ProVer: An SMT-based Approach for Process Verification

Souheib Baarir[1], Reda Bendraou[1], Hakan Metin[1], Yoann Laurent[2]

[1]Laboratoire d'Informatique de Paris 6, Paris, France
FirstName.LastName@lip6.fr
2 IQVIA, laurent.yoann@gmail.com

Abstract

Business processes are used to represent the enterprise's business and services it delivers. They are also used as a means to enforce customer's satisfaction and to create an added value to the company. It is then more than critical to seriously consider the design of such processes and to make sure that they are free of any kind of inconsistencies.

This paper highlights the issues with current approaches for process verification and proposes a new approach called ProVer. Three important design decisions will be motivated: 1) the use of UML AD as a process modeling language, 2) the formalization of the UML AD concepts for process verification as well as a well-identified set of properties in first-order logic (FOL) and 3) the use of SMT (Satisfiability Modulo Theories) as a mean to verify properties spanning different process's perspectives in a very optimal way. The originality of ProVer is the ability for non-experts to express properties on processes than span the control, data, time, and resource perspectives using the same tool.

Introduction

Processes, whatever the field (ex. software, military or healthcare), are everywhere. They represent the building block of any information system nowadays. Business processes are used to represent the enterprise's business and services it delivers. They are also used as a mean to enforce customer's satisfaction and to create an added value to the company. Software processes are critical as well since they represent the guaranty to respect development process's deadlines and to ensure a certain quality of the delivered software, which in some cases will end up being the company's information system itself. It is then more than critical to seriously consider the design of such processes and to make sure that they are free of any kind of inconsistencies.

Inconsistencies can range from very basic syntactical errors, to more complex issues such as deadlocks, inconsistent allocation of resources and time on process's activities or any proprietary business constraints that might be violated by potential process executions.

Some process models, depending on the business domain (ex. Healthcare or military), can be very complex and may contain more than 250 activities with very sophisticated control and data flows, resource and time constraints [Christov 14][Simidchieva 10]. Trying to analyze these processes and to verify them without the help of a tool would be unmanageable. In [Mendling 09], a study demonstrated that over 2000 inspected process models, 10% of these models where unsound. An equivalent study on SAP repository containing more than 600 complex process models demonstrated that more than 20% had flaws. [Mendling 06, 07]. Similarly, Gruhn et al. [Gruhn 07] collected 285 EPC (Even-Driven Process Chains) from different sources i.e., repositories, scientific papers, thesis, etc., and came to the conclusion that even though process models were syntactically correct, more than 38% of them were unsound. Finally, Vanhatalo et al. [Vanhatalo 07] analyzed more than 340 Business process models with a focus and the control flow aspect to realize that half of them were invalid i.e., contained deadlocks or unreachable activities.

More than 20 years after the introduction of business processes, it is quite surprising to realize that, despite the different approaches and tools provided by the literature for model verification, such ratio of errors still persist in the domain of process models. One possible reason could be the complexity of some process models and the fact that they integrate different perspectives and properties that need to be checked, each one requiring a specific tool or approach to validate it [Gruhn 06].

In the following we give a motivating example to explain our point.

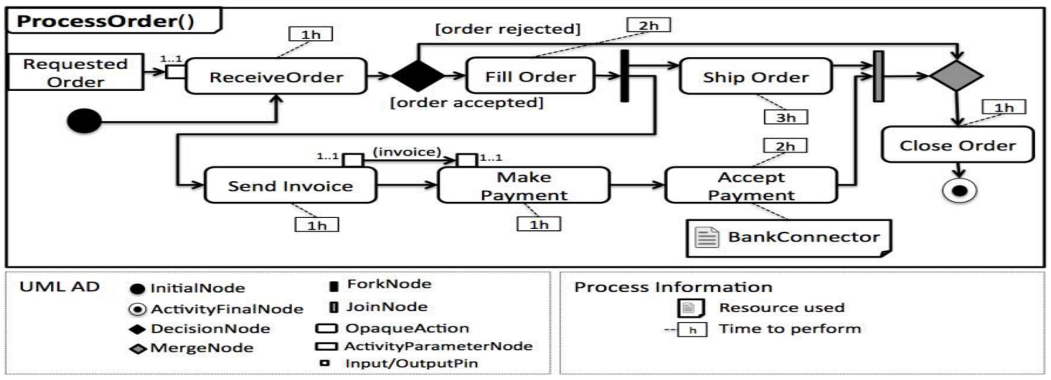


Figure 1. The *ProcessOrder* business process example extracted from the UML Specification.

Motivating Example. Process models are mainly driven by three perspectives: *Control flow, Data flow* and *Resources*. In addition, more complexity can be introduced by other constraints related to the "*Time*" perspective as well as to some project's specific business constraints (ex. some resources can be substitutable under some circumstances, some activities can be skipped under some conditions, etc.). Figure 1 presents an example which highlights process perspectives. The notation used is UML Activity diagrams. The goal of this process is to manage the reception, processing, payment and shipping of customers' orders. The process contains 7 actions connected to control nodes (*DecisionNode, ForkNode…*) and represents the control flow perspective. The data perspective is represented through the so-called *input/output Pins* (see notation details in the figure). They represent what an action might require to trigger its execution or what it might produce as the result of its execution. Some organizational aspects/constraints are annotated on the diagram such as the resources used by actions (ex. *BankConnector*) and time constraints on top of each action in terms of hours. Time units could also be used instead.

