Downloaded from orbit.dtu.dk on: Mar 19, 2024



DTU Library

BProVe: Tool support for business process verification

Corradini, Flavio; Fornari, Fabrizio; Polini, Andrea; Re, Barbara; Tiezzi, Francesco; Vandin, Andrea

Published in:
Proceedings of the 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)

Link to article, DOI:
10.1109/ASE.2017.8115708

Publication date:
2017

Document Version
Publisher's PDF, also known as Version of record

Link back to DTU Orbit

Citation (APA):
Corradini, F., Fornari, F., Polini, A., Re, B., Tiezzi, F., & Vandin, A. (2017). BProVe: Tool support for business process verification. In *Proceedings of the 2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)* (pp. 937-942). IEEE. https://doi.org/10.1109/ASE.2017.8115708

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

BProVe: Tool Support for Business Process Verification

Flavio Corradini, Fabrizio Fornari, Andrea Polini, Barbara Re, Francesco Tiezzi
University of Camerino, Italy
Email: {name.surname}@unicam.it

Andrea Vandin
DTU Compute, Denmark
Email: anvan@dtu.dk

***Abstract*—This demo introduces BProVe, a tool supporting automated verification of Business Process models. BProVe analysis is based on a formal operational semantics defined for the BPMN 2.0 modelling language, and is provided as a freely accessible service that uses open standard formats as input data. Furthermore a plug-in for the Eclipse platform has been developed making available a tool chain supporting users in modelling and visualising, in a friendly manner, the results of the verification. Finally we have conducted a validation through more than one thousand models, showing the effectiveness of our verification tool in practice.**

(Demo video: https://youtu.be/iF5OM7vKtDA)

***Index Terms*—Business Processes; BPMN; Structural Operational Semantics, MAUDE; Software Verification.**

I. Introduction

One of the most important factors impacting on the effectiveness of software engineering processes is a clear communication among domain and IT experts. Indeed, errors introduced during the design phase lead to more complex and costly problems in the subsequent phases of the development.

To improve communication effectiveness the notion of Business Process, originally introduced in the business management and organisation field, is gaining an ever-growing acceptance as modelling tool also in the software engineering field. A Business Process model generally describes a set of activities that an organisation should perform to fulfil a specific business goal [1]. In addition, models of this kind can be composed to form a so-called Collaboration model, used to describe the coordination of different organisations that cooperate to achieve shared goals. Currently, one of the most accepted notations for defining such models is the standard Business Process Modeling Notation (BPMN 2.0 – http://www.bpmn.org) maintained by the Object Management Group (OMG). Even more interestingly is probably the case of process-aware information systems [2], where defined BP models constitute the main input to set the functionality and behavior of the deployed software system.

Aiming at achieving high quality BPMN models and giving to domain experts and process designers an easy to use modelling and analysis environment, we have developed a novel tool for Business Process Verification (BProVe)[1]. It allows to automatically verify relevant properties for Business Process based software systems, like soundness and safeness.

Differently from similar tools, BProVe is based on a *native BPMN semantics* [3] defined according to a Structural Operational Semantics style [4], which makes it suitable for a MAUDE implementation. This approach has the great benefit of ensuring the faithfulness of the implemented semantics with respect to its theoretical definition, as operational rules of [3] are directly implemented as MAUDE rewriting rules.

A further distinctive characteristic of our tool is that it does not make any assumption on the structure of analysed business processes, thus supporting the verification of models with an *arbitrary topology*. We are aware that this may be not in line with recommendations to use structuredness in modelling [5]. However, in the real-world modelling practice most process designers do not always follow such a guideline, as witnessed by a study we carried out on the BPM Academic Initiative (http://bpmai.org/) repository resulting that more than 25% of models in the repository are not WS. On the other hand the restrictions imposed by structuredness are sometime considered too strict, instead by following an arbitrary topology designers are free to model the process according to the reality they feel [6]. In this way, the modelling activity is less complex [7] and more expressive [8]. Therefore we believe that considering models with an arbitrary topology is somehow important for a verification tool, and can help BProVe in having a real impact on the development of complex information systems. Nevertheless, considering a wider class of models allows BProVe to verify also structured processes.

