Automatized Checking of Business Rules for Activity Execution Sequence in Workflows

Cristiano Barros, Mark Song
Informatics Institute
Pontifícia Universidade Católica de Minas Gerais
Belo Horizonte, Brazil
Email: falecomigo@cristianobarros.com.br, song@pucminas.br

***Abstract*— In the present context of high competition, the enterprises are investing in the improvement of the efficiency of their production and management processes. To accomplish such tasks, one viable way is to formalize those processes into business models, so it is possible to evaluate and improve them. As important as the analysis of software engineering artifacts prior to building information systems, also is important the analysis of the definitions existing in the business models, to ensure that they reflect the reality of the process and its demands. This paper aims to develop an approach for the execution of validations of execution rules incorporated to workflows, using model checking to ensure the exactitude of the analysis.**

***Index Terms*— workflow, model checking, business rules**

I. Introduction

Several authors define the present business environment as highly competitive, being essential to search, not only ways to differentiate their products and services, but also new ways to increase the efficiency of the whole company, in order to obtain good economical results. Besides, in the last years, the business analysis subject has been gaining more importance due to its capacity to attack problems originated in the relation between the business and IT sectors [1]. In that sense, it is becoming popular the revision of business processes as a way to increase the efficiency in certain manners: exclusion of unnecessary activities, integration between processes, simplification of procedures, automatization of repetitive tasks, among others [2].

Workflows or business processes are defined as a series of task in a specific sequence aiming to obtain a determined objective. Inside the information systems engineering field, the business modeling technique formalizes the sequencing rules of workflows in order to aid the development of information systems by supporting and automatizing tasks. It is recognized as a good practice to model a business process before the actual development of a software to support it, so that its premises and rules are well defined, established and derived from the real procedures and not from an ideal representation. It is not desirable to invest in the development of an information system based on inefficient, inaccurate or ideal procedures.

This work was supported by FIP PUC Minas.

Like software developers worry about bugs and execute several tests to ensure the quality, business analysts should also worry and test the models. Some software bugs could be originated not from mistakes in software engineering artifacts but from misunderstandings that occurred when the workflows, over which a system is developed, were designed. Error due to problems with the business rules are hard to detect and severely affected the applicability of a software. Sooner the error are detected, cheaper they are to fix. The technique of business modeling also increases the efficiency of software engineering methods because most of the requirements and details are already defined.

This current work has the main objective to present an approach to formally check execution sequencing rules of workflow written in XPDL (XML Process Definition Language). In more details, this research creates a tool that imports a XPDL file that contains the specification of a workflow, translating it to be executed in a model checker. It allows the user to execute temporal logic queries that are answered whether the properties are true or false on that model.

II. Background and Related Work

In order to produce the expected results, it was necessary the understanding of business process modeling, the XPDL language used to describe workflows and also model checking, used to validate the execution rules. Some related works are presented.

A. Business process modeling

Business processes, also known as workflows are the "combination of a set of activities within an enterprise with a structure describing their logical order and dependence whose objective is to produce a desired result" [3].

In the last decade was possible to notice an bigger interest by the organizations to adopt business process modeling techniques as a way to increase the efficiency in corporative or even scientific works, being those supported or not by information systems [4] [5]. The development of information systems as a means to support or execute workflows that are inadequate, non-optimized or non-adherent to previous specifications and models reveals itself as a considerable source of time and money waste and also a problem that is hard to detect and solve.

doi:10.4304/jsw.7.2.374-381

The business modeling can be formally described with models or diagrams that identifies activities, information and the its flow associated with a business operation [6]. To do so, there are notation patterns for workflows which are nothing more than graphical patterns that associate shapes and element types [7]. Among those, BPMN[1], has a good acceptability by the market for being dedicated exclusively to business processes [8].

In order to transform a notation into an executable format, it is necessary to change the diagrams into process specifications, using specific languages. There are several languages available: PSL, DAML-S, RFP, XPDL, BPML e BPEL. Among those, XPDL[2] - XML Process Definition Language was used in many works.

When modeling a business process, the rules that define and detail the execution of activities are clearly established. According to [10], corporative business rules are defined as restriction or metadata to business operations. Those rules can be originated from the logic of the own process, scientific methods (that is the case of academic research, for example), or even regulations or standards [11]. In information systems, those rules can be defined within the notation or through specific languages.

