A tool for the verification of Data-aware Business Processes

Luca Sabiucciu, Marco Montali[0000−0002−8021−3430], and Sergio Tessaris[0000−0002−3156−2669]

Free University of Bozen–Bolzano, Bolzano, Italy
lsabiucciu@unibz.it
montali@inf.unibz.it
tessaris@inf.unibz.it

Abstract. Verification of data-aware Business Processes is a highly complex and time consuming activity. As Business Processes tend to increase in terms of both size and complexity, the process of verifying such, becomes difficult even for experts. Data values may cause, for example, a deadlock in the control-flow of a Business Process, due to unsatisfied constraints on the data values, preventing the procedure of the process. Although commercial and non-commercial suites handling both control-flow and data-flow are available on the market, they struggle to produce an impact, due to the fact that the data-flow is on a separated layer from the control-flow.
In this paper we present the experimental results of the first prototype of the RAW-SYS framework, a framework for the verification of data-aware Business Processes, using small sized Business Process models, but arbitrarily complex, with updates on the data values and relying on data-constraints in order, for the process, to proceed. Despite the restricted size of the models, results are good and suggest that planning techniques are a valid way of verifying data-aware Business Processes.

Keywords: Business processes · Automated planning · Data-aware workflow verification

1 Introduction

In recent years, organisations have increasingly adopted *business process management* (BPM) to understand, structure, monitor, and govern their internal work, and better achieve their strategic goals [9]. At the same time, processes have grown both in size and complexity, dealing with concurrent activities, data stored in different information systems, and many types of resources (employees, managers, consultants, etc.).

Business Process Management professionals reserve a first-class citizenship to processes, downplaying the importance of data [3]. As a result, it is often hidden inside the logic of activities how the data is modified, e.g. within Java code. Therefore, it is not possible to verify data related aspects since how the

data is modified is not formally captured. Several theoretical frameworks have been developed in order to verify formal properties considering the interplay between both aspects. However, such frameworks are often too far from the modelling languages adopted in practice, or do not properly represent data in real world scenarios. To overcame these limitations the formal framework *RAW-SYS* has been recently introduced combining process and data modelling which captures a wide range of features provided by commercial BPM systems [4]. RAW-SYS is based on *Petri Nets* (a formal representation of BPs capturing the control-flow) to model the control-flow, and relational databases (the most widely used and well-known persistent data model) to represent persistent data; moreover it enables the formal verification of BP models leveraging state of the art automated planning systems.

In this paper we present the results obtained from the development of a first prototype for the RAW-SYS framework. Results are promising as the framework efficiently verifies moderately complex BP models enriched with data.

2 Background

In this section we introduce some basic concepts that are used further in the paper.

2.1 Business Processes

A Business Process (BP), is a set of activities, belonging to a Business, organized such that, when performed, a particular goal/objective is reached. BPs represent flows of activities that are performed sequentially, in parallel or exclusively in order to achieved a certain goal. The Business Process Management Notation, described in [8]makes use of simple symbols and is thus easy to understand even from non-experts of the field (e.g. managers).

2.2 Petri Nets

A Petri Net (PN) is a bipartite graph, whose nodes are of two types, namely transitions and places [1]. It is the most widely used formal language for representing the control-flow of BPs. Rectangles, named transitions, represent activities, while circles represent momentary phases, from one transition to the following. Arcs denote the flow relation, that is, the sequence flow of activities. Two objects of the same type cannot be connected via an arc.

Moreover we rely on the *Petri Net Markup Language* for storing the PN on a file. Such representation allows for portability and a standardized way of persistently storing PNs. A simple example of PN is shown in Figure 1.