One way to validate such process models is to use formal verification techniques such as *Model-Checking*. In the field of business processes many approaches have been proposed for process verification [Eshuis 06, Wong 07, Liu 07, Trck 09, Fahland 09, Van der Aalst 11]. They address essentially what it is called soundness properties [Van der Aalst 11]. These properties guarantee the absence of deadlocks, unreachable activities, and other anomalies that can be detected without domain knowledge. However none of these approaches was really adopted in the industry [Morimoto 08]. Their complexity, the requirement of a mathematical background to use them and very often, a lack of a user-friendly tooling support was an obstacle for their adoption [Gruhn 06, Mendling 09]. But more specifically, let's illustrate what we believe to be an issue in the current approaches and the problems we intended to contribute to with our work. In the following we categorize the different issues we identified after a review of the state of the art in the domain of software and business process verification. A detailed review of the different approaches (more than 23) is given in [Laurent 18]

Identified issues in process verification. A major point with current process verification approaches is about ***the formalism and tools they rely on for performing the verification***. Whatever the process modeling language (PML), a formal semantics is given to the language by mapping its constructs to either variants of automata [Guelfi 05, Eshuis 06], Petri nets [Van der Aalst 98 & 11, Trcka 09, Fahland 09, Jung 10] or process algebra [Wong 07, Liu 07]. However, this means that we are relying on the semantics of the targeted formal language concepts in terms of expressiveness, e.g. Petri Nets, instead of the modeling language itself. For instance, if one wants to check the following property/constraint: "is the Close Order action always executed?", he can use classical Petri nets to represent the process model [Peterson 81]. However, verifying the property "is the invoice always produced after the execution of the Fill Order action?" (cf. Figure 1), implies using data from the system domain and imposes to use Colored Petri Nets (CPN)[Jensen 87]. Even if Petri nets, with their different variants, can represent anything defined in terms of an algorithm, this does not imply that the modeling effort is acceptable. Hofstede's and Wohed's paper [Hofstede 02] gives concrete examples of some Workflow Patterns that need very complex Petri nets extensions and tricks to represent them while this is expressed very naturally in UML Activity diagrams (AD) [Wohed 05] or Business Process Modeling Notation (BPMN) [OMG 11] which are among the modeling languages that support most of the workflow patterns as demonstrated in [Van der Aalst 03]. When it comes to the properties to be checked on process models, most of the approaches focus on the control flow aspect [Van der Aalst 11b], only a few of them address the data perspective [Awad 09, Trcka 09, Knuplesch 10] or the time perspective [Eshuis 02, Guelfi 05, Watahiki 11]. No approach proposes to address all these perspectives in a unified way. For instance, in the process depicted in figure 1, expressing a property such as "is it possible to reach the end of the process in less than 8 hours while making sure that the invoice artifact is produced and without using the BankConnector resource?" would be challenging.

Another obstacle for the limited adoption of formal techniques and approaches for process model verification in the industry comes from ***the process modelers' resistance to express process constraints and properties using formal languages*** [Emerson 90, Smith 02, Keleppe 03, Awad 07 & 08]. Due to the lack of mathematical background, properties are usually expressed in natural language and then given to experts that will translate these properties into a mathematical formalism with the eventual risk of misinterpreting what was initially required by the process modeler [Smith 02]. Other process modelers would prefer going through the process model and double checking it or by adding some exception handlers in the process model and extra checks in order to make sure that the process is sound. Of course, this is not feasible in the case of complex and sophisticated process models. These will definitely need a Model-Checking tool to ensure that the properties/constraints are satisfied whatever the process execution. LTL (Linear Temporal Logic) [Pnueli 77] or CTL (Computation Tree Logic)[Hafer 87] are usually used to express these properties.

Finally, current approaches ***and Model-Checking tools are not fully integrated to process modeling tools and the verification workflow is not always fully automated***. Some translation steps of the process models and of the properties to be checked towards the verification formalism have to be done manually in some approaches [Van der Aalst 99, Dijkman 08]. Moreover, usually the results of the verification in case of a counter-example is found is not explicitly reported graphically on the process model so that the process modeler can clearly identify the problem. Instead, the existing approaches propose a list of states representing the process execution that failed.

In conclusion, the issues we identified during a thorough study of the literature for process model verifications are: 1) the need to rely on the semantics of a target formal language which may limit the expressiveness of process models; 2) the inability of current approaches to support the different process perspectives (control flow, data flow, time and resources) in a unified way; 3) lack of a user-friendly tooling support which discourages the use of the current approaches and limits their adoption by the industry.