B. XPDL

XPDL language has a main purpose to formalize the specification of workflows [12]. It translates the definition made by graphic notation patterns like BPMN or even UML. The definitions are written in XML format and obey the rules created by a consort of companies of this field called Workflow Management Coalition (WfMC) [9]. It defines a formal semantics to represent tasks, its transitions, resources and even actors involved in a workflow. Its main purpose is to serve as a pattern for interoperability between systems that use different standards [13].

The XPDL elements considered in this research was the definition of processes, subflows, activities and transitions. The other components present in the language were not considered since they do not affect the proposed analysis of this work, such as participants, artifacts, lanes and application.

C. Model checking

Model checking is a formal method that allows the exploring of finite state systems as a transition graph, executing temporal logic validations. For a specific set of state transitions it is possible to verify temporal logic affirmatives to ensure whether they are true or false. [14] formalized a model checking technique with the objective of automatizing the verification process of software design in a similar way to that applied to electronic circuits. Through this technique, the finite set of states a software can assume is tested with temporal logic clauses so it is possible to evaluate the sequencing of those states, searching for logical problems. This method has advantages over traditional approaches that use simulation, tests or deductions, due to the fact that it is automatic and exhaustive.

D. Workflow verification

The majority of the works deal with the verification of general rules, intrinsic to the workflow logic, such as the search of unreachable states, deadlocks or livelocks. Some other works include also the verification of safety or liveness properties. Beyond those, some researches have objectives close to the present work.

In [15] the use of model checking is proposed to verify business models using the platform Testbed, AMBER language and checker SPIN. It shows the capability of the use of such set of tools by non-specialists with the objective of improving the design process through execution simulations. In [16] the authors show a method of transforming a business process into a non-deterministic state automaton and the application of the model checking technique on this automaton.

In [12], using XPDL and the SPIN checker, demonstrates a methodology of verification in a case study of a travel agency. It not only validates the absence of unreachable activities, deadlocks and livelocks but also sequencing of the activities. The XPDL specification is translated into the Promela language which is further manually executed in the SPIN.

In [17] is created a tool called VERBUS that allows the modeling and verification of business processes, showing its applicability. The main purpose is to evaluate invariant properties and state reachability. In this same direction, [18] uses the Promela language and the SPIN checker to verify specification of sale processes written in XPDL. Its objective is the creation of a system that allows the verification of systemic attributes (absence of deadlocks or livelocks) and also *ad hoc* properties of dependency among activities. Using Petri Nets, the article [1] describes a method to evaluate if the tasks of a flow are concluded when the whole process is finished and the capability to reach a certain element of the flow.

The main difference among these researches and this is due to the removal of the necessity for specific knowledge about the techniques used. This interface aids the non-specialists on XPDL or temporal logic to use these techniques. The knowledge needed to use the software is just concerning business models and the business logic involved in each model evaluated. However, this strategy also presents itself as a limitation for it narrows the possibilities of evaluations to those defined in the software.

III. Approach

In the diagram shown in Figure 1 is presented the way the software works. In the main interface, the first step is to choose and load a XPDL file containing the specification of a workflow. With the file loaded, the

[1]BPMN was created by OMG - Object Management Group with the purpose to become a pattern specific to graphically represent workflows.

[2]This language was created by WfMC - Workflow Management Coalition [9]. Both OMG and WfMC are organizations that congregate several companies and researchers specialized in this matter.

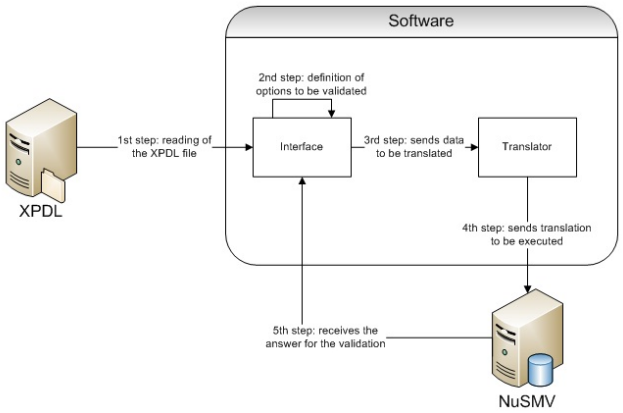

Figure 1. Diagram of the software operation

software presents the form elements that will be used to generate the temporal logic verification clauses.

