Verification of Workflow Specifications in UML Using Automated Transformations to WF-nets

Lígia Maria Soares Passos[1], **Tarcísio Abadio de Magalhães Júnior**[1], **Marcelo de Almeida Maia**[1], **Stéphane Julia**[1]

[1]Faculdade de Computação – Universidade Federal de Uberlândia (UFU)
Caixa Postal 593 – 38400-902 – Uberlândia – MG – Brazil

{ligia,marcmaia,stephane}@facom.ufu.br, tarcisiojunior@gmail.com

Abstract. *This article proposes the transformation of UML Activity Diagrams that models workflow processes into WF-nets (WorkFlow nets). The transformation is automated using the ATL transformation language, well-known in the model-driven development context. This transformation enables the verification of the soundness property for workflows using linear logic proofs in WF-net specifications. The generated proof trees also help finding possible causes for unsound diagrams. An illustrative case study is presented to demonstrate the approach effectiveness. The results from this paper could motivate the typical software company to introduce the rigor of formal verification using Petri Nets associated with linear logic without sacrificing the common practice of developers who can continue using UML notation integrated with industrial tools such as ATL.*

1. Introduction

The purpose of Workflow Management Systems [van der Aalst and van Hee 2004] is the execution of workflow processes, which represent sequences of activities executed within an organization to treat specific cases of workflow and to reach a well defined goal. A workflow process is a set of activities that can be executed simultaneously with some control specification and data flow between these activities [Leymann and Roller 1997].

Petri nets [Petri 1962, Murata 1989] are well-suited to model workflow processes and many papers have already considered this theory as an efficient tool for the modeling and analysis of Workflow Management Systems. In [van der Aalst 1998, van der Aalst and van Hee 2004], for example, high level Petri nets are used to model Workflow Management Systems. A Petri net which models a workflow process definition (i.e. the life-cycle of one case in isolation) is called WorkFlow net (WF-net).

The UML (Unified Modeling Language) Activity Diagram is a special case of statechart [Tričkovié 2000], such that all states are action states and transitions are fired by the finalization of its source states.

An approach to analyse workflow process using model transformation is presented in [Meena et al. 2005]. The authors propose the transformation of UML Activity Diagrams into Petri nets, but they just present an example with no formalization of transformation rules. In [Gehrke et al. 1998], the authors propose the transformation of an UML Activity Diagram into a Petri net, presenting just some schemas of transformation without presenting the generic rules for this transformation. In [Tričkovié 2000], a formalization

of UML Activity Diagrams semantics is proposed using Petri nets. However, the author just shows some transformation examples and the transformation rules are not given. In [Störrle and Hausmann 2005], the semantics of UML Activity Diagrams is proposed in an intuitive way, but the transformation rules are neither formalized nor sufficient to achieve the complete transformation because two additional rules for a complete transformation were not presented in that work.

There are several reasons to use Petri nets in workflow process modeling: formal semantics, expressiveness and many available analysis techniques [van der Aalst 1998]. The use of such formalism has many important advantages, because the use of a precise definition prevents ambiguity, uncertainty and contradictions [van der Aalst and van Hee 2004] - in contrast with most of other techniques of informal diagramming, for instance, UML Activity Diagrams [Dumas and ter Hofstede 2001]. Despite the good reasons to use Petri nets in the context of workflow process modeling, a resistance to this formal method is widely observed in the industry, which prefers to model workflow processes using UML Activity Diagrams or other semi-formal diagram.

In the context of Model-driven Engineering (MDE), models are the main development artifacts and the model transformations are important operations applied to the models [Jouault et al. 2008]. There are many available transformation languages, such as the ATLAS Transformation Language (ATL), which is developed by the ATLAS Group (INRIA & LINA). In the context of MDE, ATL provides an alternative to produce a set of target models from a source models set [Atlas_Group].