In the following section we draw the main lines of our approach for process verification called ProVer. Three important design decisions will be highlighted and motivated: 1) the use of UML AD as a process modeling language, 2) the formalization of the UML AD concepts for process verification as well as a well-identified set of properties in first-order logic (FOL) and 3) the use of SMT (Satisfiability Modulo Theories) as a mean to verify properties spanning different process's perspectives in a very optimal way. Section 3 gives more details about the role of SMT in ProVer while Section 4 will detail the tooling support and the validation aspects. Section 5 concludes this work and draws future steps of this contribution

ProVer: a framework for Process Verification

The traditional approach to achieve the verification of a model (a process model in our case) with respect to a given property consists beforehand in defining the two entities formally: a) the process modeling language concepts and b) the properties to be verified. A process model is then submitted to a so-called model-checker tool, which will answer the question of (un)satisfaction of the given property by the process model.

In the following sub-sections, we will detail our design choices and contributions regarding the ProVer framework.

UML Activity Diagram formalization for process modeling: our choice of UML as PML was mainly driven by the following arguments:

- UML is a standard and widespread modeling language in the industry with a mature tooling support.
- UML AD has proven to be a good candidate as PML in many works presented in the literature [Bendraou 10].
- UML AD has proven to be one of the most expressive languages in terms of satisfying the so well-known workflow patterns as presented by [Van Der Aalst 03].

However, the approach presented here can be applied to any PML such as BPMN but a formalization of BPMN concepts is required in order to achieve that. This represents the cost to adapt our approach to other PMLs.

One of the main challenges we had to face was that the semantics of UML AD is given in natural language in the standard specification which could be ambiguous and a source of misinterpretations. However, the OMG (Object Management Group) issued a new standard called fUML (Semantics of a Foundational Subset for Executable UML Models) that aims at giving a precise semantics to a subset of UML [OMG 11b]. The operational semantics of this subset are given in a pseudo java-code which is supposed to reduce ambiguity however; there are no mathematical and formal representations of this semantics that can be used straightforwardly as input to model-checking tools.

To face this issue, we decided to provide a formalization of UML AD semantics based on the fUML specification **instead of relying on a translation of UML AD towards other formalisms such as Petri nets for instance**. We aimed to define a formal model of fUML using First-Order Logic (FOL). The formalization addresses a subset of fUML that includes the set of concepts required for process modeling as identified in [Bendraou 05]. Current formalizations proposed in the literature focus mainly on the control-flow aspects of the process preventing to verify many kinds of properties related to data-flow, resources and timing constraints [Van der Aalst 11]. Therefore, our formalization covers both control and data-flow of the process through the use of the UML AD notations and takes into account organizational data such as resources and time constraints.

At this aim, we have formally reduced the representation of a software process to a vertex-labeled graph. Each graph's node corresponds to a UML Activity node according to its type (i.e. Control, Executable or Object Node). Each graph's arc corresponds to a UML Activity edge (i.e. Control or Object Flow). The execution semantics of this formalism is based on the notions of *states*, *enabling* and *firing* of transitions, similar to those used in the Colored Petri Nets [Jensen 87]. To be able to reason about each dimension of the process, the formalization covers both *control* and *data-flow* of the process through the use of the AD notations, and takes into account the associated organizational data such as *resources* and *timing* constraints. We partially published this formalization in [Laurent 14] and we extended it for the purpose of this work in order to cover more UML Concepts. Due to space constraints we can introduce it here but the interested reader can find it in [Laurent 18].

Once we had formalized the UML AD in FOL, we opted for an implementation based on Satisfiability Modulo Theories (SMT) technologies. SMT is an area of automated deduction that studies methods for checking the satisfiability of first-order formulas with respect to some logical theory T of interest [Barret 09]. What distinguishes SMT from general automated deduction is that the background theory T need not be finitely or even first-order axiomatizable, and that specialized inference methods are used for each theory. By being theory-specific and restricting their language to certain classes of formulas (such as, typically but not exclusively, quantifier-free formulas), these specialized methods can be implemented in solvers that are more efficient in practice than general-purpose theorem provers. Typical theories of interest include formalizations of various forms of arithmetic, arrays, finite sets, bit vectors, algebraic datatypes, strings, floating point numbers, equality with uninterpreted functions, and various combinations of these. These theories are supported by a standard called SMT-LIB [http://smtlib.cs.uiowa.edu/language.shtml] and are implemented in many efficient solvers (z3, Yices, CVC4, etc.) possible process's executions. They are related to the *Time* and *Resource* perspectives of a process. Examples of such properties are to make sure, for instance, that the process or an activity will terminate before a given deadline whatever the execution path, make sure that there will be enough agents to perform the activities of the process, etc. A detailed identification and formalization of such properties are then required in order to verify them on process models and to integrate them into our process verification tool.

