ((agents(Y),pool(Y),g8), (pool(Y),ready)) →
    ((pool(Y),ready))})
```

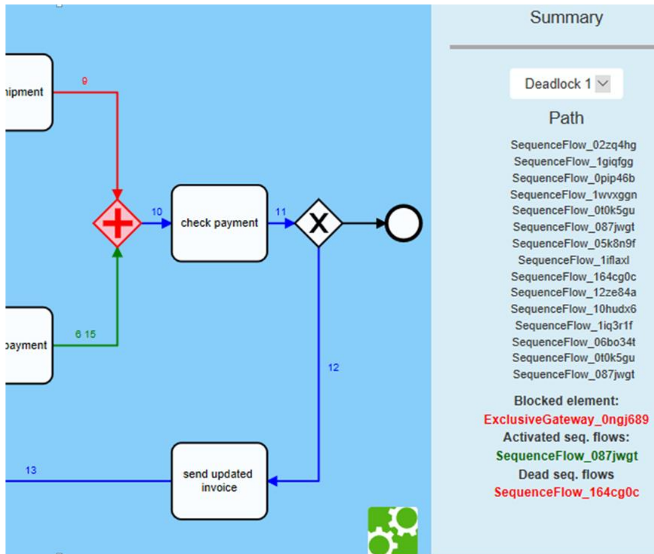


Fig. 4. Example operation of the *bpmn2imds* program.

Another example of a partial deadlock can concern the communication itself, when two Activities try to accept messages from each other. We would like to supplement the examples with more real-life ones, but text size constraints do not allow it.

VI. Conclusion and Future Work

The main goal of this article is to propose a translation of Business Process Collaboration and Process Diagrams into the IMDS, and verification of BPMN diagrams for deadlocks and termination. In order to achieve this, normalization of BPMN and translation of Process and Collaboration Diagrams into IMDS specifications was introduced. We identify partial (and total) deadlocks and check distributed termination. In IMDS, such checking is possible thanks to the preservation of information about component processes in the configuration space, and the development of temporal formulas independent of the structure of the analyzed system, and thus not requiring the designer to know temporal logic [6][7]. Compared to other tools, such as [9] and [13], our tool enables the visualization of diagrams and animation of their dynamics (not possible in [36]). Fig. 4 shows the example of deadlock animation. The designer is not limited to automatic verification of deadlocks/termination. The model can be automatically converted to the Uppaal tool [37], where arbitrary temporal questions can be asked, even with real-time constraints. However, reverse engineering of highlighting erroneous situations is impossible in such cases. Nevertheless, the simulation of a counterexample over the source BPMN diagram is preserved.

It should be noted that the complexity of every stage of work: conversion BPMN→IMDS, partial/total deadlock checking [38] and conversion IMDS→Uppaal are performed in linear time to size of a system (number of nodes and transitions). The example system of nearly 1 million configurations was checked in about an hour.F

What is rare in BPMN diagrams verification, we introduce *Boundary Links*, in order to formalize *Interrupting Boundary Intermediate Event* handling. The proposed semantics is characterized by *locality* – that is, the semantics of each of those elements depends only on other elements to which they are directly connected. During translation, the given BPMN diagrams are refactored into another semantically equivalent diagram to achieve consistency of activation and execution semantics.

One of the limitations of the proposed method is that it cannot handle diagrams that are unbounded, because it stems from the nature of finite state model checking. Another limitation is not considering BPMN elements with non-local semantics. Additionally, the need to use preallocated agents slows down the process of its analysis. Some elements are not covered by our translation, particularly *Event-based Gateways* and *Inclusive Gateways*. *Event-based Gateways* as they cannot be as easily translated into a model checking formalism. They need the creation of a set of agents of a purely technical nature to reset the gateway if an *Interrupting Boundary Intermediate Event* is bound to it. *Inclusive Gateways* have non-local semantics

We support the verification process with automatizing the verification process and giving run visualization and counterexample simulation properties (screenshots would take too much space, they can be found in [35]).