In this paper, an approach based in ATL is proposed to transform an UML Activity Diagram (that models workflow processes) into WF-nets [van der Aalst and van Hee 2004],[van der Aalst 1998], that is, a Petri net that models the same workflow process modeled by the UML Activity Diagram. Moreover, an approach to verify workflow process is also proposed.

In Section 2 workflow processes, UML Activity Diagrams and WF-nets are introduced. Section 3 presents the ATLAS Transformation Language. The transformation rules between the models are presented in Section 4. Section 5 presents the linear logic and the soundness verification to WF-nets. In Section 6 a case study is presented. Finally, the last section concludes this work with a short summary, an assessment about the approach presented and an outlook on the future work.

2. Workflow Process, UML Activity Diagrams and WF-nets

A workflow definition includes a process and its corresponding tasks that must be performed, in a specific order, to successfully complete a case. In other words, all possible routes are mapped out [van der Aalst and van Hee 2004]. Each task can be optional, i.e. there are tasks that just need to be executed for some cases and the order in which tasks will be executed can vary from case to case [van der Aalst and van Hee 2004]. Four basic constructions for routing are presented in [van der Aalst and van Hee 2004] and [van der Aalst 1998]: sequential, parallel, conditional and iterative.

The modeling elements of UML Activity Diagrams are defined in the OMG UML Superstructure [OMG 2008] and also can be used to model workflow processes. Fig. 1 shows the modeling elements used in this paper. The other elements are not considered

in this approach. Moreover, in this work, a restriction is imposed in the UML Activity Diagram, such that just one Initial Node and just one Final Node can appear, and all the activities (Action Nodes) need to have, at least, one outgoing edge (Control Flow).

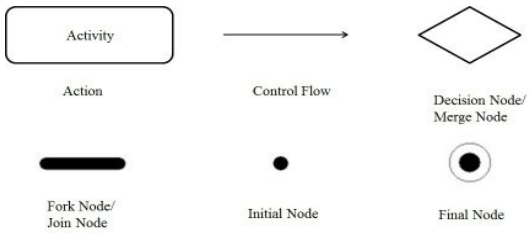

Figure 1. UML Activity Diagram Modeling Elements.

A Petri net that models a workflow process is called a WorkFlow net (WF-net) [van der Aalst and van Hee 2004, van der Aalst 1998]. A WF-net satisfies the following properties [van der Aalst 1998]:

- It has only one source place named *Start* and only one sink place named *End*. These are special places, such that the place *Start* has only outgoing arcs and the place *End* has only incoming arcs.
- A token in *Start* represents a case that needs to be handled and a token in *End* represents a case that has been handled.
- Every task t (transition) and condition p (place) should be in a path from place *Start* to place *End*.

Soundness is a correctness criterion defined for WF-nets. A WF-net is sound if, and only if, the following three requirements are satisfied [van der Aalst 1998, van der Aalst et al. 2011]:

- For each token in the place *Start*, one and only one token appears in place *End*.
- When the token appears in place *End*, all the other places are empty for this case.
- For each transition (task), it is possible to move from the initial state to a state in which that transition is enabled, i.e. there is not any dead transition.

The modeling elements of the WF-nets is shown in Fig. 2.

Figure 2. WF-nets Modeling Elements.

2.1. UML Activity Diagrams and WF-nets Metamodels

The UML Activity Diagrams and WF-nets metamodels will be used as the basis for the transformation between the respective models. Fig. 3 shows graphically the metamodel of the UML Activity Diagrams. The metamodel of the WF-nets is shown in Fig. 4.

The main difference between the UML Activity Diagrams metamodel proposed by the OMG [OMG 2008] and the proposed in this paper is that this approach does not

consider the object nodes and the object flows, which are present in the metamodel proposed by the OMG.

The WF-net metamodel proposed in this work comprehends all modeling elements proposed in [van der Aalst 1998].