A. Elements analyzed

Each activity, start event or end event are translated into boolean variables that represent if that element was executed or not at that time. In each cycle, the checker determines if a specific element has the condition of be executed. To translate the different behavior of the elements on a workflow, some cases were identified:

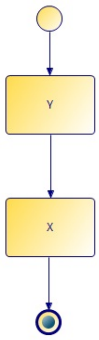

Figure 2. Simple transition

1) Simple transitions: This first case translates the simpler occurrences, in which an activity receives only one transition from another activity. There is just one condition for the execution of that activity that is the execution of the previous one. Taking the Figure 2 as an example, the translation will produce the NuSMV code (Code 1).

```
1 next(atividadeX) :=
2       case
3               atividadeX = 1 : 1;
4               atividadeX = 0 & atividadeY = 1 : 1;
5               1: 0;
6       esac;
```

Code 1. Translation fragment of a simple transition

The Code 1 shows the different behaviors applied to activity X. If that activity was previously executed, then it maintains that status (line 3). If it was not yet executed but the previous activity was, then it will be executed at that moment (line 4). For all the other conditions, it remains unchanged (line 5).

2) Start and end events: A start event is always executed, prior to any other element. This behavior is translated by the Code 2. The condition for the execution of an end event is defined in the same way as an activity.

```
1 next(atividade0) := 1;
```

Code 2. Fragment of translation for a start event

It is important however, to establish the difference between start or end events of subflows. It is possible in BPMN to enclosure a workflow into an activity, creating subflows that detail the actions of that element. The start and end events of subflows are not considered in the same way as those elements in the most abstract flow. They serve only as connectors between the elements of the subflow and the more abstract flow.

3) Exclusive gateways: When the activities being evaluated are involved with conditional gateways (Figure 3) and the condition is not specified, it is necessary to insert the characteristic of nondeterminism for the execution of the paths. The model checker will explore the possibilities derived from that division in the flow, considering all the possible paths and the elements involved.

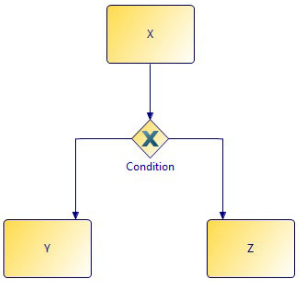

Figure 3. Example of exclusive gateway

The nondeterminism does not implicate in loss of precision because all possible paths are evaluated if there are no definitions about the condition specified by the gateway. When the condition is provided, the nondeterminism is not inserted.

```
1  next(atividadeY) :=
2      case
3              atividadeX=1 & atividadeY=1: 1;
4              atividadeX=1 & atividadeZ=1: 0;
5              1: { 0, 1 };
6      esac;
7
8  next(atividadeZ) :=
9      case
10             atividadeX=1 & atividadeZ=1: 1;
11             atividadeX=1 & atividadeY=1: 0;
12             1: 0;
13     esac;
```

Code 3. Fragment of a code for exclusive gateways

The Code 3 shows the translation of the Figure 3. The nondeterminism was defined by the line 5. In that example, the other lines have the purpose of maintain previous status and also avoid the execution of an exclusive path if any element of the other paths were executed.

A more complex case occurs when multiple exclusive gateways are connected to each other. To evaluate the condition for each element involved, it is necessary to consider the interference among all of them. Another case of different behavior refers to the possibility of detours, when one of the paths from the gateway joins the other. That occurrence demands a specific evaluation since it impose different execution conditions to the activities.

4) Parallel gateways: This case deals with the multiplication of the flow originated by parallel gateways. The parallel execution, as shown in Figure 4, is translated as the Code 4. The activities of each parallel path are executed after the activity X.

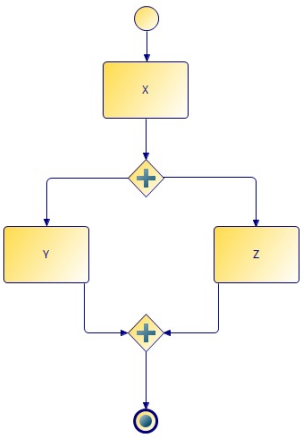

Figure 4. Example of parallelism with join

In code 4, only the split element is translated. But the case of the parallel join element is not complex. Since all the paths are entirely executed (except if encapsulated cycles occur, as it will be explained later) the condition for an element after the joining is the same for the first elements of each parallel paths. In other words, if a parallel split is reached, its end will also be reached.