Category	Definition
(1) Syntactical	
SynWorkflow	Syntactical errors on the process (*e.g. the source and target of an edge are different*)
SynOrganizational	Syntactical errors on the organizational part of the process (*e.g. the same agent cannot be assigned more than one time to the same activity*)
(2) Soundness	
OptionToComplete	A started process can always complete
ProperCompletion	No other activity should be running when the process terminates
NoDeadTransition	All the activities must be reachable
Soundness with data	
MissingData	Data is always present when needed (*e.g. no data missing to start an activity*)
UselessData	Data created is always used (*e.g. no data created but never used before the process ends*)
InconsistentData	Data can never be in an inconsistent state (*e.g. no data modified by multiple activities in parallel*)
(3) Organizational	
InTime	There is enough time to perform the activities (*e.g. the process will terminate before X hours/days*)
MissingResource	No missing resource to start an activity (*e.g. there are enough agents to do the process*)
InefficientResourceUse	No resources that are inefficiently used (*e.g. the agents have always activity to do*)
(4) Business	
ExistenceActivity	A is executed more / less / (between) X (and Y) times
ExistenceTimeActivity	A is executed before / after / (between) X (and Y) time unit
ExistenceTimeData	ArtefactA is available before / after/(between) X (and Y) time unit
ExistenceTimeResource	ResourceA is used before / after/(between) X (and Y) time unit
Relation	A is executed before / after / in-parallel / in-exclusion / (between) B (and C)
RelationData	ArtefactA is available before /after / in-exclusion of ArtefactB
RelationActivityData	ArtefactA is available before / after/in-parallel / in-exclusion / (between) the execution of B (and C)
LogicBased	e.g. Existence(A) implies Existence(B) else Existence(C)
….	e.g. Existence(A) implies (ExistenceData(ArtefactA) and ExistenceData(ArtefactB))

Table 1. Overview of the software properties we identified

***Categorization of process perspectives and properties*:** as mentioned earlier, the literature addresses essentially what it is called soundness properties [Van der Aalst 11] which aim to detect some *Behavioral* issues on process executions *vs. Syntactical* errors. Business processes and more particularly software processes are concerned with additional and critical constraints related to their human-oriented nature. They imply many creative tasks that rely on many factors such as time, human agents and resource management. The success of a software process depends also on the application of many best practices and organizational constraints. We call these constraints *Organizational* properties and we consider them as a subcategory of *Behavioral* properties since a state space exploration is required to guarantee their preservation for all

Once we formalized the UML AD, we also formalized a set of process properties that we identified through a detailed study of the literature. This set addresses the four process aspects introduced earlier (*Control* and *Data* flow, *Time* and *Resources*) plus another one that we called *Business* properties which refers to every project's specific constraints that have to be defined by the process modelers depending on the project's context. This set comes in the form of a library of properties described both in natural language and LTL which is the logic we have chosen to express our properties and which covered all of them in terms of expressivity. In a separate work [Khelladi 15], we also studied the four most used software development methods namely RUP, SCRUM, XP and KANBAN in order to extract from each of them, a set of best practices and constraints that should be enforced during the development process. This resulted in a ready-to-use library of constraints that we integrated to our process modelling and verification tool so process modelers can annotate their process models with one of these properties/constraints to make sure that the process model doesn't violate any of them.

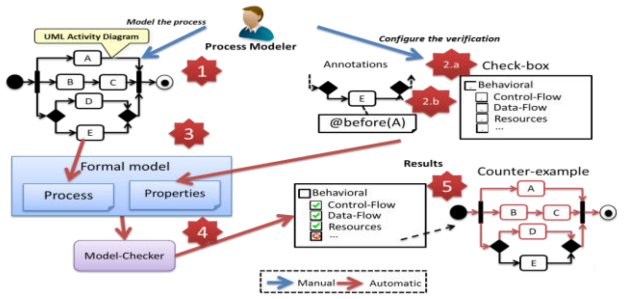


Figure 2. ProVer: overview of the approach

Table 1 depicts the set of properties we identified and their categorization.

Now that we have introduced the formalization of both the PML and the properties, the next sub-section gives an overview of the ProVer tool.

Overview of the approach. In this section, we briefly highlight the different steps required to run a process model verification using ProVer, then we present its internal architecture and the steps we followed in order to build this framework.

First, the process modeler needs either to design a process model or to import an existing one (see (1) in figure 2). In our tool, UML AD is used as a process modeling language. Secondly, the process modelers can choose among the library of predefined properties proposed by ProVer and which covers different process perspectives (*Control/ Data flow*, *Time* or *Resources*). This is done either by checking boxes (see (2.a) in figure 2) or by using an annotation-based language we defined in [Khelladi 15] (see (2.b) in figure 2). Process models and the properties are then translated into SMT specification which implements the semantics of UML AD as defined by our FOL formalization and based on the fUML standard (see (3) in figure 2). The SMT solver performs the verification (see (4) in figure 2) and if a counter-example is found, this is highlighted in the process model editor on the process model in red with a message of the unsatisfied property (see (5) in figure 2).

As we can see in the figure, the steps we followed in order to realize ProVer were 1) formalization of UML AD semantics as well as the set of properties in FOL; 2) implementation of the UML AD / Properties semantics and UML AD syntax in SMT; 3) implementation of the translations from UML AD/ Properties specifications => SMT; 3) graphical integration of the result into the process model editor.

In the following sections we will zoom-in SMT choice and the translation details. We will give examples of some concepts and properties expressed in SMT before to present the graphical interface of our tool. A user guide of the tool presenting all the details of the GUI is given here [Bendraou16].

