

A Formal Approach for the Analysis of BPMN Collaboration Models

Flavio Corradini^a, Fabrizio Fornari^a, Andrea Polini^a, Barbara Re^{a,*}, Francesco Tiezzi^a and Andrea Vandin^b

^aUniversity of Camerino, Via Madonna delle Carceri 7, 62032 Camerino, Italy

^bSant'Anna School of Advanced Studies, Piazza Martiri della Libertà 33, 56127 Pisa, Italy

ARTICLE INFO

Keywords:

BPMN

Collaboration

Verification

Model Checking

Statistical Model Checking

ABSTRACT

BPMN collaboration models have acquired increasing relevance in software development since they shorten the communication gap between domain experts and IT specialists and permit clarifying the characteristics of software systems needed to provide automatic support for the activities of complex organisations. Nonetheless, the lack of effective formal verification capabilities can hinder the full adoption of the BPMN standard by IT specialists, as it prevents precisely check the satisfaction of behavioural properties, with negative impacts on the quality of the software. To address these issues, this paper proposes BProVe, a novel verification approach for BPMN collaborations. This combines both standard model checking techniques, through the MAUDE's LTL model checker, and statistical model checking techniques, through the statistical analyser MULTIVESTA. The latter makes BProVe effective also on those scenarios suffering from the state-space explosion problem, made even more acute by the presence of asynchronous message exchanges. To support the adoption of the BProVe approach, we propose a complete web-based tool-chain that allows for BPMN modelling, verification, and result exploration. The feasibility of BProVe has been validated both on synthetically-generated models and on models retrieved from two public repositories. The performed validation highlighted the importance and complementarity of the two supported verification strategies.

1. Introduction

A business process model describes a set of activities that an organisation should perform to fulfil a specific business goal [48]. Furthermore, it is possible to use the so-called *collaborations* to describe the coordination of processes belonging to different organisations willing to cooperate to achieve a shared goal. As it happens for any modelling activity, the description of the reality of interest through the usage of a modelling notation permits to reduce the communication gap between the various users of the model, keeping the focus just on those aspects are considered relevant for the specific objective. In this respect, collaboration models help maintain the modeller's attention on the alignment of the internal behaviour of a set of processes concerning inter-process communication.


Business process modelling has been initially introduced for documentation purposes by business analysts, but it shortly started to be adopted in software development, in particular about requirements engineering activities [13, 26, 47, 65]. Successively it has gained popularity in the development of software systems supporting business process execution, and as a starting point for model-driven development of distributed systems [53, 4], as also testified by the emergence of several engines enabling the direct execution of

business process specifications (e.g., Camunda¹, Signavio², Bonita³).

A similar path has been followed in the last years by the Business Process Model and Notation (BPMN 2.0) [52], an Object Management Group (OMG) standard. The notation emerged as one of the most adopted proposals to define business process models. The success of BPMN comes from its versatility and capability to represent business processes for different purposes. The notation acquired acceptance, at first, within the business analysts community, and successively it has been more and more adopted by IT specialists to drive the development and settlement of IT systems supporting the execution of a specified process model. This shift in notation usage is particularly relevant, and it poses the basis for our work. Indeed, the adoption of BPMN for shaping IT systems and for the application of model-driven approaches to automatic code generation requires the definition of a formal verification approach to increase confidence in the quality of implemented software systems [24].

While the research community has devoted a relevant effort to support verification of single business processes, to the best of our knowledge, there is still no concrete proposal supported by tools to analyse large collaborative processes. On the other hand, modelling of such scenarios has become more and more common in practice. This is certainly a consequence of the extensive introduction of effective software system integration technologies, such as REST-based services. This has permitted to derive collaborative systems from the integration of independently developed and managed software, and to figure an API economy where openly

*Corresponding Author

 falvio.corradini@unicam.it (F. Corradini);

fabrizio.fornari@unicam.it (F. Fornari); andrea.polini@unicam.it (A. Polini); barbara.re@unicam.it (B. Re); francesco.tiezzi@unicam.it (F. Tiezzi); andrea.vandin@santannapisa.it (A. Vandin)

ORCID(s): 0000-0002-3620-1723 (F. Fornari); 0000-0002-2840-7561 (A. Polini); 0000-0001-5374-2364 (B. Re); 0000-0003-4740-7521 (F. Tiezzi); 0000-0002-2840-7561 (A. Vandin)

¹<https://camunda.com>

²<https://www.signavio.com/>

³<https://www.bonitasoft.com/>

documented interfaces, for instance adopting OAI⁴ formats, are made available and can be accessed using precisely described interaction protocols, so to create an ecosystem fostering software system collaborations [36, 64].

In deriving a verification approach for collaborative scenarios, which could be used in real contexts, we identified two additional characteristics that we judged of primary relevance. The first one concerns the fact that a collaborative scenario is by its very nature a parallel scenario, which then could easily lead to a possible explosion of the state space to be managed, making traditional verification strategies not always effective. We initially experimented with such an issue while running the experiments reported in Section 8 using the “standard” verification strategy. Therefore, the approach has been augmented to include a statistical model checking strategy. The second characteristic refers instead to the typical difficulties of introducing formal verification techniques in general modelling contexts, where it is often the case that a modeller does not have a strong background in formal methods. So, to make accepting the proposed approach easier, we decided to make available basic verification features via pre-configured and stereotyped properties. We provide a GUI where properties can be derived by selecting and composing the entries made available via a set of pop-up menus. In such a way, a modeller can start to use and experiment with our approach even without a clear understanding of the verified properties. This could help the interested modeller to acquire confidence in the approach through its usage. It is worth noticing that the modeller is not asked to modify, in any way, his/her modelling habits. The approach includes optional mechanisms for those users with a good understanding of LTL (Linear Temporal Logic) verification, enabling them to define and check customised temporal properties.

The resulting verification approach, called BProVe⁵, is offered as a web-based tool-chain⁶ that allows to graphically observe the results of the verification directly on the models under scrutiny. The verification component is provided as a REST web service that, thanks to the BPMN standard usage, allows to check properties of BPMN models independently from the modelling environment used to create them. The web service has also been integrated within an Eclipse plugin, allowing its integration in the Eclipse platform.

We have extensively validated the effectiveness of the BProVe verification approach by running scalability tests on ad-hoc designed and synthetically generated models, as well as on models from two open repositories “BPM Academic Initiative Model Collection (BPMAI)”⁷ and “Camunda BPMN for Research”⁸. This analysis has shown the potential of BProVe both on limiting-case scenarios and on realistic ones and highlighted the advantages of having two analysis engines offering complementary analysis techniques.

Summing up, the most distinctive features of our BProVe

approach are following reported.

1. The use of direct semantics for BPMN models, both for single processes and collaborations, without requiring any intermediate encoding. This allows for an easy result exploration, i.e., to graphically interpret the analysis results directly on the actual BPMN model.
2. An automated formal verification approach that permits the *analysis* of BPMN collaboration models with potentially large state spaces by integrating standard and statistical model checking analysis capabilities. The need for the inclusion of two different verification strategies emerged after observing the presence of collaborative models with large state space for which standard verification strategies were not able to provide an answer. The validation we performed aimed at assessing that the inclusion of two different strategies is indeed valuable, as they show complementary characteristics. It also permits to enlarge the set of models for which the approach can provide an answer.
3. The robust, efficient and accessible tool support allowed us to perform an extensive *validation* of the approach confirming its scalability and the complementarity of the two supported analysis techniques. The tool is offered as a REST service. A web-based frontend is available, hiding all the formalism involved, thus enabling the *verification as a service* paradigm.

The paper is organised as follows. Section 2 provides an overview of the BPMN standard and of the operational semantics at the basis of our work, using a collaboration scenario exploited as a running example in the rest of the paper. Section 3 discusses the BProVe approach presenting its main characteristics. Section 4 describes how properties can be defined in the BProVe approach, while Section 5 discusses how they can be verified, and Section 6 exemplifies this on the running example. Section 7 presents the tool-chain’s architecture and user interface. Section 8 illustrates the conducted validation experiments. Finally, Section 9 thoroughly compares our approach with related works available in the literature and Section 10 concludes by also touching upon directions for future work.

2. Background Notions

In this section we introduce the BPMN standard together with a scenario, used as running example throughout the paper, to illustrate our proposed verification approach. Then, we introduce the implemented operational semantics at the basis of BProVe.

2.1. Modelling Collaborations in BPMN

BPMN, an OMG standard [52], is currently acquiring a clear predominance among the proposed notations to model business processes thanks to: (i) its intuitive and graphical notation that is widely accepted by the industry and the

⁴<https://www.openapis.org/>

⁵<http://pros.unicam.it/bprove>

⁶<http://pros.unicam.it/bprove-web-interface>

⁷<https://bpmai.org/download>

⁸<https://github.com/camunda/bpmn-for-research>

academia; and (ii) the support provided by a broad spectrum of modelling tools.⁹

Here we discuss the BPMN elements supported by our approach and reported in Fig. 1. In our proposal, we selected a subset of BPMN elements following the pragmatic approach of retaining those features most used in practice [50]. *Pools* represent participants or organisations providing details on internal process specifications and related elements. Pools are drawn as rectangles. *Tasks* represent specific jobs to be performed within a process. Tasks are drawn as rectangles with rounded corners. *Gateways* manage the flow of a process both for parallel activities and choices. Gateways are drawn as diamonds and act as either join nodes (merging incoming sequence edges) or split nodes (forking into outgoing sequence edges). Different types of gateways are available: a *XOR gateway* describes choices, an *AND gateway* enables parallel execution flows, an *Event-Based gateway* activates its outgoing branches according to the taking place of catching events, and an *OR split gateway* which allows to execute one or more of its outgoing flows. *Events* are used to represent something that can happen. An event can be a *Start Event*, representing the point from which the process starts, an *Intermediate Event* representing something that happens during process execution (e.g., the sending/receiving of a message), or an *End Event* representing the process termination. Events are drawn as circles. We also refer to a particular type of end event, the *Terminate End Event*, displayed by a thick circle with a darkened circle inside; it stops and aborts the running process. *Connecting Edges* connect process elements in the same or different pools. A *Sequence Edge* is a solid connector used to specify the internal flow of the process, thus ordering elements in the same pool, while a *Message Edge* is a dashed connector used to visualise communication flows between organisations. A set of pools interacting through message exchanges form a *collaboration model*.

The model depicted in Fig. 2 presents a collaboration process between three participants: a *Travel Agency*, an *Airline* reservation system, and a *Customer*. The goal of the collaboration is to provide a travel offer to a customer and handle the response. The Travel Agency triggers the collaboration process, which elaborates a travel offer and sends it to a Customer. The Customer evaluates the received offer deciding whether to accept it or to reject it. The Travel Agency and the Airline handle the customer response, either by confirm the booking and handling the payment, or by terminating their processes. The execution of BPMN models is based on the notion of *tokens*, graphically denoted as black dots labelling BPMN elements (see the start events in Fig. 2). The presence of such tokens over the start events enables the execution of the three processes (representing the collaboration's initial status). Tokens traverse the sequence edges of processes and pass through their elements enabling their execution. The notation element's specific characteristics define the rules to follow to move, consume and generate tokens.

⁹Currently more than 50, see www.bpmn.org for a detailed list.

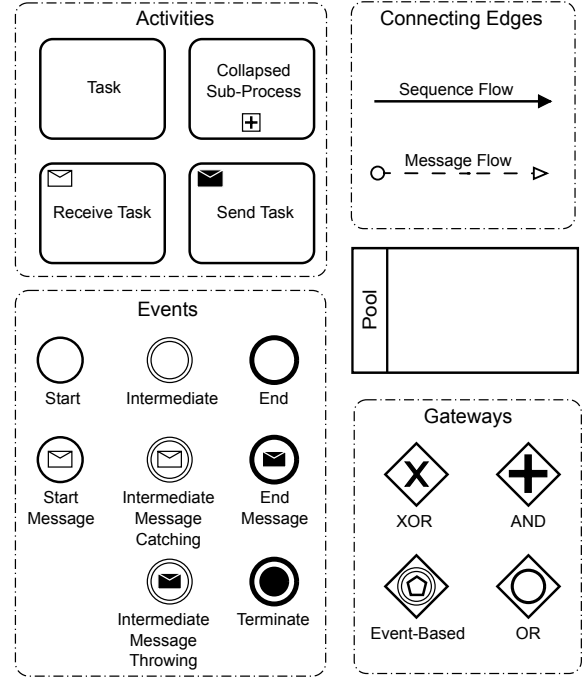


Figure 1: BPMN notation (considered elements).

2.2. BPMN Operational Semantics

The interpretation, or semantics, of the BPMN standard is given in informal natural language. In order to obtain a formal verification approach for such standard, we had first to make this semantics *formal* in [15]. This formal semantics has been then implemented in the form of an executable interpreter in MAUDE in [17, 18]. The full MAUDE implementation of our BPMN interpreter is available at <https://github.com/PROSLab/BPMNOS-Maude>. In the following, using our running example, we exemplify the BPMN syntax and semantics as implemented in MAUDE to allow the reader to grasp the main concepts behind it.

BPMN syntax in MAUDE

Listing 1 provides part of the textual representation in our MAUDE interpreter for our running example. Notably, Listing 1 specifies that the model is in the initial configuration as depicted in Fig. 2, with a token in the start event of each process in the configuration. The complete specification of our example is given in Appendix A.

As we can see from Listing 1, a collaboration is specified using the operator *collaboration* (Line 1), which takes a set of pools as arguments (one for each pool in the model). A pool, defined by means of the operator *pool* (Lines 2, 8, and 16), takes as argument its name, a BPMN process identified by the operator *proc*, and a set of incoming and outgoing messages listed after *in:* and *out:*, respectively, to communicate with other pools. We can see from Line 6 that a message is declared using its name ("offer"), followed by the operator *.msg*, in turn followed by an integer denoting the number of tokens present at the message element. The BPMN semantics, indeed, makes use of the token concept to intuitively de-

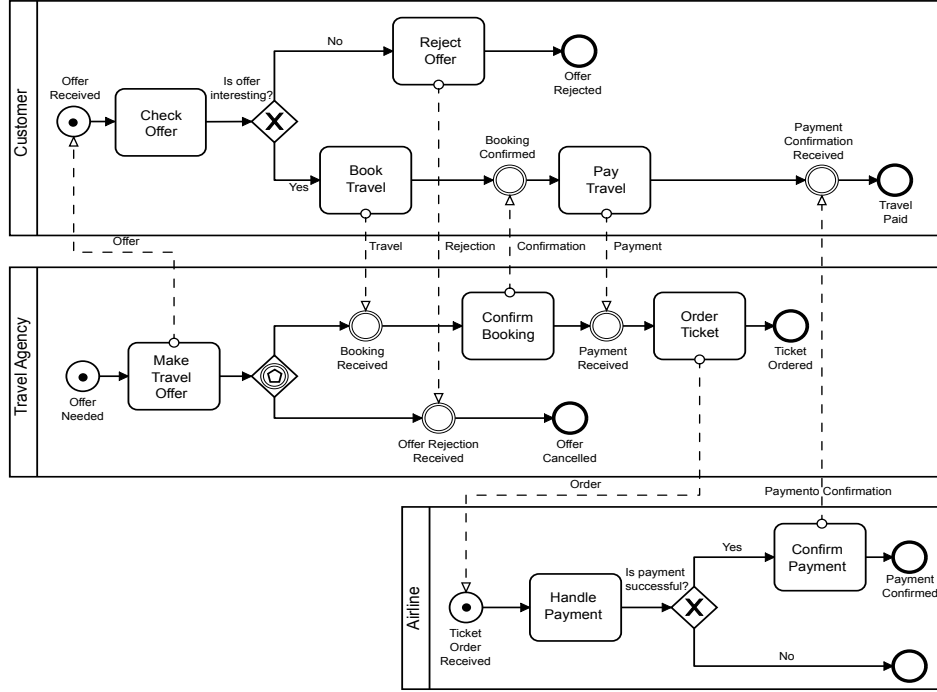


Figure 2: A collaboration model. Tokens denote an initial state (adapt. from [25, p.130]).

```

1 collaboration(
2   pool( "Customer" ,
3     proc( startRcv ( enabled , " e1 " . 0 , "Offer".msg 0 ) |
4       task( disabled , "e1" . 0 , "o1". 0 , "Check Offer") |
5       ProcElements
6     ) , in: "Offer" .msg 0 andmsg IMsgSet , out: OMsgSet ) |
7
8   pool( "Travel Agency" ,
9     proc( start( enabled , "e2". 0 ) |
10      taskSnd ( disabled , "e2" . 0 , "o2" . 0 ,
11        "Offer" .msg 0 , "Make Travel Offer") |
12      ProcElements
13    ) , in: IMsgSet , out: "Offer" .msg 0 andmsg
14      "Order" .msg 0 andmsg OMsgSet ) |
15
16   pool( "Airline" ,
17     proc( startRcv ( enabled , "e3". 0 , "Order" .msg 0 ) |
18       ProcElements
19     ) , in: "Order" .msg 0 andmsg IMsgSet , out: OMsgSet )
20 ).