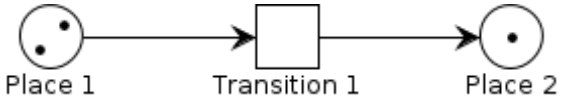

Fig. 1. Simple PN example

2.3 Action Language

Action languages are formal models of parts of the natural language that are used for talking about the effects of actions [5].Action languages are part of the reasoning branch of Artificial Intelligence, which make use of planning techniques. In particular, we make use of the *PDDL Action Language*, a language derived from *STRIPS* and used for planning competitions. The language is described by means of *actions*, having a set of *preconditions*, representing the necessary condition(s) in order to execute the action, and a set of *postconditions*, a set of conditions that will be true (or false, depending on the specification) after executing the action.

2.4 RAW-SYS Model

The RAW-SYS framework, described in [4], is a framework for verifying data-aware Business Processes. On the one hand, it graphically represents BPs by means of Petri Nets. On the other hand, it stores data on the well-known relational model of databases, called *data store* in RAW-SYS.

In RAW-SYS, the class of Petri Nets is restricted to Workflow-nets (WF-nets), Petri Nets with a single starting place and one sinking place. Additionally, the Petri Nets used are limited to 1-safeness. That is, only one token may be assigned to each place. WF-nets are equipped with a data store. This new type of nets is called Relational-AWare workflow nets, namely RAW-nets. The particularity of RAW-nets, is that transitions are equipped with guards. Guards are queries over the data store which represent either a condition on the data store that has to be satisfied, or an effect of the firing (i.e. execution) of a transition that is executed when performing the activity. An unsatisfied guard prevents the transition from being fired.

3 Petri Net Conversion

In order to convert the BP, from the PN notation, to a planning domain and problem, in PDDL, we created a *Java* program that performs such conversion. The underlying idea of the conversion is described in this section.

3.1 Domain Definition

As simple as it sounds, in action languages, including PDDL, actions represent, literally, an action. Thus PN's transitions can be mapped to PDDL's actions,

since they have the same meaning. A PDDL action has preconditions and effects for regulating the execution of it. On the other side, the firing of a Petri Net's transition, is regulated by tokens, marking the places in the preset of the transition and the postset of it. Therefore in the action language representation, PN's places have to be present along with tokens, and will be managed by predicates, moving tokens from one place to another, after the execution of an action.

Since RAW-SYS guards are a FO (First Order) query, which verifies with the local database whether the transition can be fired, in PDDL the constraint can be seen as part of an action's *precondition*, which has to be satisfied in order to execute the action. Therefore a guard's constraint is part of a PDDL action. On the other side, RAW-SYS actions are updates of the data values and are therefore associated with the firing of a transition. In PDDL, the only way to modify the values of predicates is via the *effect* of an action, since the precondition only verifies that a constraint is satisfied. Therefore the RAW-SYS action is translated into PDDL predicates and inserted into the *effect* of the action.

3.2 Problem Definition

Problems in PDDL are separated from domains, as a domain may be posed several problems . In order to verify the PN represented in the domain, the problem first declares the set of objects that may be involved for finding a solution, then describes the initial state of the PN, from the marking to the data. Lastly, it defines a goal that has to be satisfied in order for the sequence of actions to be a solution. In our case the goal is to reach the sinking place, that is, ending the process.

4 Evaluation Methodology

In this section we first provide a description of how the experiment has been conducted. Next, we provide a description of the BP models used for testing, followed by a description of the testing environment.

4.1 Experimental Approach

To investigate the behavior of the planner, we explore the relation between BP model's characteristics and time needed to verify the model. In particular we focus on two types of models: *(i)* single process instance and *(ii)* multiple process instance. On the one hand, *(i)* represents independent processes that do not need any other process instance to access nor to modify data values in order to prosecute. On the other hand, *(ii)*, represents process instances that rely on each other in order to either create, remove or update data values in order to proceed with the process flow. The two process types are of particular importance because of their expressibility, in terms of models, capturing most of the BP models that may be needed.

The planner used for the experiments is a generic planner, *Fast-Downward* [6], developed for planning competitions and accepting the *PDDL* action language, as described in Section 2.3. The language supported by the planner is not the complete PDDL language specification, but a subset, rich enough for our purpose. Other planners can be adopted, depending on the considered model and on the process related activities. We chose to use such planner because of it's generality and performances.