Use of SMT for process verification

The verification process that we adopted is the well-known *Bounded Model Checking (BMC)* procedure [Clarke01]. Classically, this procedure inputs, 1) an initial state (for the system), 2) the transition relation, 3) a property to verify and, 4) a length bound k, then outputs the (un)satisfaction of the of system w.r.t the property up the bound k.

The implementation of such a procedure is done by constructing a logical formula that is satisfiable if and only if the underling transition system can realize a sequence of k state transitions that satisfy the property. If such a path segment cannot be found at the given length *k*, the search continues for larger *k*. The procedure is symbolic, *i.e.,* symbolic Boolean variables are utilized; thus, when a check is done for a specific path segment of length k, all path segments of length k are being examined. The formula that is formed is given to a satisfiability solving program and if a satisfying assignment is found, that assignment is a witness for the path segment of interest.

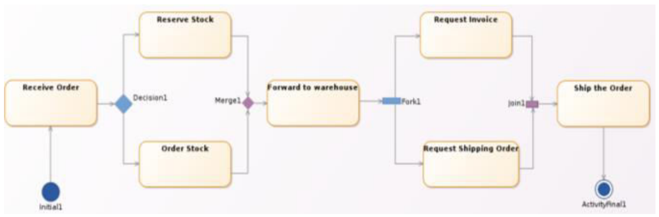


Figure 3. A process model with UML AD representing a selling process

If the formula is propositional and not very large then a basic Boolean SAT-solver can solve it efficiently. Things become more complicated when the formula is FOL! Actually, a first attempt to solve such formula is to transform it to propositional one and use SAT-solvers. However, this transformation is memory and time consuming and, most of all degenerative! For example, when comparing two 32-bits integer variables, we have to use 64 binary variables and compare them bitwise. So, the approach will have serious efficiency issues when the treated models are large.

The other option is to explore Satisfiability Modulo Theories (SMT). This area handles FOL formulae directly by use of simple transformation based on the SMT-LIB standard [Barret 15]. The SMT option seems to be very promising and is strengthened by the results obtained by SMT-solvers in the last SMT competitions (2014 and 2015).

Property	Alloy4SPV	ProVer	Speed Up	Result
Check Completion	*24*	*0.8*	*30*	*Verified*
Run Completion	*33*	*1.5*	*22*	*Model Found*
Check Total Time 6	*154*	*2*	*64*	*Verified*
Run Total Time 6	*63*	*2*	*23*	*Model Found*
Check Existence Reserve Stock = 0	*20*	*2*	*10*	*Counter Example Found*
Run Existence Reserve Stock = 0	*22*	*1*	*22*	*Model Found*

Table 2. Alloy approach (Alloy4SPV, with Alloy Analyzer) Vs. SMT-based approach (ProVer with z3 solver), performance for properties expressed on the process on figure 3.

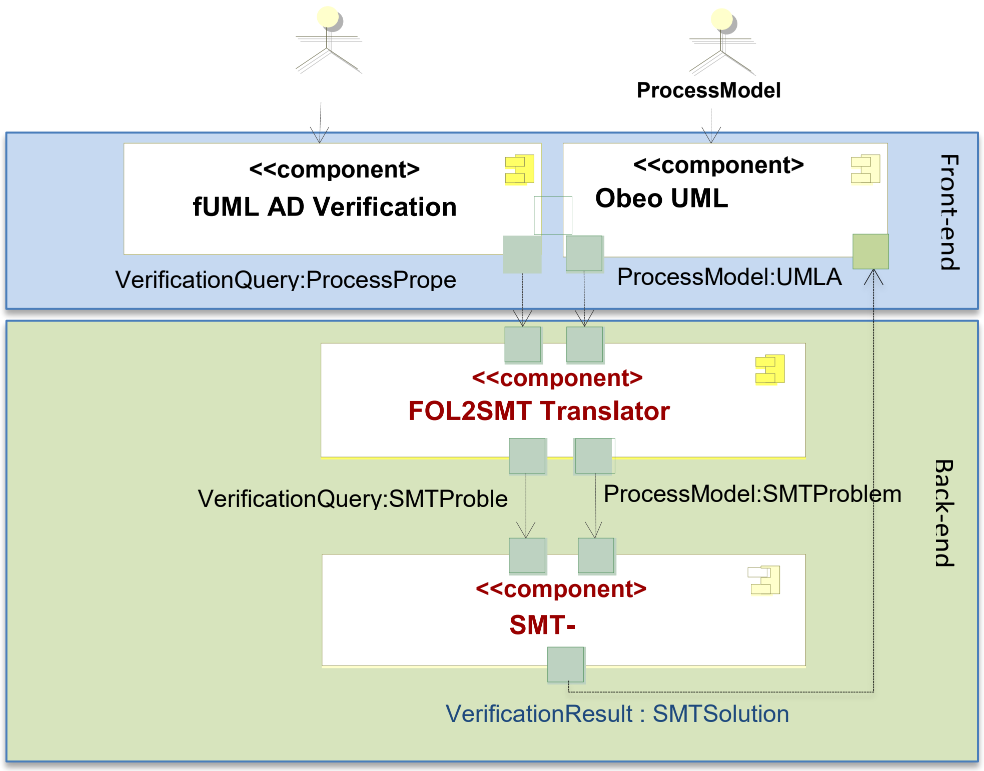


Figure 4. Architecture of the ProVer tool

Actually, solvers, like z3 (form Microsoft), Yices (from SRI International), show their ability to solve very complicated problems (from both academic and industrial worlds) in a reasonable time.