```

Listing 1: Sketch of MAUDE encoding of model in Fig. 2.

scribe the execution flow. We note that the operator `andmsg` is used to compose sets of messages, while eventual further messages are denoted in Listing 1 by a place-holder `IMsgSet` to represent incoming messages, and `OMsgSet` for outgoing ones.

A BPMN process is specified using the operator `proc`, having as argument the set of BPMN elements (separated by `|`) that composes it. For example, Lines 3-5 show the process of the Customer. Similarly to what has been done for

messages, for the sake of readability in the listing we explicitly report only the elements depicted in the left-most part of the pools in Fig. 2, while we use the place-holder `ProcElements` to denote the other elements. The control flow is specified by the presence of tokens assigned to each process element. Process elements are allowed to act only when enabled, which means they hold a token. In Line 9 we see that the start element of the pool Travel Agency is enabled, meaning that it has a token in input (denoted by the black dot within the start event in Fig. 2), and hence it is allowed to initiate the process. Even if the start events of Customer and Airline (Lines 3 and 17) have a token, they are not allowed to initiate until the corresponding messages are received. The topology of the process is defined by the edges specified as arguments of the process components. In the example, the start node of the Customer (operator `startRcv` in Line 3) is connected via sequence edge `e1` to the input of the task (operator `task`) defined in Line 4, whose output is in turn connected to other elements in `ProcElements` via sequence edge `o1`. The start node of the Travel Agency (operator `start` in Line 9) is connected via sequence edge `e2` to the input of the task (operator `task` in Line 10), whose output is in turn connected to the other elements in `ProcElements` via sequence edge `o2`. Finally, the start node of the Airline (operator `startRcv` in Line 17) is connected via sequence edge `e3` to the other elements in `ProcElements` via sequence edge `o3`. Sequence edges that have an associated value `0` do not include any token.

BPMN semantics in MAUDE

In MAUDE, the semantics is specified in terms of rewriting rules which are exhaustively applied by pattern matching on each generated state, starting from the initial one, until no new states can be generated. A rewriting rule has the following form: `cr1 [Label] : Term-1 => Term-2 if Condition(s) .`

The keyword `cr1` stands for ‘conditional rewriting rule’, whose optional name is specified in the square brackets. The body of the rule, `Term-1 => Term-2`, specifies that if `Term-1` can be matched on the part of a state; then a new state can be obtained by (i) removing the matched part from the state, and (ii) adding `Term-2` to the remaining part of the state. In the example, one of such terms can be the entire collaboration, a pool, a process, or a BPMN element. The `if` defines a guard that has to be satisfied by the considered state to enable the application of the rule. In case no condition is required, then the `if` clause is omitted, and the keyword `r1` is used in place of `cr1`.

The BPMN semantics we defined is multi-layer, meaning that it has rules for collaborations (layer 4) that depend on rules for pools (layer 3), which in turn depend on rules for processes (layer 2), triggered by rules for single BPMN elements (layer 1). Roughly, the semantics is given in this form: if a BPMN element `e11` can evolve in an element `e12`, then a process `proc1` containing `e11` can evolve in a process `proc2` obtained by replacing `e11` with `e12`, and similarly for the higher layers, if necessary keeping into account interactions with other processes or pools. This can be mimicked in MAUDE using the conditions `if e11 => e12` and `if proc1 => proc2` as sketched in Listing 2.

```
cr1 [SketchOfRuleForProcesses] :
  e11 | RestOfProcess =>
  e12 | RestOfProcess if e11 => e12 .

cr1 [SketchOfRuleForPools] :
  pool("Name", proc(proc1)) =>
  pool("Name", proc(proc2)) if proc1 => proc2 .
```

Listing 2: Sketch of rules for part of processes and pools semantics.

3. The BProVe Approach

In the Introduction, we have discussed the rationale of combining business process management and software development. In this section, we concentrate on *modelling* and *analysis*, which are activities usually completed in an iterative way until reaching a stable version of the model at the basis of the software system to be developed. In particular, we present how the BProVe approach allows shortening the distance between modelling and analysis supporting all the phases of the model development cycle reported in Fig. 3.

Model Design

Model Design involves stakeholders in collecting domain requirements to produce a model suitable to represent as-is or to-be scenarios within organisations. Our BPMN col-

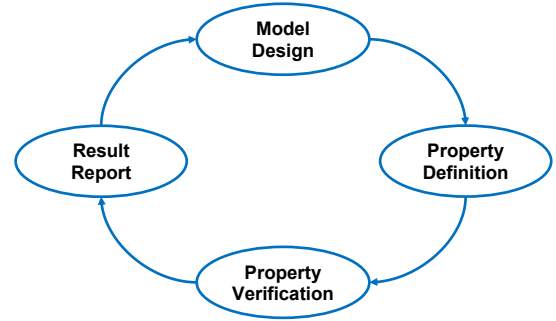


Figure 3: Development cycle of models.

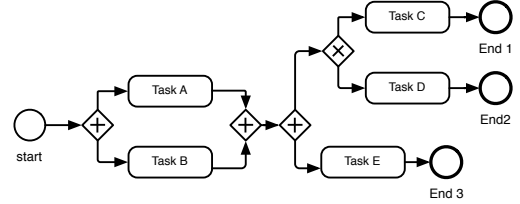


Figure 4: A non well-structured process model.

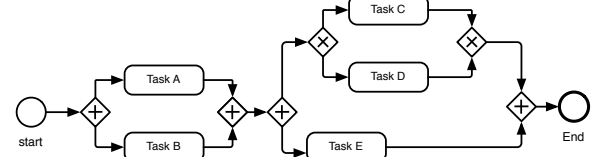


Figure 5: A well-structured process model.

laboration model and corresponding semantics give specific support for representing processes of different organisations which interact to achieve a common goal.

The BProVe approach makes it possible to reason directly in terms of BPMN models with an arbitrary topology without making any assumption on the structure of the collaboration model (i.e., well-structuredness, which asks for the proper composition of nested structures [27]). This gives us the possibility to analyse both well-structured and unstructured models available in the literature [19]. As an example, the process in Fig. 5 is the well-structured version of the unstructured process in Fig. 4. Notice, the notion of well-structuredness is extended from processes to collaborations requiring to be satisfied by all the processes involved in a collaboration [19]. By using BProVe, the analysis can be done directly on the designed models, including poorly designed models, without any redesign imposed by the analysis techniques as other verification approaches require (e.g., [31]). Modellers are free to represent the reality they perceive up to their modelling experience [56], hence the modelling activity results to be less complex [41] and more expressive [57, 58].

Property Definition

Property Definition relates both the internal characteristics of a single process in a collaboration and the whole collaboration. This holds both for generic properties that are well-established in the business process domain, such as *soundness* [25] and *safeness* [1], and for ad-hoc properties

specifically defined for given application scenarios.

Soundness can be described as the combination of three basic properties concerning the behaviour of a process model:

- (i) *Option to Complete*: any running process instance must eventually complete;
- (ii) *Proper Completion*: at the moment of completion, each token of the process instance must be in a different end event;
- (iii) *No dead activities*: any activity can be executed in at least one process instance.

On the other hand, *safeness* refers to the occurrence of no more than one token at the same time along the same sequence edge. The satisfaction of these properties is generally considered a minimum guarantee to avoid unexpected behaviours [67].

Besides system-independent properties, such as the ones mentioned above, we also consider application-dependent properties specifically defined for the given application scenario. Such kind of properties are relevant both at the process level, i.e., considering the execution of tasks within a single process, as well as at the collaboration level, i.e., considering the effects of message exchanges. As a relevant example, and in relation to the model reported in Fig. 2, we refer to the following properties/queries.

- (P1) Does the start of the Confirm Booking task in the Travel Agency pool implies that the same task will sooner or later complete (Property 1)?
- (P2) Does the completion of a specified task in the Airline pool, say Handle Payment, implies the completion of another task in the same pool, say Confirm Payment (Property 2)?
- (P3) If the Customer has sent the payment to the Travel Agency, may it happen that the corresponding confirmation, by the Airline, is never received (Property 3)?

BProVe directly allows the verification of soundness and safeness properties for any model, as well as it gives the possibility to specify and verify application-dependent properties.

Property Verification

Property Verification enables to check the considered properties detecting behavioural issues of the model. To do that, the model behaviour is systematically explored to establish if a property of interest formally holds [6].

In checking properties, BProVe offers two different analysis techniques, i.e. *LTL model checking* and *statistical model checking*, as alternative and, in some cases, complementary means to analyse BPMN models.

The LTL model checking technique we use is the one supported by the MAUDE LTL model checker [30]. It allows for an exhaustive exploration of the model reachable states and provides a reliable response in the case of systems with a finite state space. However, model checking techniques are known to be affected by the state-space explosion

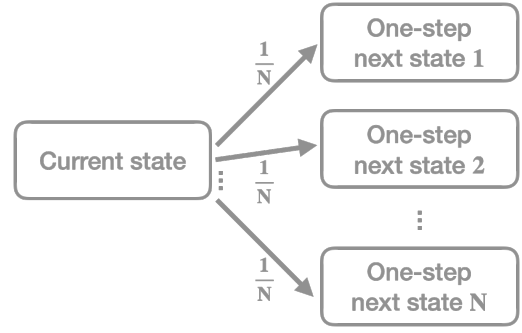


Figure 6: MultiVeStA's approach to path selection.

problem, where a model generates so many states that model checking cannot complete in a reasonable amount of time, or may fail due to high memory requirements. Indeed the design of models with large state spaces is not uncommon in the context of business process modelling. Therefore, in order to be able to provide a valid response in those cases, we resort to a simulation-based technique known as statistical model checking (SMC) [2, 45, 46]. This allows to analyse the model by running a finite number of finite executions, so to provide statistical evidence on the satisfaction or violation of a property. In other words, SMC allows to infer information on the entire state space while looking only to a reduced set of states at a time obtained via multiple simulations. An SMC analysis consists in performing independent simulations as long as a required level of statistical accuracy has not been reached.

The SMC technique we use is supported by MULTI-VESTA [61]. It enables us to easily associate a discrete uniform probability distribution to the states that can be reached in one step from a specific state, allowing for probabilistic simulations. The approach is depicted in Fig. 6: from the *current state* of the model we compute all (N) states reachable from it in one step. Each such state gets assigned the same probability ($\frac{1}{N}$) of being chosen as next state of a simulation. This is further discussed in Section 5.2. We note that this approach to path selection allows to introduce a form of fairness within our approach, which is not considered when using the MAUDE LTL model checker. In MULTIVESTA, any time a state is (re-)visited in a different simulation or during the same one (e.g., for the presence of a loop in the flow) the choice of the actual next state is done using the same probabilistic distribution. Therefore, the continuous occurring of this situation will hardly result in the process always executing the same path, so we can assume that in the presence of a choice which gets repeated, sooner or later all the paths will be explored. The fairness assumption is reasonable in the context of workflow management since all choices are made (implicitly or explicitly) by applications, humans or external actors. Indeed, we use a fair scheduler in order to resolve all forms of non-determinism and choose probabilistically the states to be considered as *next state* in each simulation (see, e.g., [9] for a similar approach in the MAUDE context). In general, SMC tends to be faster and

more scalable than model checking, with a price to pay in accuracy for not covering the entire state space of a system.

Result Report

Result Report makes possible to report the result of the verification. Model checking techniques usually generate counterexamples which witness that a given property does not hold. In this way, it can be shown whether a given model satisfies or not a certain property [6].

BProVe takes advantage of such counter-example generation by signalling to the end-user the model execution that falsifies the analysed property. BProVe enables formal reasoning at the level of BPMN model, so that diagnostic information and counter-example can be directly reported on the BPMN diagram in a way that is understandable by process stakeholders. This is especially useful when many parties interact on the basis of the models. BProVe differs from other approaches which typically provide counter-examples on low-level representations in third-party formalisms, thus hindering the interpretation of verification results at BPMN level; we provide examples of such approaches in Section 9.

Upon the interpretation of verification results, the designer can decide whether to restart or conclude the process.

4. Properties Definition

As stated in Section 3, BProVe enables the verification of properties over BPMN collaborations. To ease the definition of properties we have defined in MAUDE some recurrent predicates representing high-level BPMN-related concepts. In this section, we first describe the MAUDE predicates and how on top of them we define (i) LTL formulae to be verified with the MAUDE LTL Model Checker and (ii) MultiQuaTeX queries to be verified by the MULTIVESTA statistical model checker. Anyone can extend our approach by writing in MAUDE new predicates, formulae and queries.