BPMN 2.0 is a quite complex modelling notation including around one hundred graphical elements, even though some actually used in practice. BProVe permits to easily handle a wide set of BPMN elements, in particular those that can have non local effects, which are difficult to handle with tools transforming BPMN models to Petri Nets. For instance, the handling of *termination end events*, that permit to quickly abort a running process, is usually not supported due to the inherent complexity of managing non local propagation of tokens in Petri Nets. Instead this feature is natively supported by our semantics. Moreover, the main motivation to use Petri Nets encodings is the availability of already developed tools supporting verification [9]. However, such tools generally work well for standard Petri Nets, but they do not support extensions necessary to encode BPMN-specific features such as the

[1]BProVe is a free software; it can be redistributed and/or modified under the terms of the GPL2 License. BProVe source code, as well as instructions and software for installing its tool chain, can be found at http://pros.unicam.it/tools/bprove

978-1-5386-2684-9/17/$31.00 © 2017 IEEE

management of task state evolution (e.g., enabling, running and complete). BProVe instead is based on an extensible framework that is able to potentially support all the features of BPMN, as the approach permits to apply language extensions to cover new features without affecting the verification phase.

A further advantage of BProVe is its support for BPMN *Collaborations*. This enables the analysis of inter-organisational correctness, which is still not supported by most of the available tools [10]. Results of checking safeness and soundness over BPMN collaborations differ from results obtained through encodings, which usually introduce a mapping at process level and then compose the processes in a collaboration by means of an inner transition, often imposing an a priori upper bound on the number of pending messages [11].

Finally, due to its direct BPMN semantics, BProVe *reconducts verification results* to elements of the original model, so that diagnostic information can be reported on the diagram in a way that is understandable by process stakeholders. This is especially useful when many parties with different background need to quickly interact on the base of the models.

Overall, the mentioned aspects of BProVe have made possible the concrete verification of more than one thousand models coming from a public available repository supported by the BPM Academic Initiative.

The tool described in this paper relates to a verification framework extensively described in [12]. In particular with respect to [12] this paper provides additional details on the implementation of the tool, as well as on the envisioned users, and on how the tool could be introduced in a general modelling methodology.

II. Implementation Details

The architecture of the tool chain that we propose is synthesised in the Component Diagram of Fig. 1 and it represents a quite standard organisation of components to organise a verification as a service framework. From the left side of the figure to the right, we can see three main components: Modelling Environment, BProVe WebService, and BProVe Framework. The Modelling Environment component represents the tool used to design BPMN models and to specify which properties of the model the user wants to verify. This component, realised as an Eclipse plug-in extending the Eclipse BPMN2 Modeler (https://www.eclipse.org/bpmn2-modeler/), is connected to the BProVe WebService and interacts with it via HTTP requests. A BPMN model and a property to verify are automatically sent to the Web service, which handles the requests by parsing the BPMN model and the property into the syntax accepted by the BProVe Framework. In addition it also formats the verification results in an XML file that can be successively loaded by the graphical user interface, which visualises the violations of the properties in the model. Notably the use of a RESTful interface allows the BProVe framework to be integrated as a plug-in in different modelling environments. In particular, the result of parsing the property is an LTL formula. The BProVe Framework component is based on a running instance of MAUDE [13] loaded with the MAUDE modules

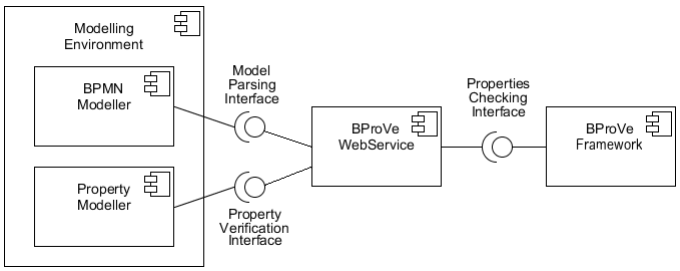


Fig. 1. Component Diagram of the BProVe Tool Chain.

implementing the BPMN semantics and the LTL MAUDE model checker [14].

III. Working Scenarios

In this section we provide an overview of the BProVe potentialities, discussing its impact on the typical phases of the lifecycle of a Business Process (BP) (see Fig. 2).