To give an idea on the effectiveness of our SMT-based approach, we use an example representing a selling process (see Figure. 3). The process includes 7 activities and 4 decision nodes (one from each type). Table 2 shows some results comparing our SAT-based approach (implemented in Alloy4SPV [Laurent 14b]) with respect to the SMT-based one. The first column of Table 2 represents the checked properties, while the second highlights the execution time (in seconds) with Alloy4SPV. The third column mentions the execution time of our ProVer tool (SMT-based) and the last column exhibits the result of the verification. We can clearly notice here the efficiency of our SMT-based approach just by looking the speed up we obtain for each property.

The results obtained so far are very interesting i.e**. 64x faster** in the example shown above and up to **92x faster** for some properties verification on another example, the OpenUP process, not presented here. The next section gives more details about ProVer architecture.

Tooling support: architecture and validation.

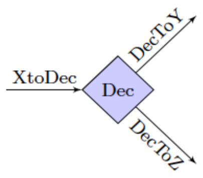
The architecture of our tool is highlighted in Fig. 4. It is based on the two classical layers: the front-end and the back-end. The former allows the interaction with the final user in a friendly graphical way. It inputs the UML AD representing the business process as well as the properties to be analysed. It outputs the result of the verification. *The Obeo UML Tool* component is dedicated to the input of the studied process model and the output of the verification result, while the *FUML AD Verification plugin* component deals with the input of the properties to be verified. **It is worth noting that no mathematical background is needed** to operate these inputs and interpret the results. All the properties can be selected through an integrated and ready to use library of properties.

Due to space limit we can't present in details the GUI of the tool but a user guide can be found here [Bendraou 16]. A java-like annotation language has also been developed to ease the specification of customized properties [Khelladi 15].
The two other components are *FOL2SMT Translator* and the *SMT-solver* components form the back-end. The former is the heart of the tool. It implements the verification request of the user as a FOL formula that is written in the SMT-LIB language. The produced SMT problem is then submitted to the SMT-solver component that resolves it. The result of that resolution is a feedback that is directly highlighted in the process model editor tool in a user friendly way. If the property is violated then an example (a path) of such a violation is highlighted.

Since the most complicated part in our tool is the one dedicated to the translation of a FOL formula to the SMT-LIB language, let us give some hints about its implementation by mean of a very simple example.

Consider the AD element of figure 5. It is a decision node (called *Dec*), that inputs the *XtoDec* arc and outputs two arcs: *DecToY* and *DecToZ*. According to the semantics defined in [Laurent 14], the execution of such a node considers two situations: (i) the node is ready to start; (ii) the node is ready to finish. The former case is identified by the presence of a token on the input arc and the absence of any on the decision node. In this case, the resulting action consists in moving the token from the input arc to the node. The later case is formalized by the presence of token on the decision node. Here, the resulting action is to move the token from the node to one of the outputs (chosen arbitrarily).

Figure 5. A decision node in an AD



If we note by *enableStart(Dec, t)* the predicate that represents the case (i) at time stamp t and *enableFinish(Dec,t)* the predicate that represents the case (ii) at time stamp t, then using the SMT-LIB language we can write:

```
- enableStart(Dec,t)=(and (>(select XToDec t)
                         0) (=(select Dec t)0)
- enableFinish(Dec,t) = (>(select Dec t)0)
```

Actually, *Dec, XtoDec, DecToY* and *DecToZ* are represented as arrays of integers indexed by integers. Each entry of these arrays defines the "marking" (number of tokens) of the element at the given time stamp. For example, `(select Dec t)` returns the marking of *Dec* at time *t.*

Similarly, if we consider *fireStart(Dec,t)* as the predicate that starts the execution of the decision node at time stamp t, while *fireFinish(Dec,t)* the one that terminates its execution, we obtain the following encoding:

```
- fireStart(Dec,t) = (and (= (select XToDec (+ t 1))
(- (select XToDec t) 1))
         ; Remove a token from XToDec
(= (select Dec (+ t 1)) (+ (select Dec t) 1))
         ; Add a token to Dec
(= (select DecToY (+ t 1)) (= (select DecToY t) 1))
         ; Between t and t+1 instants the marking
           of DecToY must remain the same
(= (select DecToZ (+ t 1)) (= (select DecToZ t) 1))
         ; Between t and t+1 instants the marking
           of DecToZ must remain the same )

- fireFinish(Dec,t) = (and (= (select Dec (+ t 1))
(- (select Dec t) 1))
         ; Remove a token from XToDec
(= (select XtoDec (+ t 1)) (select XToDec t))
         ; Between t and t+1 instants the marking
          of XToDec must remain the same
(or (and (= (select DecToY (+ t 1))
           (+ (select DecToY t) 1))
         (= (select DecToZ (+ t 1))
         (select DecToZ t)))
         ; Move the token to DecToY (choice 1)
    (and (= (select DecToY (+ t 1))
           (select DecToY t))
         (= (select DecToZ (+ t 1))
         (+ (select DecToZ t) 1)))
         ; Move the token to DecToZ (choice 2)
 ))
```

Actually, we operate such a transformation for all elements of AD that are necessary for the modeling of business processes. The implementation of this transformation needed nearly 2000 lines of code in java.