4.1. MAUDE Predicates

The defined MAUDE predicates are evaluated over a single state of the model execution: in our case a configuration of the collaboration model. Here we introduce some MAUDE predicates we have used to define the LTL formulae to be verified with the MAUDE LTL Model Checker. In particular, we consider the predicates used in checking the running example against the properties in Tables 1-2 and the MultiQuaTeX queries in Listings 8-12.

processCompletion. It is used for the implementation of the Option to Complete property, to check whether it is possible to reach a configuration with tokens only on end events implying that the process of a specific organization (OrgName) completed the execution. The property is encoded as shown in Listing 3: processCompletion evaluates to true if at least one end event, in the considered state, has been enabled and no other token remains in the process, apart from end elements.

noProperCompletion. It is used for the implementation of the Proper Completion property, to check if the process of a specific organization (OrgName) completes with more than

```
ceq collaboration(
{CollAction1}
pool( OrgName ,
proc({Action1}ProcElements1 |
end( enabled , IENAME . IEToken )
),in: inputMsgSet ,out: outputMsgSet
) | Coll1
) |= processCompletion(OrgName) = true
if noTokenPresent(ProcElements1) = true .
```

Listing 3: Definition of the processCompletion predicate.

one token on the same end event. The property is encoded as shown in Listing 4: noProperCompletion evaluates to true if at least one end event has been enabled and it has more than one token. The first condition noTokenPresent(ProcElements1) evaluates to true if there is no token assigned to the set of process elements named ProcElements1. The second and third conditions, noMessagePending(outputMsgSet) $\neq 0$ and noMessagePending(inputMsgSet) $\neq 0$, evaluate to true whether a token related to a message exchange is still pending. The last condition OEToken > 0 evaluates to true whether a token is present on the considered end event; note that this end event has already a token assigned since its state is set to enabled. The predicate has been appropriately defined also for Message end and Terminate end events; for presentation purposes we report only the definition involving the simple end event.

```
ceq collaboration(
{CollAction1}
pool( OrgName ,
proc({Action1}ProcElements1 |
start( disabled , OENAME . OEToken ) |
end( enabled , OENAME . OEToken )
),in: inputMsgSet ,out: outputMsgSet
) | Coll1
) |= noProperCompletion(OrgName) = true
if noTokenPresent(ProcElements1) = true
/\ ( noMessagePending(outputMsgSet)  $\neq 0$ 
or noMessagePending(inputMsgSet)  $\neq 0$  ) /\ OEToken > 0.
```

Listing 4: Definition of the noProperCompletion predicate.

aTaskRunning. It is used for the implementation of the No Dead Activities property, which establishes that a given task can be set, at least once, in the status running (meaning that the task is currently being executed). If this property holds for all the tasks in the model, then the model does not have dead activities. The property is encoded as shown in Listing 5: aTaskRunning evaluates to true if a label running(TaskName) is produced.

```
eq collaboration(
{collab(OrgName,running(TaskName))}
Coll1
) |= aTaskRunning(TaskName) = true .
```

Listing 5: Definition of the aTaskRunning predicate.

safeState. It is used for the implementation of the safe-state property, which verifies that on each sequence edge

there is at most one token. The property is encoded as shown in Listing 6: `safeState` evaluates to true in states that satisfy the auxiliary function `noMultipleToken`, which checks that no more than one token is present on each sequence edge.

```
ceq collaboration(
  pool( OrgName ,
    proc( {Action1}ProcElements1 ),
    in: inputMsgSet ,out: outputMsgSet
  ) | Coll1
) |= safeState(OrgName) = true
if noMultipleToken(ProcElements1) = true .
```

Listing 6: Definition of the `safeState` predicate.

Besides soundness and safeness, we also consider ad-hoc properties for the given application scenario. The MAUDE predicates we use are: ***aTaskComplete***, satisfied if a specified Task in the Collaboration will always complete; ***abPool-SndMsg*** (resp. ***abPoolRcvMsg***), satisfied if a specified message is always sent (resp. received).

These predicates had to be slightly modified when performing statistical model checking. Essentially, as shown in Listing 7, we had to change the predicates so that they would evaluate to either 1.0 or 0.0 rather than true or false depending on whether the predicate is satisfied or not.

```
ceq rval( collaboration(
  {CollAction1}
  pool( Org1Name ,
    proc(
      {Action1}ProcElements1 | end( enabled , IEName . IEToken )
    ),in: inputMsgSet ,out: outputMsgSet
  ) | Coll1
  ) , processCompletion(Org1Name)
) = 1.0
if noTokenPresent(ProcElements1) = true .
```

Listing 7: Definition for MULTIVESTA of the `processCompletion` predicate.

4.2. LTL formulae

The above mentioned MAUDE predicates are used to define LTL formulae [55] that can be checked on BPMN specifications using our BPMN interpreter and the MAUDE LTL model checker [30]. The formulae we show here are obtained as a composition of the following basic operators:

- $\langle \rangle \phi$, where the operator $\langle \rangle$ (corresponding to the LTL operator F) is used to verify if a formula ϕ **eventually** holds. That is, in any possible execution path we always encounter a state where ϕ holds.
- $[] \phi$, where the operator $[]$ (corresponding to the LTL operator G) is used to verify if a formula ϕ **globally** holds. That is, ϕ holds in all states encountered in any possible execution path.
- $\phi \rightarrow \varphi$, where the operator \rightarrow is the standard boolean implication.

Table 1

LTL Model checker: soundness and safeness.

Property	LTL formula
Soundness (i): Option to Complete	$[] \text{processStart}(\text{orgName}) \rightarrow \langle \rangle \text{processCompletion}(\text{orgName})$
Soundness (ii): Proper Completion	$[] \sim \text{noProperCompletion}(\text{orgName})$
Soundness (iii): No Dead Activities	$[] \sim \text{aTaskRunning}(\text{taskName})$
Safeness	$[] \text{safeState}(\text{orgName})$

We discuss next the LTL formulae corresponding to the Soundness and Safeness properties as shown in Table 1.

Option To Complete. This property relies on predicate *processCompletion*. It checks whether it is possible to reach a configuration with tokens only on end events implying that the process of an organization completed its execution.

Proper Completion. This property requires that the process of an organization always correctly completes its execution. In particular, we check that we never reach a state that satisfies the predicate *noProperCompletion*, i.e. a state with an erroneous completion. This is stated in the corresponding formula in Table 1, which checks that the predicate does not hold in any state of any execution of the model.

No Dead Activities. This property relies on the predicate *aTaskRunning*, which evaluates to true if the considered task has status *running* in the current state. Note that the corresponding formula in Table 1 actually verifies the opposite condition: it checks that the predicate does not hold in any state of any execution of the model. Therefore, if this formula is not satisfied, then there exists at least a state in at least one execution where the considered task has status *running*. In other words, if the formula is false for a task, then that task is not a dead activity. Furthermore, this formula has to be evaluated for all tasks in the model: if the formula is false for all the tasks in the model, then the model does not have dead activities.

Safeness. This property relies on predicate *safeState* and checks if the property holds for all the states of each model execution.

Concerning the verification of application-dependent properties, some predefined properties in our approach are reported in Table 2. They allow to check whether:

1. having a task in the running state implies that the task will sooner or later complete (passing from the state running to complete);
2. the completion of a task implies the completion of another task highlighting a relation between those tasks;
3. the exchange of messages between different processes is happening correctly.

Anyone can use and combine the predicates implemented in MAUDE to express more complex LTL formulae.

4.3. MultiQuaTEx query

As later discussed in Section 5.2, MULTIVESTA requires queries to be evaluated to 1 or 0 (corresponding to true or false, respectively) in each simulation. Here we present queries used by MULTIVESTA based on MAUDE predicates from Section 4.1, modified in order to evaluate to 1 or 0 rather

Table 2
LTL Model checker: scenario-dependent properties.

	LTL formula
1	$\square (aTaskRunning("Confirm Booking") \rightarrow (\langle \rightarrow aTaskComplete("Confirm Booking") \rangle))$
2	$\square (aTaskComplete("Handle Payment") \rightarrow (\langle \rightarrow aTaskComplete("Confirm Payment") \rangle))$
3	$\square (aBPoolSndMsg("Customer", "Payment") \rightarrow (\langle \rightarrow aBPoolRcvMsg("Customer", "PaymentConfirmation") \rangle))$

than true or false, respectively. The query used in MULTIVESTA to check Option to Complete is given in Listing 8. This is written in MultiQuaTEx, MULTIVESTA's query specification language [61]. The upper part of the listing (Lines 1-8) defines a *response* operator.¹⁰ This evaluates to 1 in a simulation if in the first *nSteps* simulation steps, every time we first encounter a state where the state observation premise holds, then we also encounter a state where the state observation conclusion holds. In particular, the operator *Response* takes care of evaluating premise observations, while a second operator (*ObsAtStep*, Lines 10-17) is used to evaluate conclusion observations. Finally, Lines 19-20 state that we want to study property *Response* in the first *N* steps of simulation, using *processStart("OrgName")* as premise, and *processCompletion("OrgName")* as conclusion. The parameter *N* comes from the fact that MULTIVESTA considers *bounded properties*. In all experiments in this paper we have set *N* as 300 as we empirically found that it is adequate to consider 'most' of the behaviour of the models analysed in this paper. We empirically obtained value 300 by considering some of the largest models in the repositories and performing single simulations and SMC analysis iteratively by increasing the value of such parameter. We chose a value after which the results stabilized.

MULTIVESTA runs several simulations and keeps track of the results of the property evaluation. In the lower part of Listing 8 we ask MULTIVESTA to make an evaluation of the *processCompletion* property.

```

1 Response(nSteps,premise,conclusion) =
2 if { s.rval(premise) == 1 }
3 then # ObsAtStep({nSteps},{premise},{conclusion})
4 else if { ( s.rval("steps") >= nSteps ) }
5 then { 1 }
6 else # Response({nSteps},{premise},{conclusion})
7 fi
8 fi ;
9
10 ObsAtStep(nSteps,premise,conclusion) =
11 if { s.rval(conclusion) == 1 }
12 then # Response({nSteps},{premise},{conclusion})
13 else if { ( s.rval("steps") >= nSteps ) }
14 then { 0 }
15 else # ObsAtStep({nSteps},{premise},{conclusion})
16 fi
17 fi ;
18
19 eval E[ Response({N}, {"processStart("OrgName")"},
20 {"processCompletion("OrgName")"} ) ] ;

```

Listing 8: MultiQuaTEx definition of Option to Complete.

¹⁰For more details on this property pattern we refer, e.g., to <https://matthewbdwyer.github.io/psp/patterns/response.html>

The query used to check Proper Completion is given in Listing 9. The upper part of the listing specifies the operator *Reach*, a simpler version of the *ObsAtStep* one from Listing 8 which focuses on studying just the reachability of states satisfying a given property (*does observation obs evaluates to 1?*). In particular, *Reach* evaluates to 1 in a simulation if in any of the first *nSteps* it meets a state where *obs* evaluates to 1. Lines 2-3 check whether the condition is met in the current state. Otherwise, if the number of permitted simulation steps has been reached it evaluates to 0 (Lines 4-5), or it triggers the execution of a new step of simulation and re-evaluates *Reach* in the next simulation step (the *#* operator in Line 6 is a one-step next operator, which triggers the execution of a step of simulation). MULTIVESTA runs several simulations and keeps track of the results of the property evaluation. In the lower part of Listing 9 we ask MULTIVESTA to make an evaluation of the *noProperCompletion* property for simulations of maximum length of *N* steps.

```

Reach(nSteps,obs) =
if { s.rval(obs) == 1 }
then { 1 }
else if { ( s.rval("steps") >= nSteps ) }
then { 0 }
else # Reach({nSteps},{obs})
fi
fi ;

eval E[ Reach({N}, {"noProperCompletion("OrgName")"} ) ] ;

```

Listing 9: MultiQuaTEx definition of No Proper Completion.

The queries used to check No Dead Activities and Safety are implemented similarly to Listing 9.

Ad-hoc properties may be handled as well by MULTIVESTA. As an example, the queries from Table 2 can be expressed as shown in Listings 10, 11 and 12. The first query checks whether a Task identified by a name "TaskName" with status "running" will eventually reach the status "completed". The second query checks whether the completion of a Task identified by a name "Handle Payment" (with status "completed") implies the completion of another task named "Confirm Payment" (with status "completed"). The third query checks whether the sending of a message "Payment" from pool "Customer" implies the receiving of a message "Payment Confirmation" from the same pool. The three queries have a similar structure. For example, Listing 10 is defined as Listing 8 with the exception that Lines 19-20 state that we want to study property *Response* in the first *N* steps of simulation, using *aTaskRunning("TaskName")* as premise, and *aTaskComplete("TaskName")* as conclusion. This property corresponds to Property 1 of Table 2 because *aTaskRunning* and *aTaskComplete* alternate for a given task, meaning that a task cannot get twice or more times in status running (resp. complete) without getting in status complete (resp. running).

Listing 11 is similar to Listing 10, but we modify *Response*. We use a variable *premObs* which takes value 1 if we meet a state where the premise holds (Line 7), and is reset to 0 as soon as a state satisfying the conclusion is met (Line 5). In

```

1 Response(nSteps,premise,conclusion) =
2 if { s.rval(premise) == 1 }
3 then # ObsAtStep({nSteps},{premise},{conclusion})
4 else if { ( s.rval("steps") >= nSteps ) }
5 then { 1 }
6 else # Response({nSteps},{premise},{conclusion})
7 fi
8 fi ;
9
10 ObsAtStep(nSteps,premise,conclusion) =
11 if { s.rval(conclusion) == 1 }
12 then # Response({nSteps},{premise},{conclusion})
13 else if { ( s.rval("steps") >= nSteps ) }
14 then { 0 }
15 else # ObsAtStep({nSteps},{premise},{conclusion})
16 fi
17 fi ;
18
19 eval E[ Response({N}, {"aTaskRunning("Confirm Booking")"},
20 {"aTaskComplete("Confirm Booking")"} ) ] ;

```

Listing 10: MultiQuaTEx definition of Property 1 in Tab. 2.

all other states we just perform a new simulation step without modifying the variable Line 8). The evaluation for each simulation terminates as soon as we reach the required simulation steps (Lines 2-3). In particular, we return 1 - premObs, because we want to return 1 in all simulations in which all occurrences of the premise (aTaskComplete("Handle Payment")) are followed by an occurrence of the conclusion (aTaskComplete("Confirm Payment")).

```

1 Response(nSteps,premise,conclusion,premObs) =
2 if { ( s.rval("steps") >= nSteps ) }
3 then { 1.0 - premObs }
4 else if { s.rval(conclusion) == 1 }
5 then # Response({nSteps},{premise},{conclusion},{0})
6 else if { s.rval(premise) == 1 }
7 then # Response({nSteps},{premise},{conclusion},{1})
8 else # Response({nSteps},{premise},{conclusion},{premObs})
9 fi
10 fi ;
11 fi ;
12
13 eval E[ Response({N}, {"aTaskComplete("Handle Payment")"}, {"aTaskComplete("Confirm Payment")"}, {0}) ] ;

```

Listing 11: MultiQuaTEx definition of Property 2 in Tab. 2.

Listing 12 has same structure as Listing 10. However, we use aBPoolSndMsg("Customer", "Payment") as premise, while for conclusion we use aBPoolRcvMsg("Customer", "PaymentConfirmation") (see Lines 8-9). This corresponds to Property 3 of Table 2 because a message that has been sent (aBPoolSndMsg) should also be received (aBPoolRcvMsg).

```

1 Response(nSteps,premise,conclusion) =
2 Same as Listing 10.
3
4 ObsAtStep(nSteps,premise,conclusion) =
5 Same as Listing 10.
6
7 eval E[ Response({N},
8 {"aBPoolSndMsg("Customer", "Payment")"},
9 {"aBPoolRcvMsg("Customer", "Payment Confirmation")"} ) ] ;

```

Listing 12: MultiQuaTEx definition of Property 3 in Tab. 2.

5. Verification

The analysis techniques featured by BProVe are based on the state space exploration capabilities offered by our BPMN interpreter implemented in the MAUDE language.

In fact, given a state of the collaboration under analysis, the interpreter permits to compute its set of one-step next states, i.e. all states reachable from the given state in one execution step, by applying the rewriting rules defining the BPMN semantics (given in [15]). On the one hand this allows the MAUDE LTL model checker to derive and explore the whole state-space of the collaboration by considering all generated one-step next states. On the other hand it allows the MULTIVESTA statistical model checker to generate single execution runs in the form of probabilistic simulations by iteratively selecting in a probabilistic way a one-step next state.