The starting point to exploit the BProVe functionalities is a BP model resulting from the **Design** done by means of a modelling environment. *Business Analysts*, who are business domain experts responsible for defining organisational BPs, together with *Process Designers*, who are responsible for modelling BPs by communicating with business domain experts and other stakeholders, collect domain requirements to produce BP models suitable to represent as-is or to-be scenarios in organisations. This can be done following different methodologies and using the Eclipse BPMN2 Modeller in the case of our tool chain. As soon as each model reaches a stable version, **Analysis** can be done to detect syntactic, structural and behavioral problems. At this level, the BProVe framework contributes underlining problems and providing feedbacks to the *Process Designer*, who iteratively improves the models by discussing with the *Business Analysts*.

In particular, the BProVe framework permits to analyse correctness of models with respect to domain independent properties such as `soundness` [15] and `safeness` [16]. Informally, `soundness` can be described as the combination of three basic characteristics concerning the dynamic behavior of a process model: (i) `Option to Complete`, requiring that a process instance should always complete, once started; (ii) `Proper Completion`, requiring that there exists no running or enabled activity for this instance when the process instance completes; (iii) `No Dead Activities`, requiring that a process model does not contain any dead activity, i.e., for each activity there exists at least one producible trace which contains the activity. Instead, `safeness` refers to the occurrence of no more than one token at the same time along the same sequence edge of a BP instance. Besides these general properties of BP models, BProVe also allows the analysis of application domain specific properties, defined by *Business Analysts* and codified by the *Process Designer*.

Once the models satisfy all the needed properties, further development activities, as well as BP **Enactment and Execution**, are enabled, and the models can be deployed in the underlying IT infrastructure. Successively the running process can

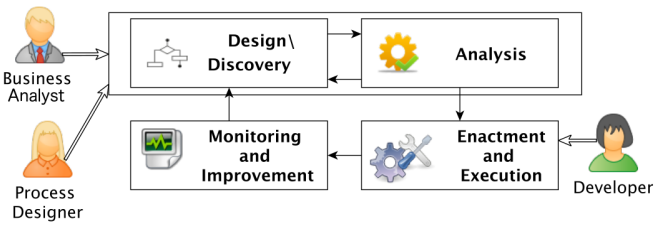


Fig. 2. BProVe Impacts on the Business Process Lifecycle.

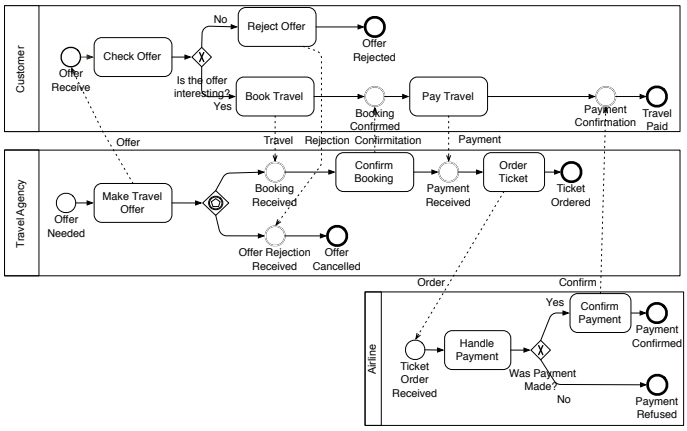


Fig. 3. Airline Collaboration Example (source [17, p. 115]).

be **monitored** and **improved**, and further verification activities can be performed in order to guarantee that modifications do not affect previously or newly specified properties.

IV. BProVe at Work on an Airline Scenario

This section shows BProVe at work on a BPMN collaboration which combines the activities of a Travel Agency, a Customer, and an Airline reservation system. The scenario is borrowed from [17], and it is represented in Fig. 3.