For the single process instance models, we varied the number of objects, while the number of concurrent process instances was free. Instead, the multiple process instance models had a fixed amount of objects, but we varied number of concurrent process instances that were running.

On the one hand, the single process model was tested with the same values 10 times and the resulting times were averaged, for each step of increasing data object. The number of data objects was initially set to 5, and incremented by 5 units until 25 data objects (latter included). On the other hand, the multiple process models were tested with a fixed number of data objects (5), and a varying number of process instances, starting from 2 to 10, incrementing at each step by 2 process instances. As before, the same configuration is run 10 times and timing is averaged.

For all tests the same search strategy for the planner is used, a greedy search with no memory nor time bounds. This search strategy makes use of the *FF* heuristic [6].

4.2 Business Process Models

For our experiments we considered three different BPs, each of different size and complexity. The first example that we considered is taken from [2] and represents the process for claims to a car insurance company. As Figure 2 shows, the process starts with two activities that can be performed in parallel: *check_insurance* and *contact_garage*. In this two activities a data value is updated and such value is used, afterwards, when the process flow is converged, where depending on the data constraint, either the *pay_damage* activity is performed or the *send_letter* activity is performed. The process flow ends after one of the two activities is performed. For simplicity, we will refer to it as *insurance model*.

The second considered BP is taken from the YAWL foundation and is described in [7]. It represents the workflow for a film production. Since the example was available only in the form of workflow-net, we converted it into a *RAW-SYS* model. As Figure 3 shows, the size is quite big, and we will thus, only describe the underlying idea. The number of shooting days is decided a priori and inserted, along with other data values, into the system via a first phase. Once such phase terminates, the process starts cycling over a series of activities, representing a shooting day. The number of cycles is defined by the number of shooting days. After the shooting days have ended, the process flow ends too in a final state. Several data updates are performed during the cycles on the same data objects, which are inserted during the first phase. We will refer at this model as *YAWL4FILM*.

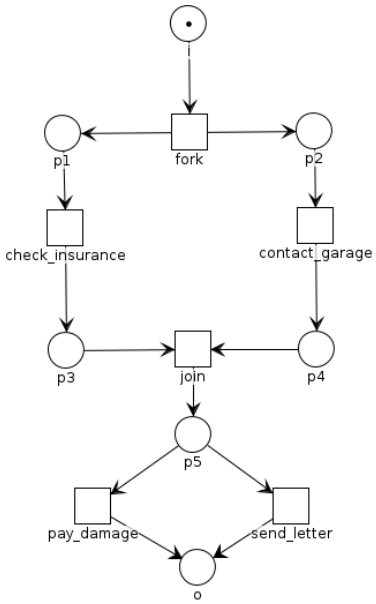

Fig. 2. PN for the insurance model example

The last example used, is a synthetic one, specially designed to stress the updates on the data from concurrent process, which have to collaborate in order to terminate. Therefore the small size. The PN is shown in Figure 4 and represents a shipping domain, where, on the one hand, goods are chosen, purchased and received from customers and, on the other hand, a warehouse employee receives the order from the customer and processes it by shipping it to the client. Note that the client has to wait for the purchased goods to be shipped from an employee, while the employee cannot ship the products before the customer has purchased those products. This relation between the data, leads to having at least two concurrent processes active at the same time. We will refer at this model as *Goods Shipping*.

4.3 Testing Environment

The timings are taken with the *time* command of the *Linux System*, considering the total time taken, from the conversion of the PN to PDDL, to the solution (there is always one), by the process, including waiting times for *I/O* operations. All the tests are conducted on a *64-bit Ubuntu 17.04* machine, mounting an *Intel Core i7-4710HQ* processor (2.5GHz, 4 cores, 8 threads), 8 GBs of RAM and 3,4 GBs of swap space on a mechanical hard-disk.