In this section, we overview how the two model checking strategies exploited in BProVe use the information produced by the interpreter to perform the required analysis.

5.1. Verification with the MAUDE LTL Model Checker

As shown in Listing 13, the MAUDE LTL Model Checker takes as input the initial state of the modelled collaboration and an LTL formula. In particular, in this case initialState denotes the term shown in Listing 1, while the considered LTL formula checks the property Option to Complete on the process of the Travel Agency pool.

The MAUDE LTL model checker computes the required analysis by executing our interpreter starting from the given initial state, traversing if necessary the whole state space. The tool returns true if the property is satisfied, i.e. holds in all possible executions of the model. Otherwise it returns false and a counterexample. The counterexample consists in an execution (a sequence of states) that violates the property. As discussed later, other components in the BProVe tool-chain will parse such textual counterexample to map directly on the BPMN model the example of execution violating the property.

```

Maude> red modelCheck(initialState,
<> processCompletion( Travel Agency )) .
result Bool: true

```

Listing 13: The MAUDE encoding of an LTL verification request.

5.2. Verification with MULTIVESTA

From non-determinism to probabilistic simulations

Our BPMN semantics, and therefore our MAUDE interpreter, is inherently non-deterministic, meaning that different rules of the semantics might be applied to different enabled components of a BPMN specification, leading to a set of next-step states. As usual (see, e.g. [9, 10, 12, 3, 28] for some examples in the MAUDE context), in order to obtain probabilistic behaviours out of non-deterministic ones

we need to resolve this non-determinism. Two main approaches exist in the MAUDE context: (i) MAUDE specifications can be enriched with probabilities and quantities (obtaining *probabilistic rewrite theories* [3]) and schedulers are used to solve the remaining non-determinism (see e.g. [12, 3, 28, 8, 7]); (ii) or probabilistic strategy languages can be used to associate probabilities to rule applications [10, 9], computing for each state a probability distribution towards its one-step next states. Both approaches resolve non-determinism by probabilistic choices. Our proposal combines aspects of the two approaches as discussed in the following. Intuitively, with the first approach it shares the use of a Java scheduler that exploits our MAUDE implementation to generate all one-step next states of the current one, and then probabilistically selects one of them (as discussed for Fig. 6, if there are n one-step next states, we select each with probability $1/n$). Instead, from the second approach it takes the idea of labelling rule applications with probabilities, as it implicitly specifies the same probability to every rule application outgoing from the current state. We made this choice in order to follow a conservative approach, as we did not have to modify our MAUDE implementation of the BPMN semantics, but we added an external probabilistic scheduler to resolve non-determinism. Crucially, we decided to avoid to use the approach (i) where models have to be enriched explicitly with quantitative aspects (e.g. adding probabilities to the choices of a XOR gateway) because we are not aware of model repositories that contain BPMN models enriched with such quantitative aspects.

MULTIVESTA

Similarly to the MAUDE LTL model checker, MULTIVESTA takes in input the initial state of the collaboration of interest and a MultiQuaTEx query. When performing an analysis, MULTIVESTA interacts with the probabilistic version of our interpreter to run a number of simulations, and provides as a result the average of the evaluations computed for each simulation. We consider properties discussed in Section 4.3, corresponding to those used for the MAUDE LTL model checker. As discussed, each such property takes either value 1 or 0 in a simulation, depending on whether it is satisfied or not in that execution. The number of required simulations is automatically computed in order to give a result with a statistical confidence required by the user. In particular, MULTIVESTA estimates MultiQuaTEx queries according to a confidence interval (a, d) provided by the user. MULTIVESTA estimates the expected value of a MultiQuaTEx query as the mean \bar{x} of n samples (taken from n simulations), with n large enough (but minimal) to guarantee that the size of the $(1 - a) \cdot 100\%$ confidence interval is bounded by d . In other words, with statistical confidence of $(1 - a) \cdot 100\%$, the actual expected value x belongs to the interval $\left[\bar{x} - \frac{d}{2}, \bar{x} + \frac{d}{2}\right]$.

In order to present in a coherent way the results obtained using the MAUDE LTL model checker and MULTIVESTA, if MULTIVESTA returns exactly 1 we interpret it as *true*: the property is verified in all considered executions. Otherwise,

we interpret it as *false*: the property was not verified in at least one of the considered executions.

6. Result Report on the Scenario

As we know from Section 2, the Soundness property can be encoded, for both the LTL model checker and the statistical model checker, as the combination of the result for three sub-properties: Option to Complete, Proper Completion and No Dead Activities. The result for those sub-properties is combined using the logical AND operator to provide the result for the Soundness property. Notably, these sub-properties refer to a single process; however, the analyses are carried out on the whole collaboration model, so

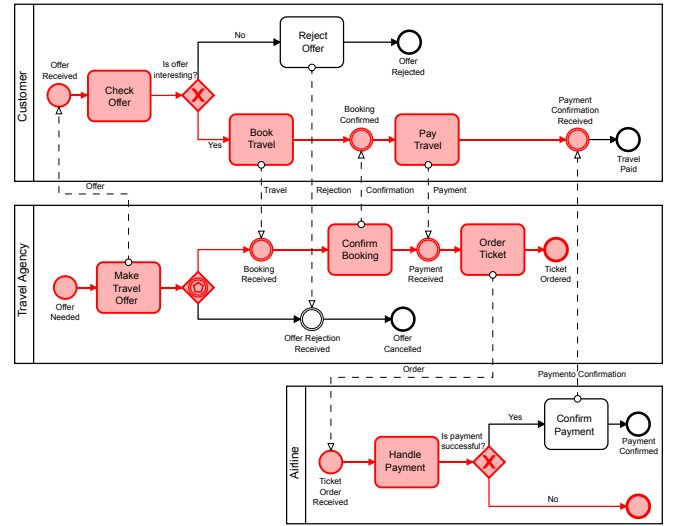


Figure 7: Automatic counter-example highlighting for our running example.

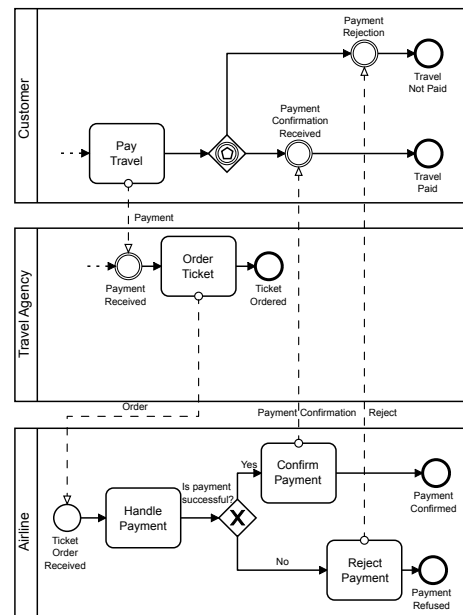


Figure 8: Counter-example guided fix of the model in Fig. 7.

Table 3

Soundness and Safeness analysis of the scenario in Section 2.1.

Property	Org/Task Name	LTL MC	MultiVeStA
Soundness (i) Option to Complete	<i>Whole model</i>	<i>F</i>	<i>F</i>
	Customer	F	0.72
	Travel Agency	T	1.0
	Airline	T	1.0
Soundness (ii) Proper Completion	<i>Whole model</i>	<i>T</i>	<i>T</i>
	Customer	T	0.0
	Travel Agency	T	0.0
	Airline	T	0.0
Soundness (iii) No Dead Activities	<i>Whole model</i>	<i>T</i>	<i>T</i>
	Make Travel Offer	F	1.0
	Check Offer	F	1.0
	Handle Payment	F	0.48
Safeness	<i>Whole model</i>	<i>T</i>	<i>T</i>
	Customer	T	1.0
	Travel Agency	T	1.0
	Airline	T	1.0

to consider the effects that message exchanges between processes may have on the single process (e.g., the non-reception of a message may cause a process to deadlock). Therefore, the sub-properties are checked for each process in the collaboration and the results obtained are combined through the logical AND operator to provide the result for the whole model.

Table 3 reports the analysis of Soundness and Safeness over our running example from Section 2.1. We can see that the property Option to Complete is not satisfied. This is because the property does not hold for the pool *Customer*, as the LTL model checker evaluates it to false. The counterexample generated by the LTL model checker is reported in Fig. 7. The model in the figure is the same as in Fig. 2; the only difference is that the executed tasks and the executed sequence flows are highlighted in red. As discussed in Section 7.2, this figure is automatically generated by the graphical component of our tool-chain. Focusing on the Customer pool (the top one), we can see that the Customer process is stuck waiting for the payment confirmation from the Airline pool (the bottom one). This message will never arrive because the Airline process refused the payment. This situation is something that must be avoided, especially if the model will be used for the enactment of such a process and possibly for the development of an application which will remain stuck waiting for a message.

Considering MULTIVESTA, we can see that it confirms the analysis of the Option to Complete property. In fact, it computes value 0.72 for the pool *Customer*. As discussed in Section 5.2, we can interpret this result as false. Considering Proper Completion, we can see that the LTL model checker evaluates all the corresponding formulae to true, therefore the model satisfies the property. This is confirmed by MULTIVESTA, which tells us that none of the considered simulations violates Proper Completion (as none satisfies noProperCompletion). As regards property No Dead Activities, we can see that the LTL analysis states that this property holds in the model. As discussed in Section 4.2, for each task we check that it can never be run in the model, and then we negate the obtained result. The results obtained for each task

Table 4

Properties of Tab. 2 and Listings 10-12 on scenario of Sec. 2.1.

Property	LTL MC	MultiVeStA
1	T	1.0
2	F	0.74
3	F	0.73

are then conjoined making No Dead Activities be evaluated to true. This is confirmed also by MULTIVESTA. When using MULTIVESTA we can check this property directly, obtaining 1.0 in every simulation where the considered task is executed at least once. Given that for all tasks we get a value greater than 0.0, we have that all tasks can be executed, therefore we don't have any dead activity.

Since the soundness property corresponds to combined values of Option to Complete, Proper Completion and No Dead Activities, we have that the considered model is unsound. In addition, we can see that the model is safe. In fact, the corresponding LTL formula is satisfied by each pool. Likewise, MULTIVESTA computes value 1.0 for each pool.

Table 4 reports results for the application-dependent properties of our scenario. Property 1 is evaluated to true by both the MAUDE LTL model checker and MULTIVESTA, implying that once the Confirmation Booking task is in the running state, it can always reach the completed state. As regards Property 2, we have that the MAUDE LTL Model Checker evaluates it to false. This is because the task Handle Payment can actually be executed without necessarily executing first the task Confirm Payment. This is confirmed by MULTIVESTA, which computes value 0.74. Finally, both the MAUDE LTL Model Checker and MULTIVESTA state that Property 3 is violated. This is because even if the Customer has sent the payment, it may happen that the corresponding confirmation will never be received.

Result interpretation and model fix

Once the analysis results have been interpreted, eventually discovered bugs should be fixed running again the model design activity, triggering a new iteration of the model development cycle in Fig. 3. Our analysis discovered that property Option to Complete is violated for the process of pool Customer. Fig. 8 depicts a new version of the model focusing only on the parts of the model that have been changed to fix the problem. The fix has been driven by the counterexample in Fig. 7. In the new model, after the payment the Customer waits for the payment management involving Travel Agency and Airline. This is represented by means of an event-based gateway. If the Airline rejects the offer, the Customer marks the travel as not paid as soon as the reject message is received. Otherwise, the payment is confirmed to the Customer. This new model successfully passes the verification of the Option to Complete property, as well as of the overall Soundness property. More in general, a new execution of Property Verification and Result Exploration activities confirms that the new version of the model satisfies the expected and desirable properties.

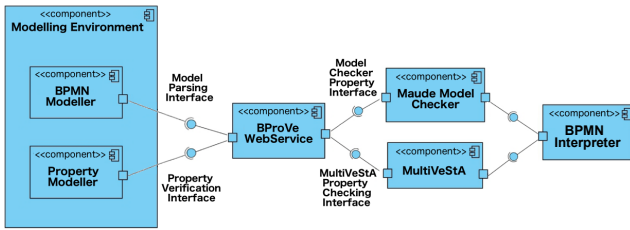


Figure 9: Component diagram of the BProVe tool-chain.

7. BProVe Tool-Chain

In this section we provide details on the architecture and usage of the BProVe verification tool-chain.

7.1. Architecture

BProVe is a tool-chain of open source software. For using the tool-chain we provide a web interface at <http://pros.unicam.it/bprove-web-interface/> as well as a Virtual-Box virtual machine containing a local installation of our tool-chain at <http://pros.unicam.it/bprove-testing-machine/>. Additional information about the tool-chain can be found at <http://pros.unicam.it/bprove>.

The overall architecture of the BProVe tool-chain is synthesised in the Component Diagram of Fig. 9. It is based on a standard client/server architecture concerning the provisioning of APIs for a BPMN verification service. Reading the figure from left to right we can see the following main components: Modelling Environment, BProVe WebService, MAUDE Model Checker, MULTIVeStA and BPMN Interpreter.

The Modelling Environment component allows to design BPMN models and to select or specify properties to be verified. In principle, any BPMN editor can be used as such component, especially those compliant with the BPMN 2.0 standard such as: bpmn-js¹¹, Eclipse BPMN2 Modeler¹², Camunda Modeler¹³ and Signavio Editor¹⁴. Currently, we offer two different deploys of the tool-chain using two different BPMN editors. The first one, based on bpmn-js, allows us to make BProVe available as a web application. The latter is distributed as an Eclipse plug-in and is based on the Eclipse BPMN2 Modeler. Both modelling environments have been augmented with the possibility to make calls to the BProVe WebService and to graphically interpret answers and counterexamples received from it. This de facto enriches the modelling environments with advanced analysis capabilities for designed models. In particular, we provide a menu to perform verification by selecting one of the two supported model checkers. BProVe offers a set of predefined properties that the user can select and verify without requiring any specific knowledge of the used analysis techniques. In addition, for the MAUDE LTL model checker we allow to specify LTL formulae describing application-dependent properties. In this case the user might use the predicates defined in the MAUDE interpreter discussed in Section 4.1.

¹¹<https://github.com/bpmn-io/bpmn-js>

¹²<https://www.eclipse.org/bpmn2-modeler>

¹³<https://camunda.com/products/modeler>

¹⁴<https://www.signavio.com>

The Modelling Environment interacts with the BProVe WebService via HTTP requests. In particular, a verification request has to include a BPMN model formatted according to the OMG standard and a property to be checked on the model. After the reception of the request, the service formats the BPMN model and the property into the syntax accepted by the back-end components, and then passes such data to the requested model checking components (MAUDE Model Checker or MULTIVeStA). The results of the model checkers are successively formatted in an XML file that is returned to the modelling environment which visualises the results in natural language. In case a counterexample is returned to signal a property violation, this is visualized directly on the model highlighting the erroneous path.

The remaining components offer the supported analysis techniques as discussed in Section 5. The MAUDE LTL Model Checker component consists of a running instance of MAUDE [14] loaded with the MAUDE modules implementing our BPMN interpreter¹⁵ and the LTL Model Checker embedded in MAUDE [30]. Instead the MULTIVeStA component consists of a running instance of the statistical analyzer MULTIVeStA, which interacts with the probabilistic version of our interpreter discussed in Section 5.2.