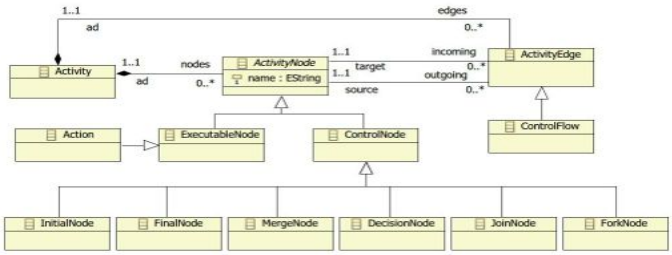

Figure 3. UML Activity Diagram Metamodel.

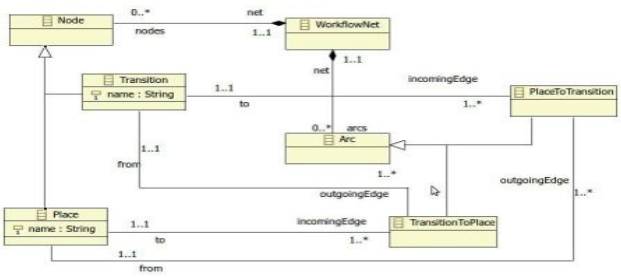

Figure 4. WF-net Metamodel.

3. ATLAS Transformation Language

The ATLAS transformation language – ATL is a hybrid model transformation DSL containing a mixture of declarative and imperative constructs. ATL transformations are unidirectional, i.e, source models are read-only and target models are write-only. During the execution of a transformation, source models may be navigated but changes are not allowed. Target models cannot be navigated. A module contains a mandatory header section and a number of transformation rules. The header section gives the name of a transformation module and declares the source and target models. The source and target models are typed by their metamodels. Declarative ATL rules are called matched rules. They specify relations between source patterns and target patterns. The name of a rule is given after the keyword rule. The source pattern of a rule specifies a set of source types and an optional guard given as a Boolean expression in OCL. A source pattern is evaluated to a set of matches in source models. The target pattern is composed of a set of elements. Each of these elements specifies a target type from the target metamodel and a set of bindings. A binding refers to a feature of the type (i.e. an attribute, a reference or an association end) and specifies an expression whose value is used to initialize the feature. In some cases, complex transformation algorithms may be required and it may be difficult to specify them in a declarative way. For this issue, ATL provides imperative constructs, but in this approach only declarative rules were used.

4. Transformation Rules

In this section, the transformation rules used to transform an UML Activity Diagram into a WF-net are presented. Some of these rules were presented in [Störrle and Hausmann 2005]. However, the proposed rules are not sufficient to complete all proposed transformation. Thus, two new rules are presented in this paper and they are highlighted in Fig. 5 with a dotted rectangle. Fig. 5 depicts the transformation rules intuitively. For example, the modeling element *Action* of the UML Activity Diagram will be transformed into the modeling element *Transition* of the WF-net, the modeling element Final Node of the UML Activity Diagram will be transformed into the modeling element Place of the WF-net, and so on.

A portion of ATL code for the transformation rules is shown in the Appendix.

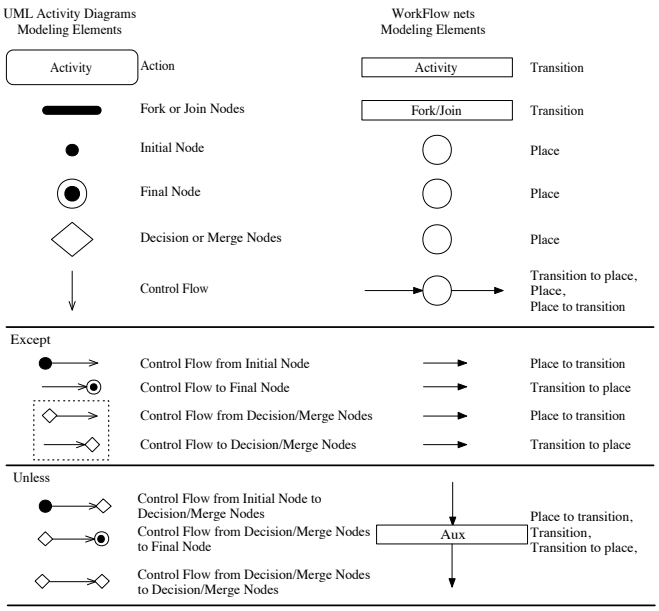

Figure 5. Transformation Rules.