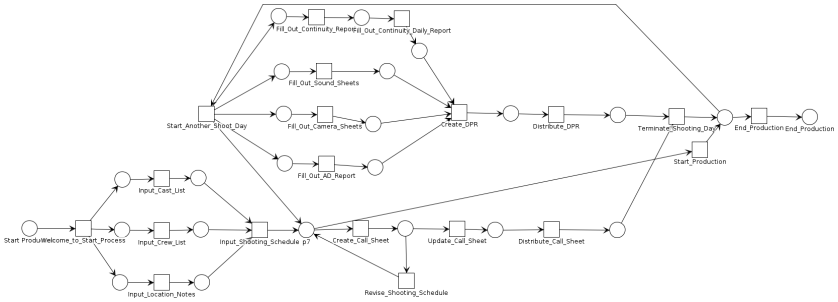

Fig. 3. PN for the YAWL4FILM example

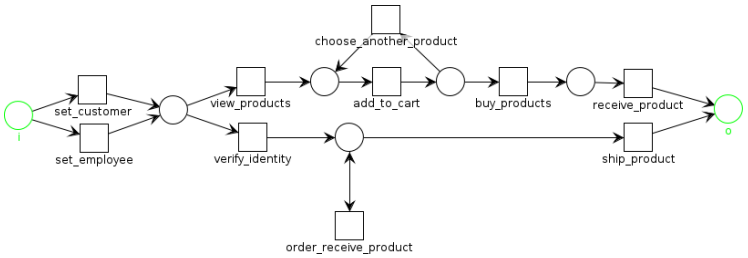

Fig. 4. PN for the *Goods Shipping* example

5 Results

In this section we present the results obtained from testing the prototype within the environment described in Section 4. We start describing the results of single process instance models, followed by the results of multiple process instance models.

5.1 Single Process Instance Models

We first tested the running time of the single process instance models, as they were expected to take less time than the multiple process instance model. The obtained results of both single process instance models are shown in Figure 5, providing a visual comparison of the running time. Expectable is that the size and the complexity of the business model influence the running time of the planner.

As the graph in Figure 5 shows, time grows along with the number of objects the planner has to handle. Notable is also the growing gap between the two models. The bigger and more complex the model, the more time is needed to verify such. Overall, the planner provides an answer rather quickly: in the worst case, slightly less than 0.5 seconds for the *YAWL4FILM* model with 25 data objects, while slightly less than 0.1 seconds for the *insurance model*.

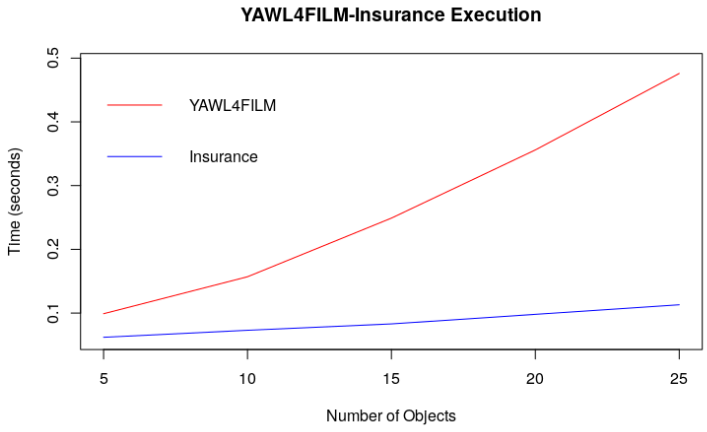

Fig. 5. Running time for single process instance

5.2 Multiple Process Instance Model

As expected, the multiple process instance model takes more than the single process instances. This is due to the fact that the number of possible choices of the planner increase exponentially, as the number of concurrent processes increases. The increase in time required by the planner is much drastic with respect to the other models.

6 Discussion

The obtained results reveal that verification of data-aware Business Processes via planning techniques is feasible, for small models at least. The three major influences on the running time, that we could notice, are size, complexity (the number of data objects as well as the constraints and loops) and number of concurrent processes in the model.