Conclusion

The most important contributions of this work are *(i)* **the identification of a reusable and configurable library of properties addressing all the process aspects (control, data flow, time and resources)**, *(ii)* **the formalization of this library as well as the UML AD semantics in first order logic**. This makes these definitions reusable for any other purposes such as the mapping to other model-checker formalism. In our case we opted for SMT theories.

The second undeniable contribution is **the integration and the verification of all process perspectives in a unified and integrated way thanks to our formalization and of our use of SMT theories**, within a user-friendly tool for process verification and execution. This framework has been adopted by our MeRGE industrial partners (European project) [MeRGE 12] and integrated to the MeRGE platform, an EMF eclipse framework for the development of safety critical systems.

Regarding our feedback using SMT for process verification it can be summarized in the following points: (1) handling complex constraints in a compact way, hence saving memory; (2) treating specific constraints with dedicated algorithms, hence saving time.

We are currently working on the validation of our approach on bigger process models from the Healthcare domain (more than 200 activities). We are also investigating the difference in performance according the property checked and the solver used. Indeed, in our experiments, the verification time for some properties was different from one solver into another.

References

[Awad 07] Awad. Bpmn-q : A language to query business processes. In *EMISA*, volume 119, pages 115–128, 2007

[Awad 09] A. Awad *et al*. “Specification, verification and explanation of violation for data aware compliance rules”. In *Service-Oriented Computing*, pp 500–515. Springer, 2009

[Barret 09] C. Barrett *et al*. Satisfiability Modulo Theories. In A. Biere, Marij J. H. Heule, H. van Maaren, and T. Walsh, editors, Handbook of Satisfiability, Vo. 185, chapter 26, pp 825–885. IOS Press, Feb. 2009.

[Barret 15] C. Barrett *et al*. The SMT-LIB Standard: Version 2.5. In Tec. report of Dep. of Comp. Science, The University of Iowa. 2015.

[Bendraou 05] R. Bendraou *et al.* "UML4SPM : A UML2.0-Based metamodel for Software Process Modeling", in Proceedings of the ACM/IEEE 8th International Conference on Model Driven Engineering Languages and Systems (MoDELS’05), Montego Bay, Jamaica, Oct. 2005, LNCS, Vol. 3713, PP 17-38

[Bendraou 10] R. Bendraou *et al*. “A comparison of six uml-based languages for software process modeling”.*Trans. Software Eng. 2010*

[Bendraou 16] ProVer Guide: https://pagesperso.lip6.fr/Reda.Bendraou/sites/Reda.Bendraou/IMG/pdf/prover_user_guide.pdf

[Christov 14] S.C. Christov, G. S. Avrunin , L.A. Clarke, American Medical Informatics Association Annual Symposium (AMIA 2014), November 15-17, 2014, Wash., DC, pp. 395-404. *(UM-CS-2014-022)*

[Clarke01] E. Clarke *et al.* Model Checking Using Satisfiability Solving. In journal of FMSD. Kluwer Academic Publishers. July 2001.

[Dijkman 08] R.M. Dijkman *et al*. “Semantics and analysis of business process models in bpmn”. *Information and Software Technology*, 50(12) :1281–1294, 2008

[Eshuis 02] Eshuis H. “Semantics and verification of uml activity diagrams for workflow modelling”. 2002

[Emerson 90] E.A. Emerson. Temporal and modal logic. “*Handbook of Theoretical Computer Science” Volume B. Formal Models and Semantics (B)*, 995 :1072, 1990

[Eshuis 06] H. Eshuis. “Symbolic model checking of uml activity diagrams”. TOSEM 15(1), (2006) 1-38

[Fahland 09] D. Fahland *et al*. “Instantaneous soundness checking of industrial business process models”. BPM. 2009. 278-293

[Hafer 87] Hafer and Wolfgang Thomas. “Computation tree logic ctl* and path quantifiers in the monadic theory of the binary tree”. In *Automata, Lang. and Prog.*, pp 269–279. Springer 1987

[Hofstede 02] T.Hofstede, A.: Workflow patterns: On the expressive power of petri-net-based workflow languages. In: of DAIMI, University of Aarhus, Citeseer (2002)

[Guelfi 05]Guelfi, N., Mammar, A.: A formal semantics of timed activity diagrams and its promela translation. In: Software Engineering Conference, 2005. APSEC'05. 12th Asia-Pacific, IEEE (2005)

[Gruhn 06] V. Gruhn and R.Laue. “Complexity metrics for business process models”. In *9th international conference on business information systems (BIS 2006)*, volume 85, pages 1–12, 2006.