When the software receives such type of element, it analyses the first activities of each path and also the element after the join that ends the parallelism. It is considered as a good practice to explicitly indicate the closure of the parallel paths through a join element [8]. The software does not demand that practice but impose a pattern that all the elements must be executed for the flow to continue.

```
1  next(atividadeY) :=
2      case
3              atividadeX=1: 1;
4              1: 0;
5      esac;
6
7  next(atividadeZ) :=
8      case
9              atividadeX=1: 1;
10             1: 0;
11     esac;
```

Code 4. Fragment of the translation with the parallelism

B. Validation types

Seven validation types were made available to test the relations between the elements:

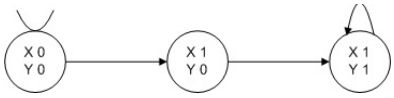

Figure 5. State diagram for "The execution of an activity implicates in the execution of another"

1) The execution of an activity implicates in the execution of another: To evaluate this validation type, shown in the state diagram in Figure 5, the following command was used. In that command, 'x' and 'y' represent activities of the flow.

SPEC AG(activityx -> AF(activityy));

The command can be translated as: "in all states of the model, whenever the activity 'x' is executed, then in all the possible paths of execution, in the future, the activity 'y' will be executed". As an example of this validation, it is possible to test in a corporative procurement process if, whenever a supervisor approves an order, the products bought must be delivered. If there are possibilities specified in the workflow that prevent this outcome after the approval, then the software would answer negative to that validation case.

Figure 6. State diagram for "The execution of an activity implicates in the non-execution of another"

2) The execution of an activity implicates in the non-execution of another: To evaluate this relationship type (Figure 6), it is used the following command:

SPEC AG(activityx -> AG(!activityy));

It can be understood as: "in all the states of the model, whenever the activity 'x' is executed, it will implicate that in all paths, in the future, the activity 'y' will be not executed". Taking the previous example, it could be verified if the order being reproved by the supervisor, the procurement would never be completed.

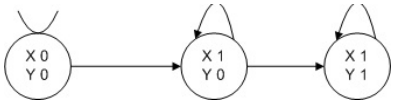

Figure 7. State diagram for "The execution of an activity represents the possibility of the execution of another"

3) The execution of an activity represents the possibility of the execution of another: To test this relation (Figure 7), the following command is used:

SPEC EF((atividadex -> EF(atividadey)) & (E[!atividadey U atividadex]));

It can be translated as: "if the activity 'x' is executed, there is at least one path in which, in the future, the activity 'y' will also be". In the previous example, a rule that could be tested is whether a procurement process, after being reproved by an auditing team, could still be completed. If in that workflow, the auditors could revise the process and approve it, then this rule would be true.

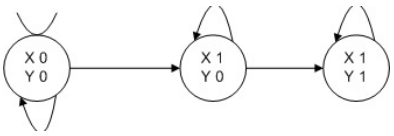

Figure 8. State diagram for "The execution of an activity depends on the previous execution of another"

4) The execution of an activity depends on the previous execution of another: For this rule (Figure 8), the command is:

SPEC !EF(!activityx & activityy) & !EF(!activityx & !activityy & EX(activityx & activityy));

That evaluation is made in two steps. Firstly it tests if there is no path where the activity 'x' is not executed but 'y' is. The second step tests if both activities aren't executed simultaneously. Using these two tests, it is guaranteed that activity 'y' will only be executed after the execution of 'x'.

This validation could be used to verify if activities that should be sequential are disposed in exclusive or parallel paths. Using the procurement example, it could be tested if the approval and payment are necessarily sequential rather than in parallel or exclusive paths.

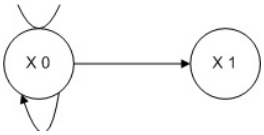

Figure 9. State diagram for "Is possible that a specific activity is not executed"

5) Is possible that a specific activity is not executed: To evaluate this relation type, shown in Figure 9, the following command is used.

SPEC !EG(!activityx);

That command is translated as: "there is a path in which the activity 'x' is not executed". In the procurement example, it could be test if there is a possibility that the activity of evaluation by a supervisor is not executed if the total cost does not reach a certain value.

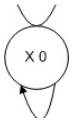

Figure 10. State diagram for "A specific activity is never executed"

6) A specific activity is never executed: This command is used to test the relation shown in Figure 10.

SPEC AG(!activityx);

Through this command it is tested if, in all the paths, the activity 'x' is not executed. That can only happen if an element has no transition directed to it, staying disconnected from the flow. This type of validation is more usable to test errors in the modeling process rather than logical problems.