7.2. User Interface

The BProVe user interface aims at fostering the usage of formal verification for BPMN models also to non-experts of formal methods. In particular, after having designed or loaded a model in the modelling environment, the user can access the BProVe functionalities by just pushing a button we added to the toolbar. As a result, a menu that requires to select a model to parse is displayed. After the parsing request is sent to the BProVe WebService, a parsed model is returned. At this point, the user can request property verifications, and if he/she does it, the “BProVe Verification” menu pops up as shown in Fig. 10.

The menu enables to select one of the supported properties (or define a new one) according to different needs and expertise of the user. In particular, the menu includes three different sections permitting to specify properties in different ways: (i) general domain properties, (ii) application-dependent properties, and (iii) properties builder. The general domain properties section embeds a drop-down menu permitting to select properties that are generally desirable for any BPMN model. In particular, the user can ask for the verification of the *soundness* and *safeness* properties discussed in Section 4. Those properties are evaluated over the whole model. The application-dependent properties section embeds several drop-down menus which allow the user to select specific elements in the model (i.e., Pools, Tasks, and Messages) to check if specific conditions on them are satisfied. In particular, the “Verification of Task execution” allows to select a Task and to check whether it will ever reach the statuses “Enabled”, “Running” or “Completed”. It also allows to verify whether the execution of a Task implies the execution of another one. The “Verification of Message exchange” allows to

¹⁵<https://github.com/PROSLab/BPMNOS-Maude.git>

Welcome to the BProVe Web Interface!

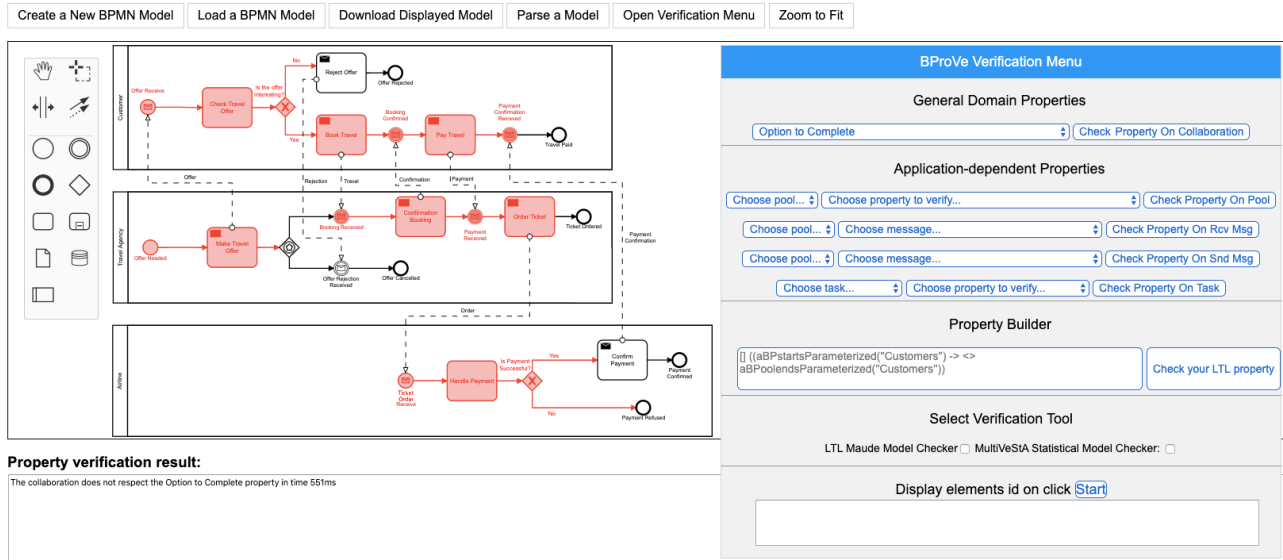


Figure 10: BProVe web interface.

select Pools and Messages and to check whether a specific Pool will send a specified Message, or to check whether a specific Pool will Receive a specified Message. The section named “property builder” is meant for highly skilled users able to write LTL formulae. In specifying the property, the user can make use of the LTL MAUDE operators and can take advantage of predefined subformulae (see Section 4).

The last section, named “MULTIVESTA Simulation”, allows to switch to another menu which includes the same sections previously described, but with the possibility of running property verifications with MULTIVESTA rather than with the MAUDE LTL Model Checker.¹⁶

Finally, verification results are reported in a text area where the user can also find the time, in milliseconds, needed to run the verification. In case the property is violated, the BProVe service returns a counter-example that can be visualised on the original model highlighting the model elements belonging to the corresponding execution path as shown in Fig. 10.

8. Experimental Evaluation

In this section, we discuss the validation we performed to assess the scalability and feasibility of the BProVe tool-chain, considering how the standard and the statistical model checking techniques complement each other. By *scalability* we mean the ability of BProVe to handle the verification of BPMN models with increasing size. This allows us to assess to what extent the tool can deal with large models and how its performance degrades as the size of models increases. By *feasibility*, instead, we mean the ability of BProVe to be applicable to real(istic) scenarios. This allows us to assess

¹⁶In the MULTIVESTA menu the “Property Builder” section has been removed, since it requires inserting LTL formulae specifically defined for LTL model checking.

whether the tool may be useful in practice.

The evaluation we have performed has been then structured over two main steps. The first one extensively validated the proposed verification approach by running a scalability analysis via ad-hoc designed and synthetically generated models. The rationale in using synthetic models was to do a scalability analysis on models on which we have full control and to show the tool capabilities in extreme scenarios.

The second one considered models retrieved from two freely accessible collections of models: the “BPM Academic Initiative Model Collection” and the “Camunda BPMN for Research”. At the time of writing, BPMAI and CAMUNDA are the most used open repositories of BPMN models to the best of our knowledge. Different practitioners and students have designed the models in the repositories during BPMN training sessions; therefore, they vary in size and usage of the BPMN notation. The rationale in using existing repositories of BPMN models is to assess the feasibility of BProVe, showing that it can also process real(istic) models designed by third-party, without knowing a priori the quality of the designed models.

Experiments have been performed on a dedicated machine running Ubuntu Linux 18.04.3, equipped with a processor Intel Core i7-6700HQ (2.60GHz), and 8 GB of RAM.

More details on the set-up and the results of the performed experiments are described in the following. For the LTL model checking analysis we used MAUDE 2.7.1¹⁷ loading our BPMN interpreter. For the statistical analysis we used MULTIVESTA, which interacted with the same version of MAUDE, and with the probabilistic version of our interpreter. MULTIVESTA requires a number of parameters,

¹⁷http://maude.cs.illinois.edu/w/index.php?title=Maude_download_and_installation

which we fixed for all models: we set a block-size of 30 in relation to how many simulations we ask to run before iteratively re-evaluating if the required statistical confidence has been reached. In case the desired confidence interval is not reached after performing a block of simulations, MULTIVESTA runs another block of simulations and repeats the checks. We set the a and d values describing the required confidence interval (see Section 5.2) to 0.1 and 0.15, respectively.¹⁸ We used the automatic parallelisation feature provided by MULTIVESTA [54]) using 3 cores of the machine to parallelise the simulations. Finally we defined a time limit of 600 seconds considering it as the maximum amount of time a user, adopting the development model presented in Section 3, may wait for receiving a response from the tool; we use this time limit to run our tests and to timeout the computations that exceed such limit.

8.1. Experiment Set-Up on Synthesised Models

To conduct a scalability analysis of our approach on models of increasingly large ‘size’, we have specifically designed some families of synthetic models. The need to devise such families comes from the fact that, although the typical notion of size for a BPMN model refers to the number of its elements, here we are more interested in the number of its execution states, which is what mainly affects the verification performance. The number of states of BPMN models may grow not only based on the growth of the number of elements but also on how they are structured. To cope with this aspect, we identified four ‘dimensions’ of growth for a BPMN model:

- *sequential growth*, where the number of sequential elements of the model increases;
- *nesting growth*, where the number of nested element blocks increases;
- *internal parallel growth*, where the number of parallel branches within an AND block increases;
- *external parallel growth*, where the number of processes within a collaboration increases.

Therefore, we defined the following families of models whose elements systematically increase along with one of these dimensions. In particular, for the sequential growth, we considered a family of process models with increasing sequential tasks. We considered a family of process models with three branches of exclusive split and joined gateways with varying nesting for the nesting growth. For internal parallel growth, we considered a family of process models with a block of parallel split and joined gateway with varying branches. Finally, we considered a family of collaboration

models with a pool sending and/or receiving a message with varying number of pools for the external parallel growth. Each family contains 6 models. In all the considered models we added a block of elements that will lead the process into a deadlock. As a consequence, all the designed models violate the Option to Complete and the No Dead Activities properties, and therefore they are not sound. The *error block* is reported in Fig. 11 and it has been added in a place that could make hard its identification when the model grows in size. This should permit us to identify possible limits of the different verification strategies. We made all the models available through the RePROSitory platform [16]¹⁹.

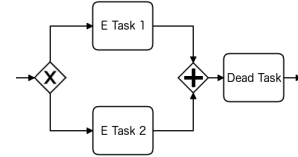


Figure 11: Error Block with Deadlock.

The first family, *Sequential Tasks*, is depicted in Fig. 12. The baseline model of the family consists of the sequential composition of a start event, a task, the *error block*, and end event. The other models are obtained by iteratively increasing by one the sequentially connected tasks.

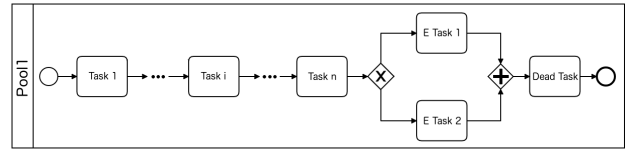


Figure 12: Model with sequential tasks.

The second family, *Nested Exclusive Branches*, is reported in Fig. 13. The baseline model contains a start event connected to an exclusive split gateway with three branches. Two of such branches are connected with a task per branch that merge into an exclusive join gateway, which is in turn connected with an end event. We say that those elements have a level of nesting equal to zero. The third branch is attached to the elements at the next level of nesting. At the highest level of nesting, level N in Fig. 13, we placed the error block from Fig. 11. We designed the remaining models by increasing level of nesting, where the elements in higher levels of nesting are structured in the same way as those in level 0.

The third family, *Parallel Branches*, is reported in Fig. 14. The baseline model is composed of a start event connected to a parallel split gateway with two branches and a single task per branch; the two branches successively merge into a parallel join gateway. At this point we placed the *error block*, then an end event that closes the model. We designed the remaining models by increasing the number of branches and tasks between the two parallel gateways.

¹⁸The values of 30 and 0.1 for the block-size and confidence a , respectively, have been chosen because are common in statistical estimations (see, e.g., [44]). The former value comes from assumptions of the law of large numbers, while the latter comes from the fact that modelers are often satisfied with 90% confidence intervals (alternatively 95% or 99% are also used). Instead, as regards d , we note that all properties are estimated as a value in the interval $[0, 1]$, therefore a *precision* of $\pm d/2 = 0.075$ is reasonable.

¹⁹https://pros.unicam.it:4200/guest/collection/fabrizio.fornari_bprove

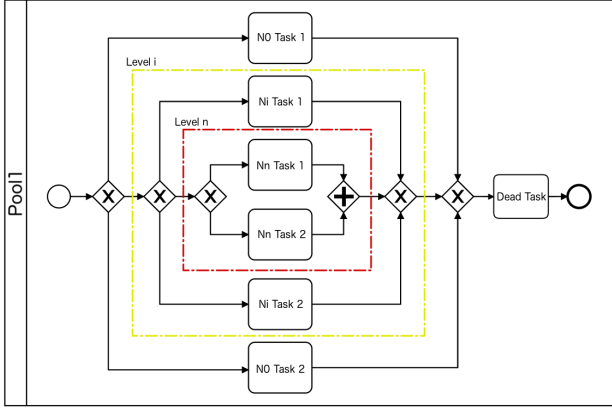


Figure 13: Model with nested exclusive branches.

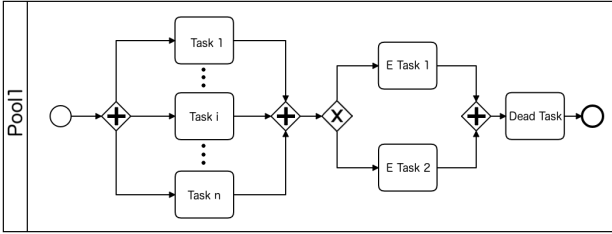


Figure 14: Model with parallel branches.

The fourth family, *Process Collaboration*, is depicted in Fig. 15. The baseline model is composed of three pools. The first one presents a start event, a send task, a receive task, the *error block* and an end event. The second one presents a start receiving message event, a send task and an end event. The third one presents a start event, a send task and an end event. The remaining models are obtained by inserting iteratively one copy of the intermediate pool before the last pool. In all models, the pools interact in sequence: the first pool sends a message to the second one, which sends a message to the third one, and so on, until getting to the n -th pool that sends a message back to the first pool.

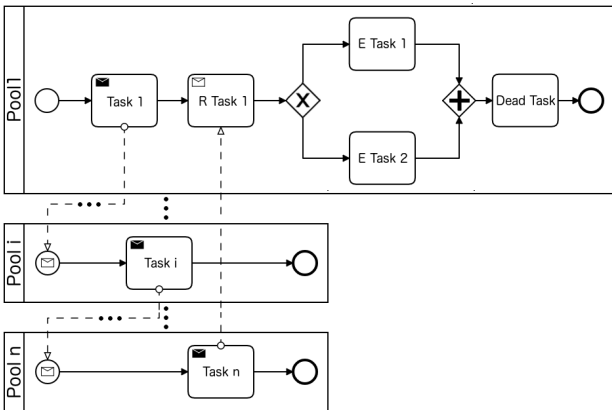


Figure 15: Model with multiple processes.

8.2. Results on Synthesised models

By analysing the synthetic models from Section 8.1 we performed a systematic analysis of the feasibility of our approach for both analysis techniques. In order to study how the analysis engines perform in terms of the *actual size* of the models, we used the state space generation capabilities offered by MAUDE [14] to compute the size of the state space of each model. This information, provided in the columns **states** in the tables presenting the analysis results, is crucial to properly study the scalability of the approach; in fact, the cost of verification depends on the size of the semantic model, which is due not only to the number of elements in the BPMN diagrams, but also to the degree of parallelism and non-determinism.

Table 5 reports the verification results, including running times in milliseconds, for our four families of synthetic models. To facilitate the comparison, for each experiment we highlight in bold the best performance between those of the two engines.

Sequential Tasks & Nested Exclusive Branches. The first two blocks of lines provide the verification results for the families *Sequential Tasks* and *Nested Exclusive Branches*, respectively. From a qualitative perspective, the two analysis engines always compute the same, correct, results.²⁰ Considering performances, for these two families the LTL MAUDE Model Checker consistently outperforms MULTIVESTA. This is because the selected models have very small state spaces, containing up to 83 states. In these cases, the LTL model checker is able to quickly explore all states and provide a response faster than MULTIVESTA. Exhaustive analysis techniques like LTL model checking are particularly well suited in these cases.