For single process instance models, the growth of time needed to verify the model, increases as expected, with the number of data objects. Noticeable is the increase of the gap between the two models with increasing number of objects. This gap suggests that the growth of time needed to discover a solution, along with the increase of data objects to handle, depends also on both size and complexity of the model. In particular, we believe that loops play a significant role in such increase for single process instance models. This hypothesis shall be confirmed by further tests stressing such aspect.

On the other side, when verifying the *Goods Shipping* model with two process instances running concurrently, the planner takes approximately the same time of the *film model* example with 25 data objects. Although, the size of the *Goods Shipping* model is significantly smaller than the size of the *YAWL4FILM* model,

Fig. 6. Running time for multiple process instances

with a different number of data objects, multi process instance models require more time to find a solution, with respect to single process instance models. From this fact, we can deduce that the running time is heavily influenced by the number of concurrent processes.

Other planners may perform differently, as we used an off-the-shelf planner, Fast-Downward. Planning techniques may vary from planner to planner and thus, may affect the running time. Also the search strategy of the planner may influence the running time of the planner. Finally, the number of threads of the machine may influence the running time as well.

7 Future Work and Conclusions

Formal verification of BP models is an important aspect of BP modeling, that has been studied only theoretically. In this paper we have shown the results of a prototype for the RAW-SYS framework, for modeling and verifying data-aware processes, based on the well-established PNs for representing BP models and the planning language PDDL for solving the verification problem. Results look promising and support the usage of planners for verification of data-aware business processes. Although very big and complex BP models may not perform as well as smaller ones, we recall the fact that the planner used for the tests is an off-the-shelf planner, which is likely to be outperformed by other, more task-specific, ones. In this regard, the system will be further developed for accommodating other planners. Performances of other planners will be tested with respect to other BP models.

Additional, complex, object types are part of the next implementation steps for

our tool. Moreover, support for other planning languages that provide different features than the ones provided by classical planning (e.g. temporal planning), are part of the next implementation steps as well.

References

1. van der Aalst, W.: Verification of workflow nets. Application and Theory of Petri Nets 1997 pp. 407–426 (1997)
2. van der Aalst, W., Stahl, C.: Modeling Business Processes: A Petri Net-Oriented Approach. MIT Press (2011)
3. Calvanese, D., Giacomo, G.D., Montali, M.: Foundations of data-aware process analysis: A database theory perspective. In: Hull, R., Fan, W. (eds.) Proceedings of the 32nd ACM SIGACT SIGMOD SIGART Symposium on Principles of Database Systems (PODS). pp. 1–12. ACM Press (2013). https://doi.org/10.1145/2463664.2467796, http://dl.acm.org/citation.cfm?doid=2463664.2467796
4. De Masellis, R., Francescomarino, C.D., Ghidini, C., Montali, M., Tessaris, S.: Add data into business process verification: Bridging the gap between theory and practice. In: Proceedings of the Thirty-First AAAI Conference on Artificial Intelligence. pp. 1091–1099 (2017), http://aaai.org/ocs/index.php/AAAI/AAAI17/paper/view/14627
5. Fox, M., Long, D.: Pddl2. 1: An extension to pddl for expressing temporal planning domains. Journal of artificial intelligence research (2003)
6. Helmert, M.: The fast downward planning system. Journal of Artificial Intelligence Research **26**, 191–246 (2006)
7. ter Hofstede, A., van der Aalst, W., Adams, M., Russell, N.: Modern Business Process Automation: YAWL and its support environment. Springer Science & Business Media (2009)
8. Model, B.P.: Notation (bpmn) version 2.0. OMG Specification, Object Management Group pp. 22–31 (2011)
9. Weske, M.: Business Process Management - Concepts, Languages, Architectures. Springer, 2nd edition edn. (2012). https://doi.org/10.1007/978-3-642-28616-2, https://doi.org/10.1007/978-3-642-28616-2