A process for handling complaints, shown in [van der Aalst and van Hee 2004], can be used to illustrate the proposed transformation: *"An incoming complaint first is recorded. Then the client who has complained and the department affected by the complaint are contacted. The client is approached for more information. The department is informed of the complaint and may be asked for its initial reaction. These two tasks may be performed in parallel, i.e. simultaneously or in any order. After this, the data are gathered and a decision is taken. Depending upon the decision, either a compensation payment is made or a letter is sent. Finally, the complaint is filed"*. The UML Activity Diagram that models this workflow process is shown in Fig. 6*(a)*. Fig. 6*(b)* shows the correspondent WF-net, which can be generated when applying the proposed transformation rules to the input UML Activity Diagram.

5. Linear Logic and Soundness Verification of WF-nets

5.1. Linear Logic and Petri nets

Linear Logic was proposed in 1987 by Girard [Girard 1987]. In Linear Logic, propositions are considered as resources which are consumed and produced at each state change

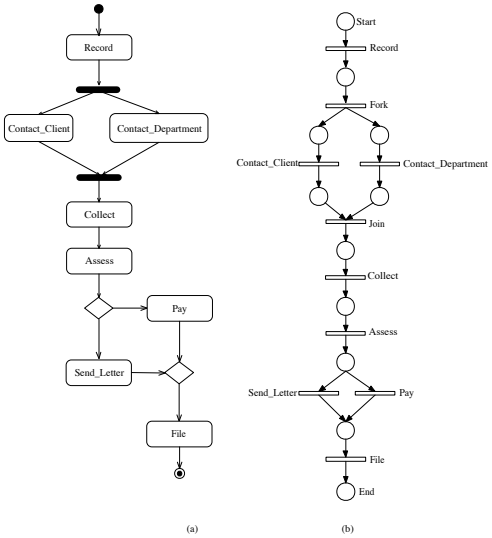

Figure 6. The process "handle complaints". *(a)*Workflow Process modeled through an UML Activity Diagram. *(b)* Workflow Process modeled through a WF-net.

[Diaz 2009]. Linear Logic introduces new connectives, as the connectives *"times"*($\otimes$) and *"linear implies"*($\multimap$), used in this paper, such that:

- The *times* connective, denoted by $\otimes$, represents simultaneous availability of resources. For instance, $A \otimes B$ represents the simultaneous availability of resources A and B.
- The *linear implies* connective, denoted by $\multimap$, represents a state change. For instance, $A \multimap B$ denotes that consuming A, B is produced (note that after the production of B, A will not be available).

The translation of a Petri net into formulas of Linear Logic is presented in [Riviere et al. 2001] and [Diaz 2009]. A marking M is a monomial in $\otimes$ and is represented by $M = A_1 \otimes A_2 \otimes \ldots \otimes A_k$ where A_i are place names. A transition is an expression of the form $M_1 \multimap M_2$ where M_1 and M_2 are markings. A sequent $M, t_i \vdash M'$ represents a scenario where M and M' are respectively the initial and final markings, and t_i is a list of non-ordered transitions.