Parallel Branches & Process Collaboration. Moving our attention to the families *Parallel Branches* and *Process Collaboration*, shown in the third and fourth block of lines of Table 5, we note a complementary outcome. We can observe that the models of such families have much larger state spaces. In fact, an increase in the number of parallel branches or of pools leads to an exponential increase on the size of the state space, leading to the well-known state space explosion problem. In some cases, such a state space could not even be fully explored within our time limit of 600 seconds, forcing us to add the label T.O. (Time Out) in the corresponding entries. For these two families, the MAUDE LTL model checker succeeded in performing the verification of all the properties within the time limit of 600 seconds only on models with less than 6 tasks in parallel or 5 pools. We write T.O. (Time Out) in the entries corresponding to analysis that did not terminate in the imposed time limit and N.A. (Not Available) to indicate that a result is not available for that property verification. An exception is the Option To Complete property, which could be computed on

²⁰The verification community refers to model checking techniques that provide binary True/False outcomes, like Maude's LTL model checking, as *qualitative*, and to those with numeric outcomes, like MultiVeStA's SMC, as *quantitative*. See, e.g., one of the reference books in model checking [6].

Table 5

Experimental results over synthetic models with the MAUDE LTL model checker and MULTIVeSTA.

Model		Option to Complete				Proper Completion				No Dead Activities				Safeness			
ID	States	LTL MC		MULTIVeSTA		LTL MC		MULTIVeSTA		LTL MC		MULTIVeSTA		LTL MC		MULTIVeSTA	
		Res.	ms	Res.	ms	Res.	ms	Res.	ms	Res.	ms	Res.	ms	Res.	ms	Res.	ms
Experimental Results over <i>Sequential Tasks</i> models																	
t1	16	F	1	F	926	T	1	T	973	F	62	F	4448	T	2	T	973
t2	20	F	1	F	981	T	2	T	980	F	68	F	5757	T	2	T	1018
t3	24	F	2	F	1018	T	3	T	1041	F	63	F	7176	T	3	T	1022
t4	28	F	3	F	1036	T	3	T	1103	F	67	F	8779	T	4	T	1099
t5	32	F	3	F	1083	T	4	T	1174	F	64	F	10441	T	5	T	1159
t6	36	F	4	F	1141	T	5	T	1225	F	65	F	12116	T	6	T	1167
Experimental Results over <i>Nested Exclusive Branches</i> models																	
e1	12	F	1	F	880	T	1	T	844	F	65	F	2036	T	1	T	867
e2	27	F	3	F	1310	T	5	T	1000	F	68	F	5224	T	6	T	943
e3	41	F	7	F	1513	T	13	T	1050	F	74	F	7960	T	14	T	1022
e4	55	F	12	F	1178	T	24	T	1076	F	84	F	8981	T	26	T	1090
e5	69	F	18	F	1072	T	40	T	1123	F	99	F	5037	T	42	T	1129
e6	83	F	26	F	1123	T	61	T	1175	F	114	F	8712	T	64	T	1154
Experimental Results over <i>Parallel Branches</i> models																	
p1	54	F	4	F	1064	T	13	T	1072	F	73	F	5946	T	14	T	1121
p2	314	F	9	F	1200	T	176	T	1198	F	231	F	8641	T	172	T	1219
p3	2014	F	21	F	1397	T	2183	T	1396	F	2176	F	12932	T	2194	T	1416
p4	12514	F	44	F	1790	T	24659	T	1861	F	29266	F	19026	T	24773	T	1804
p5	T.O.	F	81	F	2436	T	257662	T	2359	F	104030	F	29868	T	260008	T	2229
p6	T.O.	F	143	F	3093	N.A.	T.O.	T	3177	N.A.	T.O.	F	45118	N.A.	T.O.	T	3255
Experimental Results over <i>Process Collaboration</i> models																	
c1	1520	F	4	F	1340	T	656	T	4202	F	474	F	8592	T	449	T	4148
c2	10025	F	6	F	1585	T	3513	T	6272	F	3284	F	10955	T	3470	T	6184
c3	62250	F	8	F	1807	T	25967	T	9191	F	25339	F	13649	T	26508	T	8925
c4	371875	F	10	F	2043	T	186592	T	12728	F	185538	F	16746	T	195246	T	12382
c5	T.O.	F	14	F	2306	N.A.	T.O.	T	16825	N.A.	T.O.	F	20294	N.A.	T.O.	T	16315
c6	T.O.	F	20	F	2612	N.A.	T.O.	T	21887	N.A.	T.O.	F	24057	N.A.	T.O.	T	21231

all models by the LTL model checker. For such a property the MAUDE LTL model checker is able to find a counterexample by considering only a small portion of the state-space, so we have a response on all models without ever generating their full state spaces. This is enabled by the fact that the model checker adopts an on-the-fly approach to state-space generation [34] where the state-space is generated at need while performing the analysis. Apart for this property, MULTIVeSTA offers better runtime for all models with more than about 10000 states, and computes the correct result also in the cases where the LTL model checker fails to complete in the time limit. Even if not shown in the table, MULTIVeSTA was able to analyse within the chosen time limit all properties up to the twentieth instance of each family.

8.3. Experiments on Models from Open Repositories

As a proof of concept, we also run a validation of the BProVe approach on models from two open repositories “BPM Academic Initiative Model Collection”²¹ and “Camunda BPMN for Research”²². We have chosen those repositories because they have been already successfully used for validation purposes elsewhere (e.g., [11, 68, 62, 43, 60]). In [17, 18] we used the former repository to evaluate a preliminary version of our approach and tool-chain, involving only LTL model checking.

We started from a set of 7639 models from the BPMAI repository and 3739 from the Camunda repository. From these sets we filtered out models that could hinder the validity of our experiment. Especially, we filtered out syntactically invalid models and those that presented issues in the usage of the BPMN syntax (e.g., elements drawn without a specified source or a target, or some sequence flows crossing pool boundaries). The sets have been reduced respectively to 6304 and 2979 models. Given that our focus is on collaboration models, we filtered out from the remaining models those with less than 5 elements (which are typically required to form a meaningful collaboration, i.e. a pool including a start, a message task and an end event, interacting with an empty pool) resulting in 5336 and 2961 models. We also removed models presenting disconnected elements (arguably they do not represent realistic models) ending up with 4379 models for the BPMAI repository and 2232 models for the Camunda repository. Out of the final filtered sets of models, we successfully processed 2544 and 713 models, respectively. The remaining models could not be handled by BProVe due to the presence of not supported elements (e.g., timer events, conditional events, and compensation events).

In conclusion, we analysed 3257 models which we classify in Table 6 according to the size of their state spaces. We analysed all such models using both our analysis engines. For each class of models, we write in column **Best Analysis Strategy** the one that best performed on average as discussed below. We note that the majority of the models were con-

²¹<https://bpmai.org/download/>

²²<https://github.com/camunda/bpmn-for-research>

Table 6

Classification of models from open repositories according to number of states and best performing analysis strategy.

Number of States	Number of Models				Best Analysis Strategy
	BPMAl		Camunda		
1-99	1838	72.2%	52	7.3%	LTL MC
100-999	406	16.0%	502	70.4%	LTL MC
1000-9999	135	5.3%	132	18.5%	LTL MC
10000-79000	37	1.5%	16	2.3%	MULTIVeStA
>79000	128	5.0%	11	1.5%	MULTIVeStA

Table 7

Percentages of the BPMAl and Camunda models satisfying the properties.

Property	BPMAl	Camunda
Soundness	79,75%	57,69%
- Option To Complete	83,13%	62,25%
- Proper Completion	96,55%	95,69%
- No Dead Activities	89,48%	68,58%
Safeness	95,49%	95,25%

centrated in the first three classes 1-99, 100-999 and 1000-9999, meaning that their state space did not pose a threat for the LTL model checker so making it a valid option for such models. The about 50 models with a state space of size between 10000 and 79000 could be analysed as well with the LTL model checker, but MULTIVeStA offered smaller running times. The about 140 models with state space greater than 79000 could not be analysed by the LTL model checker within the time limit of 600 seconds. We deepened our analysis by manually inspecting the models within class >79000. We found that all of such models, not surprisingly, presented a high degree of parallelism resulting from the use of such BPMN elements as AND gateways, OR gateways, and Pools with message exchange. Given the interleaving nature introduced by these elements on the resulting behaviour, models become difficult to handle using an LTL model checker which exhaustively explore the state space. The introduction of MULTIVeStA allowed us to provide a response also for such kind of models. We report in Table 7 the results for the properties verification over the analysed models of the two open repositories.

8.4. Highlights from the Experiments

The validation performed on the synthetic models highlights that MULTIVeStA offers better run-time for all models with more than about 10000 states (especially those presenting a high degree of parallelism), and computes the correct result also in the cases where the MAUDE LTL model checker fails to complete in a fixed time limit. The MAUDE LTL model checker instead performs better on sequential models, where the usage of an on-the-fly approach to state-space generation allows to find a counterexample by considering only a small portion of the state-space, so providing a response without ever generating the full state space of a model. The validation performed on the two open, widely used repositories confirmed the advantages of having complementary analysis approaches, so to enlarge the set of mod-

els for which it is possible to get a feedback. The results thus justify our choice of adding a statistical strategy to the analysis facilities of BProVe. Furthermore, as somehow expected, this suggests that there is not a *best* analysis strategy among the two, but that they should be selected depending on the characteristics of the considered models.

9. Related Work

The analysis of business processes is a largely investigated topic, and different approaches for the analysis of BPMN models are available in the literature (e.g., [49, 33, 32, 63]). In this section we split the research works related to our into two categories: *i*) those that map the model to another notation for which analysis techniques have already been consolidated, and *ii*) those that define a custom semantics for BPMN and that develop specific tools for the analysis.

9.1. BPMN Process Analysis via Mapping to other Formalisms

The most common formalisations of BPMN resorting to well-known formalisms are given through the definition of mappings into Petri Nets [22, 38, 42, 59, 5, 71, 51, 35]. The Petri Net resulting from the encoding of a BPMN model can serve as input to a Petri Net based verification tool. The ProM platform is probably the most used environment enabling such kind of strategy, since it can embed many components adopting an encoding-based approach to verification [23].

To analyse a BPMN model with ProM a user has to perform the following steps: 1. design a model with an external tool and export it in the BPMN format; 2. import the BPMN model into ProM, which then translates it into an internal PROM format; 3. choose one of the mappings available in ProM, and request the generation of the corresponding Petri Net; 4. select the resulting Petri Net, and then choose one of the analysis tools available in ProM (e.g. Woflan [66]) to analyse the Petri Net model; 5. read and interpret the responses from the analysis tool, which are referred to the Petri Net model, and interpret them back on the original BPMN model. Clearly, the most critical steps in this chain are 3, 4 and 5. **Step 3** is critical due to the possible absence of a rule in the chosen mapping for a specific BPMN element included in the model. In such a case the tool may miss to generate the Petri Net, or may generate a Petri Net that defines a behaviour not fully representative of the original model. Indeed, while for the basic BPMN modelling elements the encoding in Petri Nets is rather straightforward, for other elements such encodings could be cumbersome and quite challenging to define. For example, the management of *termination end events*, permitting to abort a running process, is usually not supported by such encodings. This is due to the inherent complexity of managing non-local propagation of tokens in Petri Nets. In addition, at the time of writing, no mapping tool available in ProM is able to properly translate BPMN Collaboration Diagrams due to the lack of mapping rules for message exchanges. Moreover, the existence of different mappings poses the problem of choosing

one among them for the analysis of a given BPMN model, or it may lead the user to make more tentatives using the various mappings, which may produce contrasting verification results. **Step 4** is critical since the possibility to define ad-hoc properties is often limited, only general properties are typically supported, and anyway when this is possible the properties will refer to the elements in the Petri Net and not to the original BPMN model. **Step 5** is critical since, as said above, in case of a property violation the result has to be re-conducted to the original BPMN model directly by the user.

Other approaches are available in the literature. The BPA Analyzer, defined as a module for the tool PromniCAT²³, supports analysis based on BPMN models reported in a specific JSON format. The models are translated into Petri Nets, and the LoLA analyzer is used for verifying properties expressed in CTL [29].

Corradini et al. supported business process verification transforming BPMN 1.2 models into CSP specifications [20]. The solution presents an Eclipse based tool-chain BP4PA²⁴ integrating the modelling environment with the PAT model checker [21]. The solution has been tested on simple business processes related to the Public Administration. Tools supporting the transformation from BPMN to YAWL Nets are also available. Ye and Son implemented an open-source plug-in called BPMN2YAWL that uses ILog BPMN Modeler as a graphical editor to create BPMN models, and implements transformation and verification as ProM 5.0 plug-in [72]. Both the modeler and the verification tool seem not anymore available.

Kheldoun et al. [39, 40] proposed an encoding of BPMN in Recursive ECATNets, expressed in terms of rewriting logic and analysed using the MAUDE LTL model checker. Even if we use the same model checker, the approach suffers from the mentioned encoding problems, and does not consider neither messages nor the event-based gateway. Moreover, the approach is tested on three simple examples only, without extensively validating it.

In summary, to the best of our knowledge, our proposal is the only one fully satisfying the relevant features we listed in Section 1, while all the proposals in this subsection share to some extent the issues described for the ProM based approaches.

9.2. BPMN Process Analysis via Direct Formalisations

Not many are the approaches that permit to directly conduct analysis activities over collaborations expressed with the BPMN notation. The approach that relates the most to our is the one proposed by Houhou S. et al. [37], and that is supported by a tool called fbpmn²⁵. The authors give a direct formalization in First-Order Logic, which is then implemented in TLA+ to enable formal verification using the TLC model checker from the TLA+ tool box. As it is the case of BProVe, also fbpmn allows to evaluate properties such as op-

tion to complete, proper completion, no dead activity, safety, soundness and message-relaxed soundness. In addition, it includes the possibility to select seven different communication semantics for message exchanges, feature that is not supported in BProVe. Nevertheless, our approach supports the user in the specification of ad-hoc properties based on the specific scenario of the BPMN model, and it allows to be extended to include new properties like the ones reported in Section 4. This is not possible in fbpmn.

We tested the fbpmn tool over models of the *Parallel Branches* and *Process Collaboration* families in Section 8.1, focusing on those with a high number of states for which the BProVe implementation with the LTL Model Checker could not provide an answer. However, we have to specify that the implementation of the BPMN semantics that stand at the basis of the two approaches are different, so performance can vary based on the amount of details reported in the implementation. For instance, with respect to fbpmn, in our semantics we have the possibility to express different statuses for tasks (enabled, running, completed, etc.) which allows us to express more detailed properties (e.g. implication between the execution of a task and another). This, on the other hand, contributes to increase the state space of a model. To provide a fair comparison between the two tools, we selected in fbpmn the Bag communication semantics, that corresponds to the one implemented in BProVe. In addition, since the implementation of the Soundness and Message Relaxed Soundness properties is slightly different between the two tools, we limited our experiments to the *Safeness* property. In Table 8 we reported the results for the verification of the *Safeness* property obtained with fbpmn and those obtained by the BProVe implementation based on MULTIVESTA. The tools provided the same result for the safeness property (in fact, all considered models are safe), but they required different amounts of time. BProVe resulted capable of handling models of the *Parallel Branches* family always in less time than fbpmn. For the *Process Collaboration* family, fbpmn is able to process models up to c12 in less time than BProVe, while BProVe starts outperforming fbpmn from model c13, thus demonstrating a better scalability. Indeed, by increasing the number of parallel branches in the former family, or the number of pools in the latter, the verification time of fbpmn increases significantly faster than that of BProVe.