7) Is always possible to reach the end event: To test this rule is used this command:

SPEC AG(activitya -> AF(activityb));

In that command the activity 'a' is understood as the start event of the flow and 'b' the end event. Therefore, the command is translatable as: "whenever the start event is executed, the end event will also be". Three possibilities can generate a negative answer for this rule: no end event, paths with disconnected elements or uninterrupted cycles.

C. Execution of the translation

The whole translation, including the temporal logic rule to be validated, is then saved in a SMV file that is executed in the model checker. The answer of the validation is captured by the interface, which is then shown to the user. If the rule evaluated if false, then a counter example is received and also shown to the user. The interface (Figure 11) and the translator were created in Java.

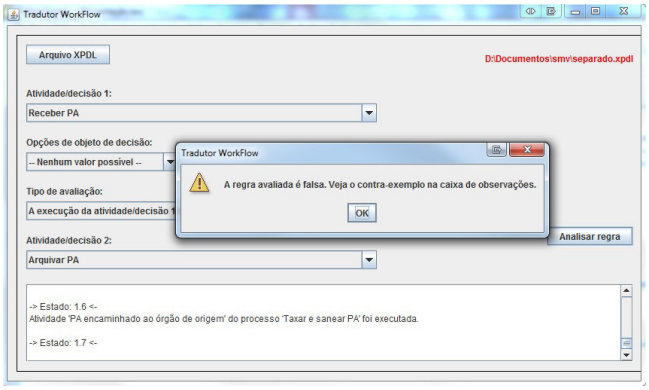

Figure 11. Interface showing a negative answer with a counterexample

IV. Case Study

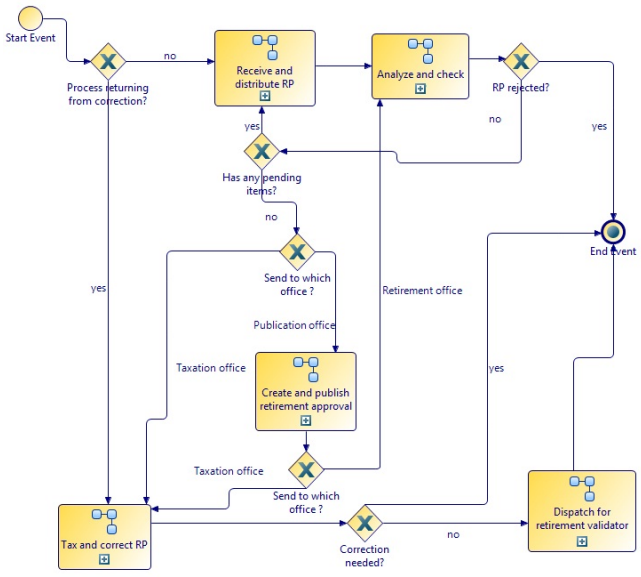

Figure 12. Workflow of case study (abstract view)

The workflow used in the case study was one that models the process of retirement grant to public employees of the State Government of Minas Gerais in Brazil, modeled in BPMN (Figure 12). In the end of the validation of that workflow, 75 elements were considered in total, excluding from that number the transitions among them.

The initial strategy for the validation of the case study was to validate all revisions of a model to compare if the corrections made to them are detectable through the software. Unfortunately, it was not possible to obtain previous versions of the case study. Then, all the seven validation types were tested for all the hypothesis of true or false responses. Some of them are presented as follows.

A. Results

1) The execution of an activity implicates in the execution of another: Take the example in Figure 13. When the rule "The execution of *Receive RP* implicates in the execution of *Distribute RP*" is tested, the answer obtained is true. That answer is correct because, in all the paths after the execution of the first activity, the second is always executed.

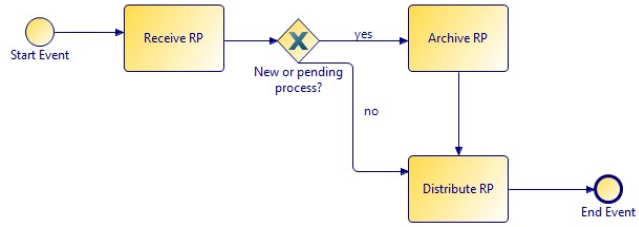

Figure 13. Flow "Receive and distribute RP"

In another test in the same workflow, a negative answer is obtained. The rule "The execution of *Receive RP* implicates in the execution of *Archive RP*" is false because there is an execution sequence in which that second activity is not executed after the first. That possibility derives from the exclusive gateway that exists between those activities.