The collaboration allows these participants to interact in order to complete a commercial transaction related to the booking of a travel. After the Travel Agency makes a travel offer, it proposes such offer to the Customer, by sending an offer message. The Customer evaluates the offer, and takes a decision (represented in figure by a XOR gateway), according to which one of the two exclusive paths outgoing the gateway is activated. The upper path is activated in case of rejection: a rejection message is sent to the travel agency, and then both the Customer and the Travel Agency processes terminate. The lower path instead is activated if the Customer accepts the offer: the Customer books the travel by sending the travel message to the Travel Agency, and waits for the confirmation message. Once the travel is confirmed, the Customer pays the travel and sends the payment message to the Travel Agency. Finally, the Customer waits for the payment confirmation message before terminating. This message exchange is supported by the behaviors of the Travel Agency and the Airline. In particular, the Travel Agency, after the offer is sent, waits for the decision of the Customer. This is represented by means of an event-based gateway. If the customer rejects the offer, the Travel Agency cancels it as soon as the rejection message is received. On the other hand, if the customer accepts the offer, the Travel Agency receives the travel message and, hence, confirms the booking by sending the confirmation message. Then, as soon as the payment is received, before terminating the Travel Agency activates the airline to order the flight tickets, by sending the order message. The Airline handles the payment and, according to this, activates one of two exclusive paths by means of a XOR gateway. The upper path is activated if the payment is confirmed; this leads to the sending of the payment confirmation message to the Customer and, then, to the termination of the collaboration. The lower path is activated if the payment is not properly made; in this case the airline immediately terminates thus refusing the ticket order.

The user, after having designed the BPMN model with the Modelling Environment, requests the execution of the verification step (see Fig. 4, left-hand side). The Modelling Environment sends a request to the BProVe WebService asking for a parsing of the BPMN model. The BProVe WebService evaluates the model, verifying that it does not contain BPMN elements that cannot be handled by the core framework. In case of a negative answer, BProVe would inform the Modelling Environment, and hence the user, on the ineligibility of the model for parsing. In case of a positive answer, like for the Airline Scenario, the model is parsed and sent back to the Modelling Environment. The user is then requested to specify a property he/she wants to verify over the model, by just clicking on the button referring to the codified property. The Modelling Environment sends a verification request to the BProVe WebService, which transforms the property in an LTL formula compatible with the MAUDE LTL Model Checker [14], and it passes the property together with the model to the BProVe Framework. This latter component starts an instance of MAUDE loaded with the LTL MAUDE Model Checker and the MAUDE modules containing the semantic rules. Then, it verifies the property and sends the result back to the BProVe WebService, which handles the result, properly formatting it, and sends it to the Modelling Environment that will display it to the user. When the property verification result is negative, it means that the property is not satisfied by that BPMN model and a counterexample is reported to the user. This information is visualised directly within the BPMN model (see the elements colored in magenta in the screenshot in Fig. 4). This facilitates the interpretation of the verification result to users, especially for those unfamiliar with the underlying formal verification technique. Instead, if the property verification result is positive, it means that the property is satisfied and a message stating this is displayed.

TABLE I
Soundness and Safeness checking for the Airline scenario.

Property	LTL formula	Result
Soundness (i): Option to Complete	`[](aPoolCanStart(poolName) ⇒ <>aPoolEnds(poolName))`	False
Soundness (ii): Proper Completion	`[](aPoolEnds(poolName) ⇒ NoDandlingToken(poolName))`	True
Soundness (iii): No Dead Activities	`<> aTaskRunning(taskName)`	True
Safeness	`[] safeState(poolName)`	True

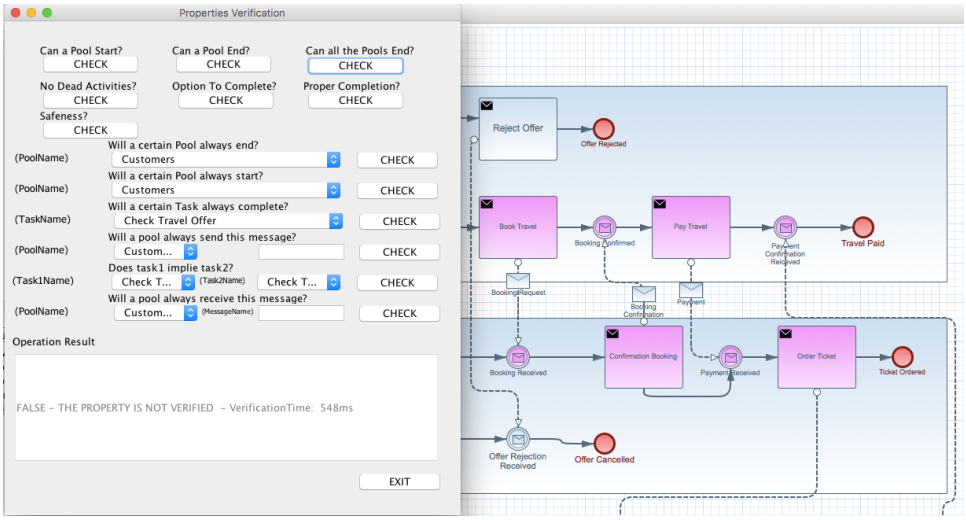


Fig. 4. BProVe User Interface - Counter Example Visualisation.