When comparing the whole tool-chains, it is worth mentioning that fbpmn requires to perform several installation steps before actually being able to use it. In addition, it comes as a command line tool hardly usable by business process practitioners. Our tool-chain, instead, is immediately available to the final user, who can access it from the previously discussed web front-end. Moreover, all the components of our tool-chain are open-source, and a full installation is provided in a downloadable virtual machine, permitting to anyone to test every part of it without the need to follow complex installation procedures. Finally, fbpmn includes some low level technical constraints that hinder its usability; for example, the collaboration id must be equal to

²³<http://github.com/tobiashoppe/promnicat>

²⁴<http://bp4pa.sourceforge.net>

²⁵<https://github.com/pascalpoizat/fbpmn>

Table 8

BProVe MultiVeStA and fbpmn results for the Safeness property.

Model	BProVe MultiVeStA		fbpmn	
	Result	Time (ms)	Result	Time (ms)
<i>Parallel Branches models</i>				
p6	T	3255	T	3277
p7	T	4483	T	5550
p8	T	5648	T	12945
p9	T	7703	T	42340
<i>Process Collaboration models</i>				
c6	T	21231	T	3589
c7	T	26596	T	4759
c8	T	33029	T	7766
c9	T	36245	T	13636
c10	T	40995	T	10749
c11	T	43654	T	18830
c12	T	45091	T	32786
c13	T	50833	T	77262
c14	T	65120	T	156696

the name of the .bpmn file, which is not common especially with the Camunda modeler used in the fbpmn's tool-chain.

In [70, 69] the authors present a tool,²⁶ implemented in Haskell, that relies on a translation from a subset of well-formed BPMN 1.2 process diagrams to a CSP-like language based on the Z notation. The tool permits to check message-based properties, like consistency between BPMN diagrams with different levels of abstraction and compatibility between participants within a business process collaboration. The benefits of the proposed solution were illustrated only through simple scenarios, without an extensive validation like the one we conducted on BProVe using synthetic and real-word models. This tool, differently from BProVe, does not bring back the verification results directly on the analysed BPMN model. In addition, the tool usage is limited to models designed with the ILOG JViews BPMN Modeler which is not anymore available, making the comparison with this tool unfeasible.

9.3. Detailed Comparison with Other Approaches

Table 9 reports the list of all the BPMN elements that the various approaches are able to handle. The symbol X means that the element is correctly represented, X* means that the element is represented but it is mapped to its simplest variant (e.g., Start Receive Message Event is mapped and treated as a simple Start Event, User Task is mapped to a simple Task, etc.), while a cell left blank means that the element is not handled. Columns M0, M1, M2, M3 refer to four mappings supported by ProM. M0 (*Select BPMN Diagram*) is a plugin by H.M.W. Verbeek that allows ProM to convert a BPMN file into a BPMN Diagram and then apply other plugins over such representation; M1 (*Convert BPMN diagram to Petri net*), M2 (*Convert BPMN Diagram to Data Petri net*) and M3 (*Convert BPMN to Petrinet*) are plugins respectively by D. Fahland, A. Kalenkova and D. Fahland, and R. Con-

forti, that can be used to map BPMN Diagrams into Petri Nets. All the other approaches have been already discussed in Sections 9.1 and 9.2. Table 9 also highlights the subset of the supported BPMN notation. We notice that few elements are supported by all the approaches emphasising a core set of BPMN elements composed of None Start and None End events; Task activities; Parallel and Exclusive gateways; and Sequence flows. Instead, the less supported elements (that in our table are considered by less than four approaches) are: Signal Start and Conditional Start events; Terminate End events; Conditional, Signal and Escalation Intermediate events; User, Manual, Script, Service Rule Based Activities; Data Objects; Conditional Default and Association Flows. Overall, the table underlines the lack of a comprehensive approach supporting the whole list of elements. Our approach performs quite well with respect to BPMN elements covering 61% of the elements in the table, and only [37] performs better covering 68% of the elements.

Table 10 summarizes the main characteristics of the approaches relying on encodings from BPMN to another notation (under the column *Encodings*) and of the ones relying on the definition of a direct semantics for BPMN (under the column *Direct Semantics*). With the term *Predefined Properties* we refer to such properties that can be verified over the encoded model; *Custom Properties* refers to the possibility to verify model dependent properties and the possibility to encode new properties; *Tool Support* specifies which tool, if any, supports the approach (N.A. stands for Not Available); *Diagnostic* specifies how verification results are reported to the user (e.g. Graphical means that a violating trace is shown directly over the graphical representation of the analyzed model); *Necessary Background* refers to the formalism and tools required to use the approach, in the case of BProVe some knowledge on how to compose LTL formulae is required only for adding and verifying custom properties that the user may want to verify over a model; *Installation Process* indicates if a user has to install the tool by following an automatic procedure, a manual procedure (meaning that each component of the tool must be installed separately by the end user), or if he/she does not have to install it at all; *Automated verification* indicates whether the property verification is fully automatized, meaning that only few clicks are needed to request the property verification, or if the user has to perform some additional steps before achieving property verification.

For the two tables it is evident that there is not a single *best* approach. A user may chose the approach based on its main features (e.g., list of properties it can verify, possibility to encode his/her own properties, etc.), based on the BPMN elements used in the model, or even based on the usability of the supporting tool. In our case, a tool like BProVe that requires no installation for being exploited and that allows for a fully automated verification of properties, hiding the formalism which it relies on, could reasonably be considered easier to use compared to the other available tools. Especially, BProVe may be appealing for BPM practitioners who are not required to go through any manual installation

²⁶<http://www.cs.ox.ac.uk/peter.wong/bpmn>

Table 9

BPMN elements supported by approaches based on encodings or on direct semantics (X stands for a correctly represented element, X* for an element represented but mapped to its simplest variant, a blank cell for an unhandled element).

		Encodings								Direct Semantics		
Notation	Element	M0	M1	M2	M3	[29]	[20]	[72]	[39, 40]	[37]	[70, 69]	BProVe
Start Events	None	X	X	X	X	X	X	X	X	X	X	X
	Receive Message	X	X*	X*	X*	X*	X		X	X	X	X
	Conditional										X	
	Timer	X	X*	X*	X*	X*				X	X	
	Signal										X	
End Events	None	X	X	X	X	X	X	X	X	X	X	X
	Send Message	X	X*	X*	X*		X		X	X	X	X
	Terminate event									X	X	X
	Error	X	X*	X*	X*				X		X	
Intermediate Events	None Throwing	X	X	X		X						
	Message Send	X	X*	X*			X		X	X	X	X
	Message Receive	X	X*	X*			X	X		X	X	X
	Timer	X	X*	X*			X	X	X	X	X	
	Error (catch)						X	X	X			
	Error (throw)							X	X			
	Conditional										X	
	Signal							X				
	Escalation								X			
Activities	Task	X	X	X	X	X	X	X	X	X	X	X
	Send	X	X	X	X		X			X		X
	Receive	X	X	X	X		X			X		X
	User									X		X*
	Manual									X		X*
	Script									X		X*
	Service									X		X*
	Rule Based									X		X*
	Loop	X	X	X	X			X	X		X	
	Multiple Instance	X	X*	X*	X*			X	X		X	
	Subprocesses	X	X*	X*	X*	X		X*	X*	X	X*	X*
Gateways	Parallel	X	X	X	X	X	X	X	X	X	X	X
	Exclusive	X	X	X	X	X	X	X	X	X	X	X
	Inclusive (split)	X	X	X	X		X	X		X		X
	Inclusive (join)	X	X	X	X			X		X		X
	Event-based	X	X*	X*			X	X	X	X		X
Data	Data Object								X			
Flows	Sequence	X	X	X	X	X	X	X	X	X	X	X
	Conditional							X		X		X*
	Default							X		X		X*
	Message	X				X	X		X	X	X	X
	Association								X			
Swimlanes	Collaboration Pool	X					X			X	X	X

or configuration steps to require and interpret the verification of properties over their designed BPMN models.

10. Conclusions and Future Work

The wider adoption of BPMN to support the development of software systems asks for a clearer and rigorous interpretation of BPMN models, and in particular of the properties they satisfy. The adoption of formal verification contributes to striving model-driven software development, contributing to increase software quality. The literature discusses several approaches for BPMN verification, but available proposals generally miss to consider business process collaborations.

In this paper we propose the BProVe approach to support the entire model development cycle of BPMN collaboration models. BProVe relies on a direct formal semantics

for BPMN models, avoiding typical problems of approaches based on intermediate encodings. Within the BProVe approach, both standard model checking and statistical model checking techniques are integrated to effectively support verification. In particular, the use of statistical model checking allows to analyse business processes whose behavior generates large state spaces that are not easily manageable by classic exhaustive model checking techniques.

The BProVe approach has been instantiated in a complete web-based tool-chain, which makes transparent to the final user the inner formal layer. Our tool-chain overcomes the limitation of tools available in the literature, which in most of the cases are just prototypes mainly used for demonstration purposes. Indeed, some of them are not maintained anymore, becoming practically obsolete due to the development of new and incompatible versions of modelling environments and programming languages.

Table 10

Main features of the considered analysis approaches for BPMN models.

Approach	Encodings					Direct Semantics		
	[23]	[29]	[20]	[72]	[39, 40]	[37]	[70], [69]	BProVe
Predefined Properties	Option to Complete, Proper Completion, No Dead Activities	Soundness, Weak Soundness, RelaxedSoundness, Dead Transitions, Uncovered Transitions, Unbounded Places, Quasi Liveness, Liveness, Transitioncover, isBounded, isCyclic, isFreeChoice, isExtendedFreeChoice, isSNet, isTnet, isWorkflowNet	Coordination, Control, Sharing, Transparency	Soundness, Deadlock Freedom, No Dead Task, Proper Completion, No OR-join	Soundness, Proper Termination	Soundness, Message relaxed Soundness, Safeness	Absence, Universality, Existence, Bounded Existence, Order, Precedence, Response, Chain Precedence, Chain Response, Deadlock freedom	Soundness, Option to Complete, Proper Completion, No Dead Activities, Message Relaxed Soundness, Safeness
Custom Properties								Supported
Tool Support	Woflan Plugin	PromniCat Module	Eclipse Plugin	N.A.	N.A.	Standalone	N.A.	Web App, Eclipse Plugin, Standalone
Diagnostic	Textual	Textual	Textual & Graphical		Textual	Graphical	Textual	Textual & Graphical
Necessary Background	PN, ProM	PN	PAT	PN, ProM	LTL, Maude RECATNet	Haskell	CSP, Haskell	LTL (for custom properties)
Installation Process	Mandatory, Automatic	Mandatory, Manual	Mandatory, Automatic	Mandatory, Automatic	Mandatory, Manual	Mandatory, Manual	Mandatory, Manual	Not Mandatory
Automated Verification	Partial	Partial	Full	Partial	Partial	Partial	Partial	Full

Both the approach and the tool-chain have been extensively validated to assess their capabilities on realistic scenarios, and to possibly highlight scalability issues. Performed experiments confirmed the opportunity to offer an integrated verification approach providing complementary support both to non-deterministic and to statistical verification techniques.

In the future, we plan to continue bringing forward our program fostering the introduction of formal methods for modelling and verification of BPMN models. In particular, we intend to study and define structural metrics that could permit to establish a priori which verification techniques is the most suitable for analyzing a given BPMN model. For instance, a metric based on the degree of parallelism could suggest the adoption of the statistical approach.

Specification and verification of time constraints and resource allocation over BPMN models is certainly an interesting future extension. This will ask to revise the semantics, that currently does not include any information related to quantitative aspects, and to adapt mechanisms for quantitative analysis to the BPMN context.

References

- [1] van der Aalst, W.M., 2000. Workflow Verification: Finding Control-Flow Errors Using Petri-Net-Based Techniques, in: Business Process Management. Springer. volume 1806 of *LNCS*, pp. 161–183.
- [2] Agha, G., Palmskog, K., 2018. A Survey of Statistical Model Checking. *ACM Trans. Model. Comp. Simul.* 28, 6:1–6:39.
- [3] Agha, G.A., Meseguer, J., Sen, K., 2006. PMAude: Rewrite-based Specification Language for Probabilistic Object Systems, in: *QAPL*. Elsevier. volume 153(2) of *ENTCS*, pp. 213–239.
- [4] Aldazabal, A., Bailly, T., Nanclores, F., Sadovych, A., Hein, C., Ritter, T., 2008. Automated model driven development processes, in: *Model Driven Tool and Process Integration*, pp. 361 – 375.
- [5] Awad, A., Decker, G., Lohmann, N., 2010. Diagnosing and Repairing Data Anomalies in Process Models, in: *Business Process Management Workshops*, Springer. pp. 5–16.
- [6] Baier, C., Katoen, J.P., 2008. Principles of model checking. MIT press.
- [7] ter Beek, M.H., Legay, A., Lluch-Lafuente, A., Vandin, A., 2015. Statistical analysis of probabilistic models of software product lines with quantitative constraints, in: *Proceedings of the 19th International Conference on Software Product Line*, 2015, pp. 11–15.
- [8] ter Beek, M.H., Legay, A., Lluch-Lafuente, A., Vandin, A., 2016. Statistical Model Checking for Product Lines, in: *Leveraging Applications of Formal Methods, Verification and Validation: Foundational Techniques - 7th International Symposium*, pp. 114–133.
- [9] Belzner, L., De Nicola, R., Vandin, A., Wirsing, M., 2014. Reasoning (on) service component ensembles in rewriting logic, in: *Specification, Algebra, and Software*. Springer. volume 8373 of *LNCS*, pp. 188–211.
- [10] Bentea, L., Ölveczky, P.C., 2012. A Probabilistic Strategy Language for Probabilistic Rewrite Theories and Its Application to Cloud Computing, in: *WADT*, pp. 77–94.
- [11] Bergmann, R., Müller, G., 2018. Similarity-based retrieval and automatic adaptation of semantic workflows, in: *Synergies Between Knowledge Engineering and Software Engineering*. Springer. volume 626 of *AISC*, pp. 31–54.
- [12] Bruni, R., Corradini, A., Gadducci, F., Lluch-Lafuente, A., Vandin, A., 2015. Modelling and analyzing adaptive self-assembly strategies with maude. *Sci. Comput. Program.* 99, 75–94.
- [13] Campos, A.L., Oliveira, T., 2013. Software processes with bpmn: an empirical analysis, in: *International Conference on Product Focused Software Process Improvement*, Springer. pp. 338–341.
- [14] Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C., 2007. All about MAUDE - a high-performance logical framework: how to specify, program and verify systems in rewriting logic. volume 4350 of *LNCS*. Springer.
- [15] Corradini, F., Fornari, F., Polini, A., Re, B., Tiezzi, F., 2018. A formal