Whenever the answer for a validation is false, it is shown a counterexample in the interface (Figure 11). The counterexample shows a sequence that is obtainable in the model and that violates the rule tested.

Another possibility of obtaining false for the first validation type is when, between the elements being tested, there is a non-interruptible cycle. That kind of cycle occurs when it is not used the encapsulation property available in BPMN and XPDL, which has also a counter that limits the execution of a giving cycle. The model explained in this article has a limitation to process these uninterrupted cycles. Two cycles exist in the workflow of the case study (Figure 12). Therefore, when tested if the execution of an activity in the beginning of the flow implicates in the execution of an activity in the end, the result is a negative answer from the software. Whenever a negative answer is obtained, a counterexample is given to show an execution sequence in which the rule does not apply.

Finally, the third possibility of obtaining false as an answer for this test occurs when there is no possible paths to reach an activity after the execution of another. That possibility is present in the flow shown in Figure 12. In that flow, it is noticeable that there is not a sequence in which the execution of *Tax and correct RP* implicates in the execution of *Receive and distribute RP*. Therefore, when that rule is tested, the software returns false as answer.

2) The execution of an activity implicates in the non-execution of another: This validation type is not the exact opposite of the last. Negative answers for that validation do not guarantees a positive answer here. That is the case in the validation of the rule involving *Receive RP* and *Archive RP* in Figure 13. When tested in the last validation type, the answer was negative. When tested with the type, the answer is also negative since it is not impossible to execute the second activity after the first. Therefore, negative answers are obtained when there is a sequence in which both activities are executed. The first two tests in the last type were also tested for this validation and, as expected, both got negative answers.

Positive answers are given if the elements tested are in excluding paths. Excluding paths are derived from exclusive gateways not inserted in cycles. Elements after an exclusive gateway but inside a cycle could be executed in sequence even when in excluding paths. The only exclusive gateway not inserted in cycles occurs in the subflow *Tax and correct RP* (Figure 14).

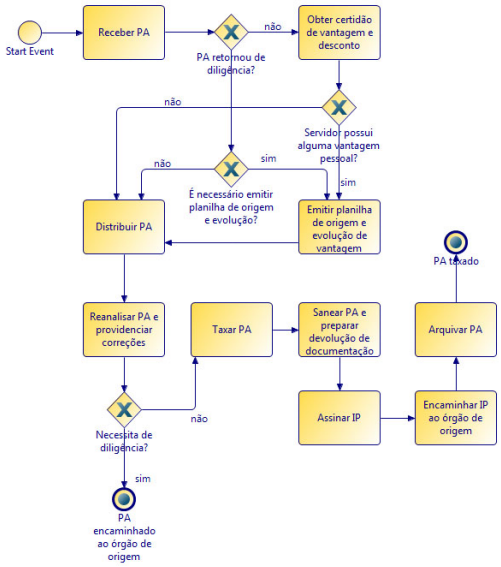

Figure 14. Flow "Tax and correct RP"

When tested the elements in the excluding paths after the gateway in that flow, the answer obtained was true.

3) The execution of an activity represents the possibility of the execution of another: Positive answers for this validation are obtained if there is at least one possibility for both activities to be executed in sequence.

Logically, all positive answers for the first validation type will generate positive answers in this one. And also, all positive answers for the second type will produce negative answers in this one. Therefore, the rules tested in the last validation types that received such answers were again tested. The answers given were all as expected.

It is not correct to conclude that negative answers for the first type would mean negative answers here or also that, negative answers for the second would produce positive answers in this current type. Once again, this affirmative was proved by remaking the tests that produced the referred answers in the first two validation types, obtaining the expected results.

4) The execution of an activity depends on the previous execution of another: This validation can be considered as an specific case of the first type. To obtain positive answers, it is necessary, not only to obtain true in that type, but also that the activities analyzed are not executed simultaneously. Therefore, it is guaranteed that one certain activity will only be executed if another is before.

The first test, between the activities *Receive RP* and *Distribute RP* in Figure 13, was used to test the correctness of the answer for this type. When tested whether the second activity depends on the previous execution of the first, the answer received was true, as expected.

A negative result would occur whenever a negative answer is obtained in the first validation. To test that, the three rules that produced that outcome were again validated, generating the expected negative answers.