In this paper, just some rules of Linear Logic will be considered. These rules will be used to build the proof trees presented in this approach. For this, F, G, and H are considered as formulas and Γ and Δ as blocks of formulas. The following rules will be the ones used in this paper [Riviere et al. 2001]:

- The $\multimap_L$ rule, $\dfrac{\Gamma \vdash F \quad \Delta, G \vdash H}{\Gamma, \Delta, F \multimap G \vdash H} \multimap_L$, expresses a transition firing and generates two sequents such that the right sequent represents the subsequent remaining to be proved and the left sequent represents the consumed tokens by this firing.
- The $\otimes_L$ rule, $\dfrac{\Gamma, F, G \vdash H}{\Gamma, F \otimes G \vdash H} \otimes_L$ transforms a marking in an atoms list.
- The $\otimes_R$ rule, $\dfrac{\Gamma \vdash F \quad \Delta \vdash G}{\Delta, \Gamma \vdash F \otimes G} \otimes_R$ transforms a sequent such $A, B \vdash A \otimes B$ into two identity sequents $A \vdash A$ and $B \vdash B$.

In the approach presented in this paper, a Linear Logic proof tree is read from bottom-up. The proof stops when the atom End is produced, i.e. the identity sequent $End \vdash End$ appears in the proof tree, when there is not any rule that can be applied or when all the leaves of the proof tree are identity sequents. This proof is decidable once each transition of the WorkFlow net appears at the most once in the linear sequent. Therefore, if the number of transitions is finite, the construction of the proof tree is decidable [Passos and Julia 2009].

5.2. Soundness Verification for WF-nets

In the context of WF-nets, the main aim of qualitative analysis is to prove Soundness property, once this criterion ensures the correctness of a workflow process mapped into a WF-net.

To prove the Soundness correctness criterion for a WF-net using Linear Logic [Passos and Julia 2009, Soares Passos and Julia 2009], initially, it is necessary to represent this WorkFlow net by formulas of Linear Logic. A WF-net is represented by one or more linear sequents. A scenario in the context of WF-nets corresponds to a well defined route mapped into the WF-net and if the WF-net has more than one route, it is then necessary to build a different linear sequent for each existing scenario. After the definition of the linear sequents that represent the WF-net, these linear sequents need to be proved. To prove these sequents, proof trees of Linear Logic will be produced. After the construction of the proof trees each scenario of the analysed WorkFlow net must be analysed respecting the following steps:

1. For each proof tree:
   (a) If just one atom End was produced in the proof tree (this is represented in the proof tree by the identity sequent $End \vdash End$), then the first requirement for Soundness is proved, i.e just one token appears in the place *End*.
   (b) If there is not any available atom for consumption on the proof tree, then it means that all places are empty, i.e. the second requirement for Soundness is proved.
   (c) There is not any available transition formula in the proof tree that was not fired.
2. Considering all scenarios $Sc_1, Sc_2, ..., Sc_n$ for the WorkFlow net analysed, each transition $t \in T$ needs to appear in, at least, one scenario. This proves that all transitions were fired (i.e., there is not any dead transition), i.e. the third requirement for Soundness is verified.

If the conditions 1 and 2 above are satisfied, the WorkFlow net is *Sound*.

6. Case Study

In this section, a case study of a transformation is presented. An UML Activity Diagram that models a workflow process is transformed into a WF-net. The workflow process analysed is based on a standard part selection workflow within an airline design process [OMG 2008]. The *Expert Part Search* behavior can result in a found part or not. When a part is not found, it is assigned to the *Assign Standards Engineer* activity. Lastly, *Schedule Part Mod Workflow* invocation produces entire activities and they are passed to subsequent

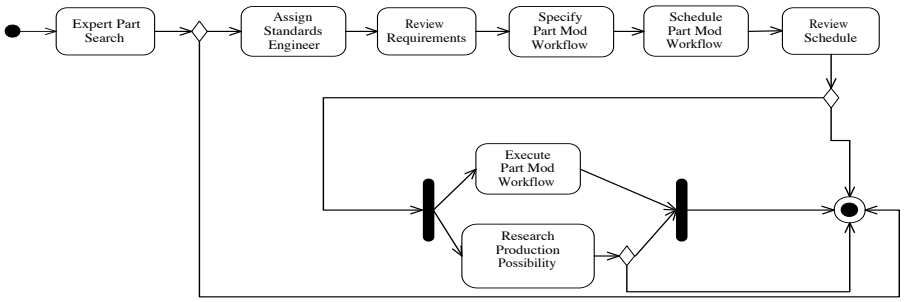

Figure 7. Activity Diagram that models the Workflow Process.