[Gruhn 07] V.Gruhn and R. Laue.” What business process modelers can learn from programmers”. *S. of Comp. Prog.*, 65(1) :4–13, 2007.

[Khelladi 15]D. Khelladi *et al.* A framework to formally verify conformance of a software process to a software method. SAC 2015 : 1518-1525

[Knuplesch 10] D. Knuplesch *et al*. On enabling data-aware compliance checking of business process models. *Con. Modeling–ER 2010*, pages 332–346, 2010

[Kleppe 03] A. Kleppe *et al*. The model driven architecture: practice and promise, 2003

[Jensen 87] K. Jensen. ***Coloured petri nets***. Springer, 1987

[Jung 10] Jung, H.T., Joo, S.H.: Transformation of an activity model colored petri net model. In: TISC, IEEE (2010) 32-37

[Mendling 06] J an Mendling *et al*.. Faulty epcs in the sap reference model. In *Buss. Proc. Manag.*, pp 451–457. Springer, 2006.

[Laurent 14] Y. Laurent *et al*. “Formalization of fUML: An Application to Process Verification”, CAiSE 2014, Springer, pp . 347-363

[Laurent 14b] Y. Laurent *et al*. Alloy4SPV : a Formal Framework for Software Process Verification, 10th ECMFA, LNCS, pp.83-100, 2014

[Laurent 18] Y. Laurent’s PhD Thesis document, LIP6, March 2018, https://drive.google.com/open?id=14p9i4ulLacjdUIgFhZKUVI0wfQa4x6hP

[Liu 07]Liu, Y., et. al.: A static compliance-checking framework for business process models. IBM Systems Journal 46(2) (2007) 335-361

[Mendling 07] Jan Mendling, Gustaf Neumann, and Wil Van Der Aalst. Understanding the occurrence of errors in process models based on metrics. In On the Move to Meaningful Internet Systems 2007 : CoopIS, DOA, ODBASE, GADA, and IS, pp 113–130. Springer, 2007.

[Mendling 09] Jan Mendling. Empirical studies in process model verification. In TPNOMC II, pp 208–224. Springer, 2009.

[MeRGE 12] Merge, ITEA project, safety & security. http://www.merge-project.eu/. last vist Oct 2015.

[Morimoto 08] S.Morimoto. A survey of formal verification for business process modeling. In *Computational Science–ICCS 2008*, pages 514–522. Springer, 2008

[OMG 11] Object Management Group (OMG). Business process model and notation (bpmn) version 2.0, jan 2011

[OMG 11b] OMG. Semantics of a foundational subset for executable uml models (fuml) version 1.0. http: //www.omg.org/spec/FUML/, 2011

[Peterson 81] Peterson J.L. Petri net theory and the modeling of systems. 1981. 5, 29

[Pnueli 77] A. Pnueli. The temporal logic of programs. In FCS, pp 46–57. 1977.

[Simidchieva 10] B I. Simidchieva, *et al*., Proceedings of the 2010 Electronic Voting Technology Workshop/Workshop on Trustworthy Elections (EVT/WOTE '10), August 9-10, 2010, Washington, DC.

[Smith 02] R.L Smith *et al*. An approach supporting property elucidation. In *Proceedings of the 24th ICSE.*, pp 11–21. ACM, 2002

[Trcka 09] N. Trcka *et al*. Data-flow anti-patterns: Discovering data-flow errors in workflows. In CAISE, Springer (2009). 425-439.

[Van der Aalst 98] van der Aalst, W.M.: The application of petri nets to workflow management. JCSC 8(01) (1998) 21-66.

[Van der Aalst 99] van der Aalst, W.M. Formalization and verification of event-driven process chains. *IST*, 41(10): 639–650, 1999

[Van Der Aalst 03] van Der Aalst, W.M., Ter Hofstede, A.H., Kiepuszewski, B., Barros, A.P.: Workflow patterns. Distributed and parallel databases 14(1) (2003) 5, 51

[Van der Aalst 11] van der Aalst,et al. “Soundness of workflow nets: classification, decidability, and analysis”. Formal Aspects of Computing 23(3) (2011) 333-363

[Vanhatalo 07] J.Vanhatalo *et al*.. Faster and more focused control-flow analysis for business process models through sese decomposition. In ICSOC 2007, pp 43–55. Springer, 2007. 3, 33, 143

[Watahiki 11]K. Watahiki *et al*. Formal verification of business processes with temporal and resource constraints. In *Systems, Man, and Cybernetifcs (SMC)*, pp1173–1180., 2011

[Wohed 05] Wohed *et al*. Pattern-based analysis of the control-ow perspective of uml activity diagrams. In: Conceptual Modeling ER 2005. Springer (2005) 63-78

[Wong 07] Wong, P., Gibbons, J.: A process-algebraic approach to workflow specification and refinement. In: Software Composition, Springer (2007) 51-65