For the Airline scenario, the verification results shown in Table I report that the collaboration is not *sound*, but it is *safe*. In particular, `Option to Complete` is violated. More in detail, considering `Option to Complete` we may wonder if the collaboration involves processes that can start and then also end; if this is not the case, probably there is something wrong with the model. This property would hold if we would only focus on the Airline and Travel Agency processes, but it does not hold when we consider also the Customer, who can be stack waiting for a message from the Airline that will never arrive. We may also want to check if the collaboration has no pending token once the process has finished. To verify this, we can use `Proper Completion`. As it is not difficult to observe this property is satisfied since there is never the possibility that in a process there is more than one token. Finally, also the `No Dead Activities` property is satisfied, as all tasks are not dead, and for each of them there exists at least one execution trace that includes it.

Considering application domain properties, it is possibile to verify whether the completion of a specified task, say Handle Payment, implies the completion of another task, such as Confirm Payment. This property does not hold, since the former task can actually be executed without the implication of executing the latter task. The issues raised by the verification activity can be now easily fixed, as shown in the model in Fig. 5. In particular, in the new model, the Customer, after the payment, 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 all verification checks.

V. Experimental Evaluation

This section focuses on answering to a concrete research question: *Can the proposed tool chain efficiently and effectively be used in practice?* Our objective is mainly to assess the characteristics of the tool with respect to properties related

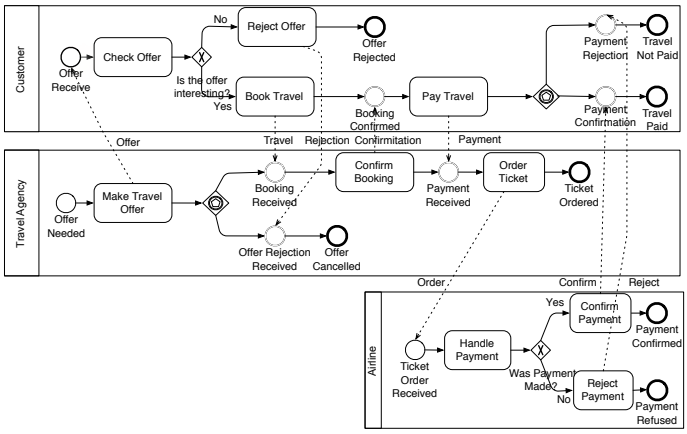


Fig. 5. Airline Collaboration Example - Issues Resolved.

to performance, while usability characteristics are not targeted here. More details on the evaluation can be found in [12]. To carry on the evaluation, we run BProVe using a collection of real-world BP models provided by the BPM Academic Initiative. The set-up and the results of the performed experiments are described below. The experiments can be replicated using a VirtualBox virtual machine containing an installation of our framework, available at http://pros.unicam.it/tools/bprove.

A. Experiment Set-Up

The BPM Academic Initiative repository consists of 16 032 BPMN models. However, we restricted to the latest revision of the models with 100% connectedness (this measure refers to the size of the largest connected sub-graph against the size of the overall model). A model without this level of connectedness includes disconnected fragments, which typically means that the model has not been finalised. Including such models in our validation would have resulted in verification data difficult to interpret. This gave us a dataset of 7 639 models with reasonable quality assurances. From these models we selected 1 245 models including more than 5 BPMN elements. This constraint is motivated since our focus is on collaboration, and 5 is the minimum number to have two pools exchanging

a message. Considering our reference dataset, we perform a preliminary transformation step from .json (the repository format) to .bpmn (the format we manage), and then we checked soundness and safeness.

The verification has been performed on the above mentioned Virtual Machine located into a beta version of the GARR cloud platform (https://cloud.garr.it/). The machine runs Ubuntu 16.04.2 LTS 64 bits and it has 4 VCPU, and 8 GB of RAM.