A possible improvement to the proposed translation method is to use preallocated agents for the entire diagram rather than for individual elements. This problem could be solved generally by introducing dynamic agent creation or agent reusability.

It may seem a controversial way of ordering messages sent and received by an *Activity* in the syntactic order of their appearance on the edge of the symbol. Other communication semantics can be envisioned. For example, first sending all outgoing messages and then waiting for all incoming. Alternatively, any order of sending and waiting for incoming messages. with the cost of exponential number of states in implementing server.

References

[1] W. M. P. van der Aalst, "The Application of Petri Nets to Workflow Management," J. Circuits, Syst. Comput., vol. 08, no. 01, pp. 21–66, Feb. 1998. doi: 10.1142/S0218126698000043

[2] W. Reisig, Understanding Petri Nets. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013. doi: 10.1007/978-3-642-33278-4

[3] Object Management Group, "Business Process Model and Notation

(BPMN) Version 2.0.2," 2013. http://www.omg.org/spec/BPMN/2.0.2
[4] C. Baier and J.-P. Katoen, Principles of Model Checking. Cambridge, MA: MIT Press, 2008. doi: 10.1007/978-3-030-12835-7_6
[5] F. Kossak et al., A Rigorous Semantics for BPMN 2.0 Process Diagrams. Cham: Springer International Publishing, 2014. doi: 10.1007/978-3-319-09931-6
[6] W. B. Daszczuk, "Specification and Verification in Integrated Model of Distributed Systems (IMDS)," MDPI Comput., vol. 7, no. 4, pp. 1–26, Dec. 2018. doi: 10.3390/computers7040065
[7] W. B. Daszczuk, "Using the Dedan Program," in Integrated Model of Distributed Systems, Cham, Switzerland: Springer Nature, 2020, pp. 87–97. doi: 10.1007/978-3-030-12835-7_6
[8] W. B. Daszczuk, "Fairness in Temporal Verification of Distributed Systems," in 13th International Conference on Dependability and Complex Systems DepCoS-RELCOMEX, 2-6 July 2018, Brunów, Poland, AISC vol.761, 2019, pp. 135–150. doi: 10.1007/978-3-319-91446-6_14
[9] R. M. Dijkman, M. Dumas, and C. Ouyang, "Semantics and analysis of business process models in BPMN," Inf. Softw. Technol., vol. 50, no. 12, pp. 1281–1294, Nov. 2008. doi: 10.1016/j.infsof.2008.02.006
[10] J. Billington et al., "The Petri Net Markup Language: Concepts, Technology, and Tools," in ICATPN 2003: Applications and Theory of Petri Nets, Eindhoven, The Netherlands, 23–27 June 2003, LNCS vol. 2679, 2003, pp. 483–505. doi: 10.1007/3-540-44919-1_31
[11] A. Rachdi, "Liveness and Reachability Analysis of BPMN Process Models," J. Comput. Inf. Technol., vol. 24, no. 2, pp. 195–207, Jun. 2016. doi: 10.20532/cit.2016.1002774
[12] K. Kluza, K. Jobczyk, P. Wiśniewski, and A. Ligęza, "Overview of Time Issues with Temporal Logics for Business Process Models," in 11 Federated Conference on Computer Science and Information Systems (FedCSIS), 11-14 Sept 2016, Gdansk, Poland, 2016, pp. 1115–1123. doi: 10.15439/2016F328
[13] C. Dechsupa, W. Vatanawood, and A. Thongtak, "Hierarchical Verification for the BPMN Design Model Using State Space Analysis," IEEE Access, vol. 7, pp. 16795–16815, 2019. doi: 10.1109/ACCESS.2019.2892958