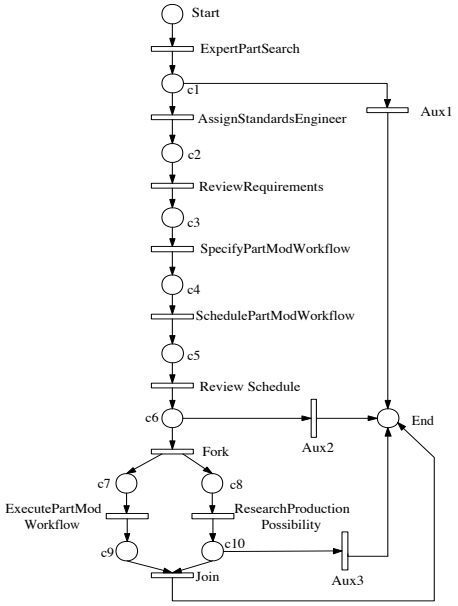

Figure 8. WF-net that models the Workflow Process.

invocations for scheduling and execution (i.e., *Schedule Part Mod Workflow*, *Execute Part Mod Workflow*, and *Research Production Possibility*). The UML Activity Diagrams that models this process is shown in Fig. 7. Fig. 8 shows the corresponding WF-net.

There are four different scenarios in the WF-net shown in Fig. 8: the first scenario, Sc_1, where the task *Aux1* will be executed (firing the transition *Aux1*), the second scenario, Sc_2, where task *Aux2* will be executed, the third scenario, Sc_3, where task $Join$ will be executed, and the fourth scenario, Sc_4, where task *Aux3* will be executed. Thus, for this WF-net, four linear sequents will have to be proved. For scenario Sc_1, it is necessary to prove the sequent $Start, t_1, t_{11} \vdash End$, for scenario Sc_2 the sequent $Start, t_1, t_2, t_3, t_4, t_5, t_6, t_{12} \vdash End$, for scenario Sc_3 the sequent $Start, t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10} \vdash End$ and for scenario Sc_4 the sequent $Start, t_1, t_2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{13} \vdash End$ needs to be proved, considering that:
$t_1 = ExpertPartSearch = Start \multimap c1$,
$t_2 = AssignStandardsEngineer = c1 \multimap c2$,
$t_3 = ReviewRequirements = c2 \multimap c3$,
$t_4 = SpecifyPartModWorkflow = c3 \multimap c4$,
$t_5 = SchedulePartModWorkflow = c4 \multimap c5$,

$t_6 = ReviewSchedule = c5 \multimap c6$,
$t_7 = Fork = c6 \multimap c7 \otimes c8$,
$t_8 = ExecutePartModWorkflow = c7 \multimap c9$,
$t_9 = ResearchProductionPossibility = c8 \multimap c10$,
$t_{10} = Join = c9 \otimes c10 \multimap End$, $t_{11} = Aux1 = c1 \multimap End$,
$t_{12} = Aux2 = c6 \multimap End$, $t_{13} = Aux3 = c10 \multimap End$.

Following are shown the proof trees for scenarios Sc_1, Sc_2, Sc_3 and Sc_4.