- approach to modeling and verification of business process collaborations. *Science of Computer Programming* 166, 35–70.
- [16] Corradini, F., Fornari, F., Polini, A., Re, B., Tiezzi, F., 2019. RePROSitory: a Repository Platform for Sharing Business Process models, in: *Proceedings of the Dissertation Award, Doctoral Consortium, and Demonstration Track at BPM 2019*, pp. 149–153.
- [17] Corradini, F., Fornari, F., Polini, A., Re, B., Tiezzi, F., Vandin, A., 2017a. BProVe: a formal verification framework for business process models, in: *Automated Software Engineering, IEEE Computer Society*. pp. 217–228.
- [18] Corradini, F., Fornari, F., Polini, A., Re, B., Tiezzi, F., Vandin, A., 2017b. BProVe: tool support for business process verification, in: *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, October 30 - November 03, 2017*, pp. 937–942.
- [19] Corradini, F., Morichetta, A., Muzi, C., Re, B., Tiezzi, F., 2021. Well-structuredness, safeness and soundness: A formal classification of bpmn collaborations. *Journal of Logical and Algebraic Methods in Programming* 119, 100630.
- [20] Corradini, F., Polini, A., Polzonetti, A., Re, B., 2010a. Business Processes Verification for e-Government Service Delivery. *Information Systems Management* 27, 293–308.
- [21] Corradini, F., Polzonetti, A., Re, B., Falcioni, D., 2010b. An ECLIPSE Plug-In for Formal Verification of BPMN Processes, in: *Communication Theory, Reliability, and Quality of Service, IEEE*. pp. 144–149.
- [22] Dijkman, R.M., Dumas, M., Ouyang, C., 2008. Semantics and analysis of business process models in BPMN. *Information and Software Technology* 50, 1281–1294.
- [23] van Dongen, B.F., de Medeiros, A.K.A., Verbeek, H.M.W., Weijters, A.J.M.M., van der Aalst, W.M.P., 2005. The ProM Framework: A New Era in Process Mining Tool Support, in: *Applications and Theory of Petri Nets. Springer. volume 3536 of LNCS*, pp. 444–454.
- [24] D’silva, V., Kroening, D., Weissenbacher, G., 2008. A survey of automated techniques for formal software verification. *Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27, 1165–1178.
- [25] Dumas, M., La Rosa, M., Mendling, J., Reijers, H.A., 2018. *Fundamentals of Business Process Management*. 2nd edition ed., Springer.
- [26] Dumas, M., Pfahl, D., 2016. Modeling software processes using bpmn: When and when not?, in: *Managing Software Process Evolution. Springer*, pp. 165–183.
- [27] Dumas, M., Rosa, M.L., Mendling, J., Maesalu, R., Reijers, H.A., Semenenko, N., 2012. Understanding business process models: The costs and benefits of structuredness, in: *Ralyté, J., Franch, X., Brinkkemper, S., Wrycza, S. (Eds.), Advanced Information Systems Engineering - 24th International Conference, CAiSE 2012, Gdansk, Poland, June 25-29, 2012. Proceedings, Springer*. pp. 31–46.
- [28] Eckhardt, J., Mühlbauer, T., AlTurki, M., Meseguer, J., Wirsing, M., 2012. Stable Availability under Denial of Service Attacks through Formal Patterns, in: *FASE 2012, Springer*. pp. 78–93.
- [29] Eid-Sabbagh, R.H., Hewelt, M., Weske, M., 2013. A Tool for Business Process Architecture Analysis, in: *Service-Oriented Computing. Springer. volume 8274, pp. 688–691*.
- [30] Eker, S., Meseguer, J., Sridharanarayanan, A., 2004. The Maude LTL model checker. *ENTCS* 71, 162–187.
- [31] El-Saber, N., Boronat, A., 2014. BPMN Formalization and Verification using Maude, in: *Workshop on Behaviour Modelling-Foundations and Applications, ACM*. pp. 1–12.
- [32] Fellman, M., Zasada, A., 2014. State-of-the-Art of Business Process Compliance Approaches: A Survey, in: *Information Systems*.
- [33] Groefsema, H., Bucur, D., 2013. A survey of formal business process verification: From soundness to variability, in: *Business Modeling and Software Design*, pp. 198–203.
- [34] Grumberg, O., Peled, D.A., Clarke, E., 1999. *Model checking*.
- [35] Heinze, T.S., Amme, W., Moser, S., 2018. Static analysis and process model transformation for an advanced business process to petri net mapping. *Software: Practice and Experience* 48, 161–195.
- [36] Hesenius, M., Usov, A., Rink, C., Schmidt, D., Gruhn, V., 2019. A flexible platform architecture for the dynamic composition of third-party-services, in: *Proceedings - 2019 IEEE International Conference on Software Architecture - Companion, ICSCA-C 2019*, pp. 210–217.
- [37] Houhou, S., Baair, S., Poizat, P., Quéinnec, P., 2019. A First-Order Logic Semantics for Communication-Parametric BPMN Collaborations, in: *Business Process Management - 17th International Conferences. Springer. volume 11675 of LNCS*, pp. 52–68.
- [38] Huai, W., Liu, X., Sun, H., 2010. Towards Trustworthy Composite Service Through Business Process Model Verification, in: *Ubiquitous Intelligence & Computing and Autonomic & Trusted Computing, IEEE*. pp. 422–427.
- [39] Kheldoun, A., Barkaoui, K., Ioualalen, M., 2015. Specification and Verification of Complex Business Processes - A High-Level Petri Net-Based Approach, in: *Business Process Management. Springer. volume 9253 of LNCS*, pp. 55–71.
- [40] Kheldoun, A., Barkaoui, K., Ioualalen, M., 2017. Formal verification of complex business processes based on high-level Petri nets. *Information Sciences* 385–386, 39–54.
- [41] Kiepuszewski, B., ter Hofstede, A.H.M., Bussler, C.J., 2000. On structured workflow modelling, in: *CAISE. Springer. volume 1789 of LNCS*, pp. 431–445.
- [42] Koniewski, R., Dzielinski, A., Amborski, K., 2006. Use of Petri Nets and Business Processes Management Notation in Modelling and Simulation of Multimodal Logistics Chains, in: *Modeling and Simulation*, pp. 28–31.
- [43] Lapeña, R., Font, J., Cetina, C., Pastor, Ó., 2018. Exploring New Directions in Traceability Link Recovery in Models: The Process Models Case, in: *CAISE. Springer. volume 10816 of LNCS*, pp. 359–373.
- [44] Law, A.M., Kelton, D.M., 2015. *Simulation Modeling and Analysis*. 5th ed., McGraw-Hill Higher Education, <http://www.averill-law.com/simulation-book/>.
- [45] Legay, A., Delahaye, B., Bensalem, S., 2010. Statistical Model Checking: An Overview, in: *Proceedings of the 1st International Conference on Runtime Verification. Springer. volume 6418 of LNCS*, pp. 122–135.
- [46] Legay, A., Lukina, A., Traonouez, L., Yang, J., Smolka, S.A., Grosu, R., 2019. Statistical Model Checking, in: *Computing and Software Science: State of the Art and Perspectives. Springer. volume 10000 of LNCS*, pp. 478–504.
- [47] Li, J., Jeffery, R., Fung, K.H., Zhu, L., Wang, Q., Zhang, H., Xu, X., 2012. A Business Process-Driven Approach for Requirements Dependency Analysis, in: *Business Process Management. Springer. volume 7481 of LNCS*, pp. 200–215.
- [48] Lindsay, A., Downs, D., Lunn, K., 2003. Business processes - attempts to find a definition. *Information and Software Technology* 45, 1015–1019.
- [49] Morimoto, S., 2008. A Survey of Formal Verification for Business Process Modeling, in: *Computational Science. Springer. volume 5102 of LNCS*, pp. 514–522.
- [50] Muehlen, M.Z., Recker, J., 2008. How Much Language Is Enough? Theoretical and Practical Use of the Business Process Modeling Notation, in: *Advanced Information Systems Engineering. Springer. volume 5074 of LNCS*, pp. 465–479.
- [51] Mutarraf, U., Barkaoui, K., Li, Z., Wu, N., Qu, T., 2018. Transformation of Business Process Model and Notation models onto Petri nets and their analysis. *Advances in Mechanical Engineering* 10, 1–21.
- [52] OMG, 2011. *Business Process Model and Notation (BPMN 2.0)*. URL: <https://www.omg.org/spec/BPMN/2.0/>.
- [53] Pastor, O., 2017. Model-Driven Development in Practice: From Requirements to Code, in: *Theory and Practice of Computer Science. Springer. volume 10139 of LNCS*, pp. 405–410.
- [54] Pianini, D., Sebastio, S., Vandin, A., 2014. Distributed statistical analysis of complex systems modeled through a chemical metaphor, in: *International Conference on High Performance Computing & Simulation*, pp. 416–423.
- [55] Pnueli, A., 1977. The temporal logic of programs, in: *Foundations of Computer Science, IEEE*. pp. 46–57.

- [56] Polyvyanyy, A., Bussler, C., 2013. The Structured Phase of Concurrency, in: *Seminal Contributions to Information Systems Engineering, 25 Years of CAiSE*. Springer, pp. 257–263.
- [57] Polyvyanyy, A., García-Bañuelos, L., Dumas, M., 2012. Structuring acyclic process models. *Information Systems* 37, 518–538.
- [58] Polyvyanyy, A., García-Bañuelos, L., Fahland, D., Weske, M., 2014. Maximal Structuring of Acyclic Process Models. *The Computer Journal* 57, 12–35.
- [59] Ramadan, M., Elmongui, H.G., Hassan, R., 2011. BPMN formalisation using coloured petri nets, in: *Software Engineering & Applications*, ACTA.
- [60] Schoknecht, A., Oberweis, A., 2017. LS3: Latent Semantic Analysis-based Similarity Search for Process Models. *Enterprise Modelling and Information Systems Architectures* 12, 2–1.
- [61] Sebastio, S., Vandin, A., 2013. MultiVeStA: Statistical model checking for discrete event simulators, in: *Performance Evaluation Methodologies and Tools*, ICST/ACM. pp. 310–315.
- [62] Skouradaki, M., Göerlach, K., Hahn, M., Leymann, F., 2015. Application of sub-graph isomorphism to extract reoccurring structures from BPMN 2.0 process models, in: *Service-Oriented System Engineering*, IEEE. pp. 11–20.
- [63] Sourì, A., Navimipour, N.J., Rahmani, A.M., 2018. Formal verification approaches and standards in the cloud computing: a comprehensive and systematic review. *Computer Standards & Interfaces* 58, 1–22.
- [64] Tan, W., Fan, Y., Ghoneim, A., Hossain, M.A., Dustdar, S., 2016. From the service-oriented architecture to the web api economy. *IEEE Internet Computing* 20, 64–68.
- [65] de Vasconcelos, A.M., de la Vara, J.L., Sanchez, J., Pastor, O., 2012. Towards CMMI-compliant Business Process-Driven Requirements Engineering, in: *Quality of Information and Communications Technology*, IEEE. pp. 193–198.
- [66] Verbeek, H.M.W., Basten, T., van der Aalst, W.M.P., 2001. Diagnosing Workflow Processes using Woflan. *Comput. J.* 44, 246–279.
- [67] Weske, M., 2012. *Business Process Management - Concepts, Languages, Architectures*, 2nd Edition. Springer.
- [68] Wiśniewski, P., Ligkeza, A., 2018. Constraint-Based Identification of Complex Gateway Structures in Business Process Models, in: *Artificial Intelligence and Soft Computing*. Springer. volume 10842 of *LNCS*, pp. 788–798.
- [69] Wong, P.Y., Gibbons, J., 2011. Formalisations and applications of BPMN. *Science of Computer Programming* 76, 633–650.
- [70] Wong, P.Y.H., Gibbons, J., 2008. A Process Semantics for BPMN, in: *Formal Methods and Software Engineering*. Springer. volume 5256 of *LNCS*, pp. 355–374.
- [71] Xiu, P., Zhao, W., Yang, J., 2017. Correctness Verification for Service-Based Business Processes, in: *Web Services*, IEEE. pp. 752–759.
- [72] Ye, J., Song, W., 2010. Transformation of BPMN diagrams to YAWL nets. *J. Softw.* 5, 396–404.

A. Full Specification of the Running Example in MAUDE

We report in Listing 14 the full MAUDE specification of our running example.

```
collaboration(
  pool( "Customer" ,
    proc( {emptyAction}
      startRcv(enabled , "SequenceFlow_11" . 0 , "Offer" .msg 0) |
      task( disabled , "SequenceFlow_11" . 0 , "SequenceFlow_12" . 0 , "Check Offer") |
      xorSplit( "SequenceFlow_12" . 0 , edges( "SequenceFlow_13" . 0 and "SequenceFlow_14" . 0 ) ) |
      taskSnd( disabled , "SequenceFlow_13" . 0 , "SequenceFlow_15" . 0 , "Rejection" .msg 0 , "Reject Offer") |
      end( "SequenceFlow_15" . 0 ) |
      taskSnd( disabled , "SequenceFlow_14" . 0 , "SequenceFlow_16" . 0 , "Travel" .msg 0 , "Book Travel") |
      interRcv( disabled , "SequenceFlow_16" . 0 , "SequenceFlow_17" . 0 , "Confirmation" .msg 0 ) |
      taskSnd( disabled , "SequenceFlow_17" . 0 , "SequenceFlow_18" . 0 , "Payment" .msg 0 , "Pay Travel") |
      interRcv( disabled , "SequenceFlow_18" . 0 , "SequenceFlow_19" . 0 , "Payment Confirmation" .msg 0 ) |
      end( "SequenceFlow_19" . 0 ) ,
      in: "Offer" .msg 0 andmsg "Confirmation" .msg 0 andmsg "Payment Confirmation" .msg 0 andmsg emptyMsgSet ,
      out: "Rejection" .msg 0 andmsg "Travel" .msg 0 andmsg "Payment" .msg 0 andmsg emptyMsgSet ) |
    pool( "Travel Agency" ,
      proc( {emptyAction}
        start( enabled , "SequenceFlow_21" . 0 ) |
        taskSnd( disabled , "SequenceFlow_21" . 0 , "SequenceFlow_22" . 0 , "Offer" .msg 0 , "Make Travel Offer") |
        eventSplit( "SequenceFlow_22" . 0 , eventRcvSplit(
          eventInterRcv( disabled , "SequenceFlow_23" . 0 , "Travel" .msg 0 ) |
          eventInterRcv( disabled , "SequenceFlow_24" . 0 , "Rejection" .msg 0 ) ) ) |
        taskSnd( disabled , "SequenceFlow_23" . 0 , "SequenceFlow_25" . 0 , "Confirmation" .msg 0 , "Confirm Booking") |
        interRcv( disabled , "SequenceFlow_25" . 0 , "SequenceFlow_26" . 0 , "Payment" .msg 0 ) |
        taskSnd( disabled , "SequenceFlow_26" . 0 , "SequenceFlow_27" . 0 , "Order" .msg 0 , "Order Ticket") |
        end( "SequenceFlow_27" . 0 ) | end( "SequenceFlow_24" . 0 ) ,
        in: "Travel" .msg 0 andmsg "Rejection" .msg 0 andmsg "Payment" .msg 0 andmsg emptyMsgSet ,
        out: "Offer" .msg 0 andmsg "Confirmation" .msg 0 andmsg "Order" .msg 0 andmsg emptyMsgSet ) |
      pool( "Airline Process" ,
        proc( {emptyAction}
          startRcv(enabled , "SequenceFlow_30" . 0 , "Order" .msg 0) |
          task( disabled , "SequenceFlow_30" . 0 , "SequenceFlow_31" . 0 , "Handle Payment") |
          xorSplit( "SequenceFlow_31" . 0 , edges( "SequenceFlow_32" . 0 and "SequenceFlow_33" . 0 ) ) |
          taskSnd( disabled , "SequenceFlow_32" . 0 , "SequenceFlow_34" . 0 , "Payment Confirmation" .msg 0 , "Confirm Payment") |
          end( "SequenceFlow_34" . 0 ) | end( "SequenceFlow_33" . 0 ) ,
          in: "Order" .msg 0 andmsg emptyMsgSet ,
          out: "Payment Confirmation" .msg 0 andmsg emptyMsgSet ) )
        )
      )
    )
  )
)
```

Listing 14: Full specification in MAUDE of the running example