B. Experimental Results

Table II provides more insights on the complexity of the considered models, and on the time necessary to parse them. In particular, we classified the models in terms of the number of BPMN elements they contain (column *Class*). Column *AVG Elements* provides the average number of elements of models in the class, while column *Time* provides the average time in milliseconds necessary to parse each model in order to derive the format needed by MAUDE. As it can be observed, the parsing time slightly increases with model dimension.

TABLE II
COMPLEXITY OF THE CONSIDERED MODELS, AND PARSING TIME.

Class	*AVG Elements*	*Models*	*Time (ms)*
05–10	7	487	244
11–20	15	366	277
21–30	24	120	326
31–40	34	42	388
41–70	50	11	925

Table III provides more information on the complexity of the verification in terms of the time needed to check the considered properties. In particular, for each class of models the columns of the table report minimum, maximum, average, and median time as well as standard deviation. Time values are indicated as milliseconds needed to verify each property. It is worth mentioning that the values observed for class 41–70 are not fully significant given the small number of models belonging to such a class. Overall the observed data shows that properties can be verified in reasonable time and we were able to assess the properties in less than 15 minutes.

As it can be expected, verification time increases with the size of the considered model. There is also a high variability justified by the rather high values assumed by the standard deviations. This is mainly due to the fact that verification activities are particularly affected by the presence of notation elements leading to the interleaving of activities, such as parallel or pool statements, that are not always included in models. Nonetheless, the tool was able to provide an answer in reasonable time also for the most complex models in the repository. Finally, considering the median values, we discover that the used repository includes, for a large fraction, very simple models. The value of the median is indeed much smaller (up to 20 times) than the average. This tells us that most of the considered models are rather simple and that few models present real issues for checking. Nevertheless the conducted experiments confirmed the applicability of our approach.

TABLE III
RUNNING TIMES (IN MS.)

(a) Option to Complete.

Class	*Min*	*Max*	*Avg*	*Median*	*Std Dev.*
05–10	0	21 524	71	3	976
11–20	0	53 878	419	41	3 833
21–30	0	42 721	1 221	124	5 213
31–40	0	54 907	4 817	219	12 319
41–70	0	66 613	6 738	155	18 969

(b) Proper Completion.

Class	*Min*	*Max*	*Avg*	*Median*	*Std Dev.*
05–10	0	25 496	218	19	1 620
11–20	0	63 741	1 666	150	6 083
21–30	0	41 461	3 661	786	7 913
31–40	0	77 605	9 068	742	18 089
41–70	0	240 035	22 578	588	68 775

(c) No Dead Activities.

Class	*Min*	*Max*	*Avg*	*Median*	*Std Dev.*
05–10	0	5 802	61	2	295
11–20	0	523 742	5 473	225	39 921
21–30	0	477 232	13 647	1 554	52 296
31–40	0	877 553	59 765	1 860	159 123
41–70	0	303 632	29 555	606	86 744

(d) Safeness.

Class	*Min*	*Max*	*Avg*	*Median*	*Std Dev.*
05–10	0	25 569	207	5	1 364
11–20	0	587 756	9 913	127	68 316
21–30	0	519 712	25 752	937	103 418
31–40	0	685 198	42 505	1 157	130 045
41–70	0	144 513	15 256	890	40 985

VI. RELATED WORKS

Considering commercial and open source solutions, BPMN is currently supported by several tools. To get an idea of the state of the art, we made an analysis considering the modelling environment listed in the OMG website (www.bpmn.org) in order to check if they support formal BP verification. We observed that none of them support formal verification.

Successively we focused on tools mainly related to academic efforts. In particular: GrGen.NET (is.tm.tue.nl/staff/rdijkman/bpmn.html), BPMN Transform (is.ieis.tue.nl/staff/rdijkman/bpmn.html), BPMN to CSP (www.cs.ox.ac.uk/peter.wong/bpmn/), BPMN Checker (recatnets.cnam.fr), PromiCAT (github.com/tobiashoppe/promnicat/), BPMN to YAWL (is.tm.tue.nl/staff/rdijkman/bpmn.html), and BP4PA (bp4pa.sourceforge.net). GrGen.NET implements BPMN 2.0 formalisation using graph rewriting rules [18]. It focuses on the capabilities of GROOVE to automate orchestration and choreography engines verification. The benefits of the solution are illustrated by means of a simple scenario, which showcases the approach. Dijkman et al. present a transformation from BPMN 1.2 models to Petri Nets [16]. The Petri Nets resulting from the mapping serve as input to a generic Petri Net-based verification tool (e.g., ProM) for static analysis of the model. The main contribution is the transformation rather than verification, indeed the validation they ran only focused on the transformation of 13 models, without providing any detail on models correctness. Wong and Gibbons discuss a mapping from BPMN 1.2 to CSP in Haskell and propose a verification step using FDR [19]. Consistency and compatibility are the checked properties. Also in this case benefits of the solution