Proof tree for Scenario Sc_1:

$$\dfrac{\dfrac{}{Start \vdash Start} \quad \dfrac{}{c1 \vdash c1} \quad \dfrac{}{End \vdash End}}{Start, Start \multimap c1, c1 \multimap End \vdash End}$$

Proof tree for Scenario Sc_2:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{Start \vdash Start} \quad \dfrac{}{c1 \vdash c1} \quad \dfrac{}{c2 \vdash c2} \quad \dfrac{}{c3 \vdash c3} \quad \dfrac{}{c4 \vdash c4} \quad \dfrac{}{c5 \vdash c5} \quad \dfrac{}{c6 \vdash c6} \quad \dfrac{}{End \vdash End}}{Start, Start \multimap c1, t_2, t_3, t_4, t_5, t_6, t_{12} \vdash End}}{c1 \multimap c2, t_3, t_4, t_5, t_6, t_{12} \vdash End}}{c2 \multimap c3, t_4, t_5, t_6, t_{12} \vdash End}}{c3 \multimap c4, t_5, t_6, t_{12} \vdash End}}{c4 \multimap c5, t_6, t_{12} \vdash End}}{c5 \multimap c6, t_{12} \vdash End}}{c6 \multimap End \vdash End}$$

Proof tree for Scenario Sc_3:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{Start \vdash Start} \quad \dfrac{}{c1 \vdash c1} \quad \dfrac{}{c2 \vdash c2} \quad \dfrac{}{c3 \vdash c3} \quad \dfrac{}{c4 \vdash c4} \quad \dfrac{}{c5 \vdash c5} \quad \dfrac{}{c6 \vdash c6} \quad \dfrac{\dfrac{}{c9 \vdash c9} \quad \dfrac{}{c10 \vdash c10}}{c9, c10 \vdash c9 \otimes c10} \otimes_R \quad \dfrac{}{End \vdash End}}{c7, c8, c7 \multimap c9, t_9, t_{10} \vdash End} \otimes_L}{c8 \vdash c8 \quad c9, c10, c9 \otimes c10 \multimap End \vdash End}}{c7 \vdash c7 \quad c8, c9, c8 \multimap c10, t_{10} \vdash End}}{c6 \vdash c6 \quad c7 \otimes c8, t_8, t_9, t_{10} \vdash End}}{c5 \vdash c5 \quad c6, c6 \multimap c7 \otimes c8, t_8, t_9, t_{10} \vdash End}}{c4 \vdash c4 \quad c5, c5 \multimap c6, t_7, t_8, t_9, t_{10} \vdash End}}{c3 \vdash c3 \quad c4, c4 \multimap c5, t_6, t_7, t_8, t_9, t_{10} \vdash End}}{c2 \vdash c2 \quad c3, c3 \multimap c4, t_5, t_6, t_7, t_8, t_9, t_{10} \vdash End}}{c1 \vdash c1 \quad c2, c2 \multimap c3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10} \vdash End}}{Start \vdash Start \quad c1, c1 \multimap c2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{10} \vdash End}$$

Proof tree for Scenario Sc_4:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{Start \vdash Start} \quad \dfrac{}{c1 \vdash c1} \quad \dfrac{}{c2 \vdash c2} \quad \dfrac{}{c3 \vdash c3} \quad \dfrac{}{c4 \vdash c4} \quad \dfrac{}{c5 \vdash c5} \quad \dfrac{}{c6 \vdash c6} \quad \dfrac{}{c7 \otimes c8, t_8, t_9, t_{13} \vdash End}}{c7, c8, c7 \multimap c9, t_9, t_{13} \vdash End} \otimes_L}{c8 \vdash c8 \quad c9, c10, c10 \multimap End \vdash End}}{c7 \vdash c7 \quad c8, c9, c8 \multimap c10, t_{13} \vdash End}}{c6 \vdash c6 \quad c7 \otimes c8, t_8, t_9, t_{13} \vdash End}}{c5 \vdash c5 \quad c6, c6 \multimap c7 \otimes c8, t_8, t_9, t_{13} \vdash End}}{c4 \vdash c4 \quad c5, c5 \multimap c6, t_7, t_8, t_9, t_{13} \vdash End}}{c3 \vdash c3 \quad c4, c4 \multimap c5, t_6, t_7, t_8, t_9, t_{13} \vdash End}}{c2 \vdash c2 \quad c3, c3 \multimap c4, t_5, t_6, t_7, t_8, t_9, t_{13} \vdash End}}{c1 \vdash c1 \quad c2, c2 \multimap c3, t_4, t_5, t_6, t_7, t_8, t_9, t_{13} \vdash End}}{Start \vdash Start \quad c1, c1 \multimap c2, t_3, t_4, t_5, t_6, t_7, t_8, t_9, t_{13} \vdash End}$$