The other possibility of obtaining false, even if a positive answer is obtained in the first validation type, occurs when the activities are simultaneously executed. In that case, it was identified an error in the studied workflow. It was detected that in the flow it is possible to publish a retirement process without the signing of a supervisor because these activities were in parallel paths. When these activities were tested in this fourth validation type, the answer was negative.

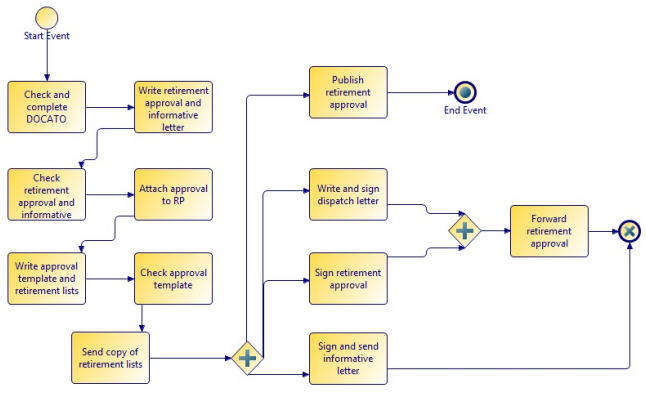

Figure 15. Flow "Create and publish retirement approval"

It is important to highlight that, even after several revisions made by the teams involved in the studied workflow, the software was able to detect this flaw. It is also important to highlight that one only error being detected does not indicate lack of effectiveness of the software but indicates the almost complete correctness of the modeling process.

5) Is possible that a specific activity is not executed: For this validation type, positive answers is obtained if the element analyzed is in excluding path after an exclusive gateway or it is located after an uninterrupted cycle. Testing this type with the activities *Receive RP* and *Archive RP* in Figure 13, it was obtained true because the whole subflow is located in a excluding path after the gateway *Process returning from correction?*.

To obtain negative answers, it is necessary that the element analyzed is executed in all possible sequences. In the studied workflow, that condition applies only to the start and end events, as it can be observed in Figure 12.

6) A specific activity is never executed: In some cases, due to the fact that not all the possibilities for all the validation types were available to be tested in the case study, some controlled errors were inserted. That was the case in this test. This condition only happens if the tested element is disconnected from the flow. If an element has any transition pointing at it, then this rule does not apply.

The activity *Attach approval to RP* in Figure 15 was tested by removing the transition pointing at it. Before removing, the answer obtained was false. After the removal, the answer was true. The same test was done with the activity *Publish retirement approval*, in the same flow. The same answers were obtained before and after the removal of the transition.

7) Is always possible to reach the end event: To obtain a positive answer here, it is necessary that an end event exists, that it is not disconnected and that there is no uninterrupted cycles in the flow. When the studied workflow is tested, the answer is false because uninterrupted cycles exist in the flow. To obtain a positive answer, was necessary to change the workflow to incorporate those cycles into activities with the *Standard Loop* property defining an execution limit to those cycles. With that modification, the desired answer was obtained.

It was also tested the other possibilities for a negative answer. By removing the end event, the software shows an error message, preventing this validation to be run. If, after restoring the end event and removing the transitions to it, the validation is run but the answer is false, as expected.

V. Conclusion

The subject of the verification of business models is getting more attention in the last decade and it has shown that is effectible for early inconsistencies solution that may represent increasing costs to solve. The model checking technique shows itself as adequate for this purpose. So does the tool created in this research to allow a comprehensive evaluation of rules concerning activity execution in workflows.

As a premise to this work, the software should not present an effort of adaptation of the models already created in BPMN or XPDL. The tool should conform to

the definitions of the patterns. Another premise was that the users should not need to know about model checking or the languages to use the software in their flows. The software is intended to be used by business specialist and not system analysts or programmers.

Nowadays, it is common to adapt workflows from a business analysis perspective to an orchestration environment. It is essential to maintain the same business rules from one to the other. In those adapted flows, the approach in this paper can be useful to validate the adherence to those rules.

The tool was able, not only to detect all the hypotheses for all the validation types, involving or not inserted errors, but also detected a error existing in the final version of the case study. By detecting controlled errors and even mistakes in models that were considered to be final versions, the software proved to be useful to those concerned about activity execution business rules. It facilitates the process of verification and offers more exactitude to that kind of analysis.