[14] L. Li and F. Dai, "Transformation and Visualization of BPMN Models to Petri Nets," in International Conference of Green Buildings and Environmental Management (GBEM 2018), Qingdao, China, 23–25 Aug. 2018, IOP Conference Series: Earth and Environmental Science vol. 186, 2018, vol. 186, p. 012047. doi: 10.1088/1755-1315/186/5/012047
[15] R. Boussetoua, H. Bennoui, A. Chaoui, K. Khalfaoui, and E. Kerkouche, "An automatic approach to transform BPMN models to Pi-Calculus," in 2015 IEEE/ACS 12th International Conference of Computer Systems and Applications (AICCSA), Marrakech, Morocco, 17-20 Nov. 2015, 2015, pp. 1–8. doi: 10.1109/AICCSA.2015.7507176
[16] S. Yamasathien and W. Vatanawood, "An approach to construct formal model of business process model from BPMN workflow patterns," in Fourth International Conference on Digital Information and Communication Technology and its Applications (DICTAP), Bangkok, Thailand, 6-8 May 2014, 2014, pp. 211–215. doi: 10.1109/DICTAP.2014.6821684
[17] D. Prandi, P. Quaglia, and N. Zannone, "Formal Analysis of BPMN Via a Translation into COWS," in International Conference on Coordination Models and Languages COORDINATION 2008: Oslo, Norway, 4-6 June 2008, LNCS, vol. 5052, 2008, pp. 249–263. doi: 10.1007/978-3-540-68265-3_16
[18] M. Szpyrka, G. J. Nalepa, and K. Kluza, "From Process Models to Concurrent Systems in Alvis Language," Informatica, vol. 28, no. 3, pp. 525–545, Jan. 2017. doi: 10.15388/Informatica.2017.143
[19] J. Ye and W. Song, "Transformation of BPMN Diagrams to YAWL Nets," J. Softw., vol. 5, no. 4, pp. 396–404, Apr. 2010. doi: 10.4304/jsw.5.4.396-404
[20] V. Rafe and A. T. Rahmani, "A Graph Transformation-Based Approach to Formal Modeling and Verification of Workflows," in CSICC 2008: Advances in Computer Science and Engineering, Kish Island, Iran, 9-11 March 2008, 2008, pp. 291–298. doi: 10.1007/978-3-540-89985-3_36
[21] F. Corradini, F. Fornari, A. Polini, B. Re, F. Tiezzi, and A. Vandin, "BProVe: Tool support for business process verification," in 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), Urbana, IL, 30 Oct.-3 Nov. 2017, 2017, pp. 937–942. doi: 10.1109/ASE.2017.8115708
[22] F. Corradini, F. Fornari, A. Polini, B. Re, F. Tiezzi, and A. Vandin, "BProVe: A formal verification framework for business process models," in 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), Urbana, IL, 30 Oct.-3 Nov. 2017, 2017, pp. 217–228. doi: 10.1109/ASE.2017.8115635
[23] A. Krishna, P. Poizat, and G. Salaün, "VBPMN: Automated Verification of BPMN Processes," in 13th International Conference on integrated Formal Methods (iFM 2017), Turin, Italy, Sep 2017, 2017, pp. 1–8. http://convecs.inria.fr/doc/publications/Krishna-Poizat-Salaun-17.pdf. Accessed on 23.07.2023
[24] G. Salaün and P. Poizat, "VBPMN Samples." 2017. https://pascalpoizat.github.io/vbpmn-web/ Accessed on 23.07.2023
[25] N. Tantitharanukul, P. Sugunnasil, and W. Jumpamule, "Detecting deadlock and multiple termination in BPMN model using process automata," in 2010 ECTI International Confernce on Electrical Engineering/Electronics, Computer, Telecommunications and Information Technology, Chiang Mai, Thailand, 19-21 May 2010, 2010, pp. 478–482. https://ieeexplore.ieee.org/document/5491443 Accessed on 23.07.2023