are illustrated by means of simple scenarios such as our Airline Reservation and Travel Agency. Kheldouna et al. implement a BPMN 2.0 Checker [20]. Starting from BPMN they transform the model in ECATNets by using ATLAS Transformation Language and Acceleo transformation tool. The authors illustrated the approach through three examples and no extensive validation is reported. PromniCAT supports analysis based on a customised transformation from BPMN to Petri Net and uses LoLA for property checking [21]. However, there is no evidence of its use on real scenarios, neither in terms of empirical investigation on the set of models it deals with. Huai et al. propose a mapping from BPMN 1.2 to Petri Net followed by an analysis step [22]. The mapping is supported by rules implemented as XSLT code. The verification takes a PNML file in input and implements reachability analysis. It has been applied to a request for day off. Ye and Son implement an open-source plug-in called BPMN2YAWL. It uses ILog BPMN Modeller as a graphical editor to create BPMN models and it implements a mapping to YAWL, in order to check models via ProM [23]. Decker et al. also consider BPMN as source language and YAWL as target language [24]. In this case the tool has been tested to a limited number of models. Finally, Corradini et al. supported BP verification transforming BPMN 1.2 into CSP model [25]. The solution presents an Eclipse based tool chain integrating the modelling environment with the PAT model checker [26]. The solution has been tested on simple BPs related to the Public Administration.

Summing up, the literature discusses several approaches for BP verification, but only some of them introduce a complete tool chain for BPMN 2.0. Most of them are prototypes coming from research projects resulting as rudimentary systems used just for demonstration purpose. Some of them are not maintained at all, becoming practically obsolete due to development of new and incompatible versions of modelling environments and programming languages. From the validation point of view all reviewed solutions use just few simple examples. Thus, none of them introduces an extensive validation.

VII. Concluding Remarks

In this paper, we have presented an easy-to-use tool chain for the verification of business processes modelled in BPMN, which we validated against more than one thousand models publicly available. The results of the validation are reported and discussed in detail. Validation proves practical benefits and effectiveness of the approach. In the future, we plan to extend our framework to other BPMN relevant characteristics, such as data management, time constraints and resource allocation, so to enable also a quantitative analysis for BPMN models.

References

[1] A. Lindsay, D. Downs, and K. Lunn, "Business processes attempts to find a definition," *Information and Software Technology*, vol. 45, no. 15, pp. 1015–1019, 2003.

[2] J. Li, R. Jeffery, K. H. Fung, L. Zhu, Q. Wang, H. Zhang, and X. Xu, "A Business Process-Driven Approach for Requirements Dependency Analysis," in *Business Process Management*, ser. LNCS. Springer, 2012, vol. 7481, pp. 200–215.

[3] F. Corradini, A. Polini, B. Re, and F. Tiezzi, "An operational semantics of BPMN collaboration," in *Formal Aspects of Component Software*, ser. LNCS, vol. 9539. Springer, 2015, pp. 161–180.

[4] G. D. Plotkin, "A structural approach to operational semantics," *J. Log. Algebr. Program.*, vol. 60, no. 61, pp. 17–139, 2004.

[5] R. Laue and J. Mendling, "The Impact of Structuredness on Error Probability of Process Models," in *Information Systems and e-Business Technologies*, ser. LNBIP. Springer, 2008, vol. 5, pp. 585–590.

[6] A. Polyvyanyy and C. Bussler, "The structured phase of concurrency," in *Seminal Contributions to Information Systems Engineering*. Springer, 2013, pp. 257–263.