Some complementary works are viable like including other languages or model checkers, including new kinds of relation between activities, create a feature to simulate behaviors, development of a plugin to adapt this software to a modeling tool, among others.

REFERENCES

[1] T. Takemura, "Formal semantics and verification of bpmn transaction and compensation," *2008 IEEE Asia-Pacific Services Computing Conference*, 2008.

[2] *A guide to the Business Analysis Body of Knowledge (BABOK Guide) - Version 2.0*, International Institute of Business Analysis, 2009.

[3] R. S. Aguilar-Saven, "Business process modelling: Review and framework," *International Journal of Production Economics - Volume 90 - Issue 2 - 28 July 2004*, vol. 90, pp. 129–149, 2004.

[4] A. Andrade, A. Ribeiro, E. Borges, and W. Neves, "Um estudo de aplicacao de modelagem de processo de negocio para apoiar a especificacao de requisitos de um sistema," *VI Simposio Internacional de Melhoria de Processos de Software, Sao Paulo - SP, Brasil. 24-26/11/2004*, 2004.

[5] J. S. Conery, J. M. Catchen, and M. Lynch, "Rule-based workflow management for bioinformatics," *The VLDB Journal - The International Journal on Very Large Data Bases, Volume 14, Issue 3 - September 2005*, 2005.

[6] L. An and J.-J. Jeng, "On developing system dynamics model for business process simulation," *Proceedings of the 2005 Winter Simulation Conference*, 2005.

[7] C.-H. Tsai, H.-J. Luo, and F.-J. Wang, "Constructing a bpm environment with bpmn*," *Proceedings of the 11th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS'07)*, 2007.

[8] OMG, *Business Process Modeling Notation (BPMN) Specification, version 1.2*, Object Management Group (OMG), 01 2009, especificao da notao BPMN. [Online]. Available: www.bpmn.org

[9] WfMC, *XPDL 2.1 Specification*, wfmc-tc-1025-oct-10-08-a ed., The Workflow Management Coalition, 10 2008. [Online]. Available: http://www.wfmc.org/View-document-details/WFMC-TC-1025-Oct-10-08-A-Final-XPDL-2.1-Specification.html

[10] N. Zsifkov and R. Campeanu, "Business rules domains and business rules modeling," *International Symposium on Information and Communication Technologies*, 2004.

[11] M. zur Muehlen, M. Indulska, and G. Kamp, "Business process and business rule modeling languages for compliance management: A representational analysis," *Twenty-Sixth International Conference on Conceptual Modeling - ER*, 2007.

[12] P. Matousek, "Verification of business process models," Departament of Computer Science - University of Ostrava, Tech. Rep., 2003.

[13] A. Haller, W. Gaaloul, and M. Mateusz, "Towards an xpdl compliant process ontology," *2008 IEEE Congress on Services*, 2008.

[14] E. M. Clarke, *Model checking*. MIT Press, 1999.

[15] W. Janssen, R. Mateescu, S. Mauw, P. Fennema, and P. van der Stappen, "Model checking for managers," *Proceedings of the 5th and 6th International SPIN Workshops on Theoretical and Practical Aspects of SPIN Model Checking*, 1999.

[16] J. Khoeler, G. Tirenni, and S. Kumaran, "From business process model to consistent implementation: A case for formal verification methods," *6th International Enterprise Distributed Object Computing Conference (EDOC 2002)*, 2002.

[17] J. A. Fisteus, A. M. Lopez, and C. D. Kloos, "Verbus: A formal model for business process verification," *Information Resources Management Association International Conference. New Orleans, Louisiana, USA*, 2004.

[18] Y.-H. Huang, "Using model checking to verify the consistency between business process models," Department of Accountancy - National Cheng-Kung University, Tech. Rep., 2006.

Mark A. J. Song was born in Belo Horizonte, Brazil. He received the B.S. (1991), M.S. (1996) and Ph.D. (2004) degrees in Computer Science from the Federal University of Minas Gerais, Belo Horizonte, Brazil.

Since 1997 he hold research and teaching position at Pontifical Catholic University of Minas Gerais, Brazil. Professor Song has research interest in formal methods - especially SMC (Symbolic Model Checking).

Cristiano de M. Barros was born in Coronel Fabriciano, Brazil. He received the B.S. in Public Management in 2001, B.S. in Computer Science in 2004 and M.S. in Computer Science in 2010.

He holds a teaching position at State University of Minas Gerais and works as a Specialist in Public Policies and Government Management in the State Government of Minas Gerais.