[26] W. B. Daszczuk, "Graphic modeling in Distributed Autonomous and Asynchronous Automata (DA3)," Softw. Syst. Model., vol. 20, no. 5, pp. 363–398, 2021. doi: 10.1007/s10270-021-00917-7
[27] K. Kluza and G. J. Nalepa, "Towards Rule-based Pattern Perspective for BPMN 2.0 Business Process Models," in 11 Federated Conference on Computer Science and Information Systems (FedCSIS), 11-14 Sept 2016, Gdansk, Poland, 2016, pp. 1359–1364. doi: 10.15439/2016F324
[28] K. Kluza and P. Wiśniewski, "Spreadsheet-Based Business Process Modeling," in 11 Federated Conference on Computer Science and Information Systems (FedCSIS), 11-14 Sept 2016, Gdansk, Poland, 2016, pp. 1355–1358. doi: 10.15439/2016F376
[29] W. M. P. van der Aalst, "Using Free-Choice Nets for Process Mining and Business Process Management," in 16 Federated Conference on Computer Science and Information Systems (FedCSIS), 2-5 Sept 2021, Sofia, Bulgaria, 2021, pp. 9–15. doi: 10.15439/2021F002
[30] S. J. Niepostyn and I. Bluemke, "The Function-Behaviour-Structure Diagram for Modelling Workflow of Information Systems," in CAiSE 2012: International Conference on Advanced Information Systems Engineering, Gdańsk, Poland, 25-26 June 2012, 2012, pp. 425–439. doi: 10.1007/978-3-642-31069-0_34
[31] S. Robak, B. Franczyk, and M. Robak, "Applying Linked Data concepts in BPM," in 6 Federated Conference on Computer Science and Information Systems (FedCSIS), 9-12 Sept 2012, Wrocław, Poland, 2012, pp. 1105–1110. url: https://ieeexplore.ieee.org/abstract/document/6354386 Accessed on 23.07.2023
[32] A. Suchenia, P. Wiśniewski, and A. Ligęza, "Overview of Verification Tools for Business Process Models," in 12 Federated Conference on Computer Science and Information Systems (FedCSIS), 3-6 Sept 2017, Prague, Czech Republic, 2017, pp. 295–302. doi: 10.15439/2017F308 doi: 10.15439/2017F308
[33] T. Lopes and S. Guerreiro, "Assessing business process models: a literature review on techniques for BPMN testing and formal verification," Bus. Process Manag. J., vol. 29, no. 8, pp. 133–162, Apr. 2023. doi: 10.1108/BPMJ-11-2022-0557
[34] Z. Manna and A. Pnueli, The Temporal Logic of Reactive and Concurrent Systems. New York, NY: Springer, 1992. doi: 10.1007/978-1-4612-0931-7
[35] J. Jałowiec, "Translation of Business Process Model and Notation into Integrated Model of Distributed Systems," B.Sc. thesis, Warsaw University of Technology, Institute of Computer Science, 2019. https://repo.pw.edu.pl/info/bachelor/WUT31de757656da422c87be61e7ede00630/?r=diploma&tab=&lang=pl Accessed on 23.07.2023
[36] F. Corradini, C. Muzi, B. Re, L. Rossi, and F. Tiezzi, "Global vs. Local Semantics of BPMN 2.0 OR-Join," in 44th International Conference on Current Trends in Theory and Practice of Computer Science, Krems, Austria, 29 Jan. - 2 Feb., 2018, LNCS, vol. 10706, 2018, pp. 321–336. doi: 10.1007/978-3-319-73117-9_23
[37] G. Behrmann, A. David, K. G. Larsen, P. Pettersson, and W. Yi, "Developing UPPAAL over 15 years," Softw. Pract. Exp., vol. 41, no. 2, pp. 133–142, Feb. 2011. doi: 10.1002/spe.1006
[38] W. B. Daszczuk, "Evaluation of temporal formulas based on 'Checking By Spheres,'" in Euromicro Symposium on Digital Systems Design, Warsaw, Poland, 4-6 Sept. 2001, 2001, pp. 158–164. doi: 10.1109/DSD.2001.952267