[7] B. Kiepuszewski, A. H. M. ter Hofstede, and C. J. Bussler, "On structured workflow modelling," in *Advanced Information Systems Engineering*, ser. LNCS, vol. 1789. Springer, 2000, pp. 431–445.

[8] A. Polyvyanyy, L. Garcia-Banuelos, D. Fahland, and M. Weske, "Maximal Structuring of Acyclic Process Models," *The Computer Journal*, vol. 57, no. 1, pp. 12–35, 2014.

[9] S. Morimoto, "A Survey of Formal Verification for Business Process Modeling," in *Computational Science*, ser. LNCS. Springer, 2008, vol. 5102, pp. 514–522.

[10] R. Breu, S. Dustdar, J. Eder, C. Huemer, G. Kappel, J. Kopke, P. Langer, J. Mangler, J. Mendling, G. Neumann, S. Rinderle-Ma, S. Schulte, S. Sobernig, and B. Weber, "Towards Living Inter-organizational Processes," in *Business Informatics*. IEEE, 2013, pp. 363–366.

[11] W. M. van der Aalst, "Structural characterizations of sound workflow nets," *Computing Science Reports*, vol. 96, no. 23, pp. 18–22, 1996.

[12] F. Corradini, F. Fornari, A. Polini, B. Re, F. Tiezzi, and A. Vandin, "BProVe: a Formal Verification Framework for Business Process Models," in *32nd IEEE/ACM International Conference on Automated Software Engineering*, 2017.

[13] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott, *All about maude-a high-performance logical framework: how to spec., program and verify syst. in rewriting logic*. Springer, 2007.

[14] S. Eker, J. Meseguer, and A. Sridharanarayanan, "The Maude LTL model checker," *ENTCS*, vol. 71, pp. 162–187, 2004.

[15] M. T. Wynn, H. M. W. Verbeek, W. M. van der Aalst, A. H. ter Hofstede, and D. Edmond, "Business process verification-finally a reality!" *BPM Journal*, vol. 15, no. 1, pp. 74–92, 2009.

[16] R. M. Dijkman, M. Dumas, and C. Ouyang, "Semantics and analysis of business process models in BPMN," *Information and Software Technology*, vol. 50, no. 12, pp. 1281–1294, 2008.

[17] M. Dumas, M. La Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of business process management*. Springer, 2013.

[18] P. M. Kwantes, P. Van Gorp, J. Kleijn, and A. Rensink, "Towards Compliance Verification Between Global and Local Process Models," in *Graph Transformation*, 2015, vol. LNCS 9151, pp. 221–236.

[19] P. Y. H. Wong and J. Gibbons, "A Process Semantics for BPMN," in *Formal Methods and Software Engineering*, ser. LNCS. Springer, 2008, vol. 5256, pp. 355–374.

[20] A. Kheldoun, K. Barkaoui, and M. Ioualalen, "Formal verification of complex business processes based on high-level Petri nets," *Information Sciences*, vol. 385-386, pp. 39–54, 2017.

[21] R.-H. Eid-Sabbagh, M. Hewelt, and M. Weske, "A Tool for Business Process Architecture Analysis," in *Service-Oriented Computing*, ser. LNCS. Springer, 2013, vol. 8274, pp. 688–691.

[22] W. Huai, X. Liu, and H. Sun, "Towards Trustworthy Composite Service Through Business Process Model Verification," in *Ubiquitous Intelligence & Computing and Autonomic & Trusted Computing*. IEEE, 2010, pp. 422–427.

[23] J. Ye and W. Song, "Transformation of BPMN Diagrams to YAWL Nets," *Journal of Software*, vol. 5, no. 4, 2010.

[24] G. Decker, R. Dijkman, M. Dumas, and L. García-Bañuelos, "Transforming BPMN diagrams into YAWL nets," in *Business Process Management*, ser. LNCS. Springer, 2008, vol. 5240, pp. 386–389.

[25] F. Corradini, A. Polini, A. Polzonetti, and B. Re, "Business Processes Verification for e-Government Service Delivery," *Information Systems Management*, vol. 27, no. 4, pp. 293–308, Oct. 2010.

[26] F. Corradini, A. Polzonetti, B. Re, and D. Falcioni, "An eclipse plug-in for formal verification of BPMN processes," in *Communication Theory, Reliability, and Quality of Service*. IEEE, 2010, pp. 144–149.