Considering the proof trees for scenarios Sc_1, Sc_2, Sc_3 and Sc_4, the WF-net shown in Fig. 8 is not sound, because the condition 1(b) for *soundness* is not satisfied, i.e., there is a available atom for consumption in the proof tree for scenario Sc_4, c9. This fact shows that when the token appears in the place *End*, there is still a place that is not empty for this case, which means that the process can finish and still have an activity that is executing. Thus, the second requirement for *soundness* is not satisfied.

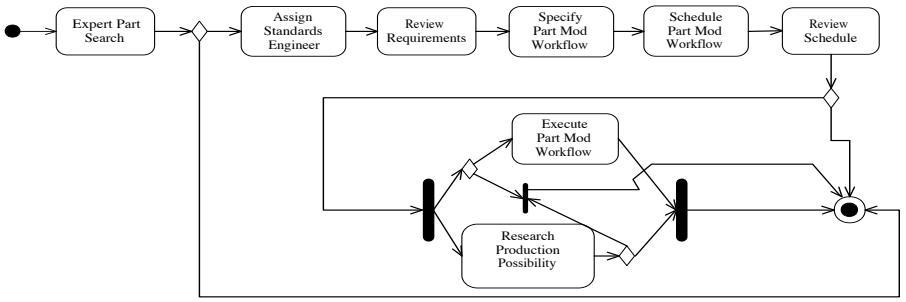

Figure 9. New UML Activity Diagram that models the Workflow Process.

6.1. Finding and Debugging Critical Points in UML Activity Diagrams using proof trees of Linear Logic

Some critical points can be found out in the workflow process analysing the proof trees of linear logic, which were built to prove the soundness property. For example, if there is a transition formula of linear logic that was not fired in any scenario, it means that this transition is dead and it is not necessary in the workflow process, because the correspondent activity never will be executed. So, some correction can be made, removing this activity, in the UML Activity Diagram to turn the correspondent WF-net sound. Another situation is when the construction of the proof tree finished and there is an available atom for consumption in this proof tree: it indicates that there is a critical point in the region of the place represented by the atom. For an illustrative example, consider the proof tree for scenario Sc_4. Note that there is an available atom c9 in this proof tree. It indicates that in the region of place c9 of the WF-net shown in Fig. 8 there is some modeling error. So, the analyst can study and correct this correspondent region in the UML Activity Diagram, preserving the necessary semantic. Fig. 9 shows an UML Activity Diagram with a possible correction for this example. This correction determines that the process can finish only if both the activities *Research Production Possibility* and *Execute Part Mod Workflow* have already finished or if the activity *Research Production Possibility* has already finished and the activity *Execute Part Mod Workflow* has not started. The updated UML Activity Diagram can be transformed into a corresponding WF-net and this one can be analysed, constructing new proof trees. The corresponding WF-net is sound and it is shown in Fig. 10.

7. Conclusion

This work presented formal rules to transform UML Activity Diagrams that models workflow process into WF-nets. These rules were implemented with the ATL transformation language. Although [Störrle and Hausmann 2005] has proposed a kind of transformation related with the transformation proposed in this paper, that work has neither presented a complete set of transformations, nor shown the feasibility of proving soundness for the generated diagrams.

One of the advantages of this approach is the automated transformation of a semi-formal model into a formal one, in which some properties, can be formally verified. Another advantage is the finding and debugging of critical points in UML Activity Diagrams, based on linear logic proof trees, that allows the analysis and correction of the initial semi-formal model.

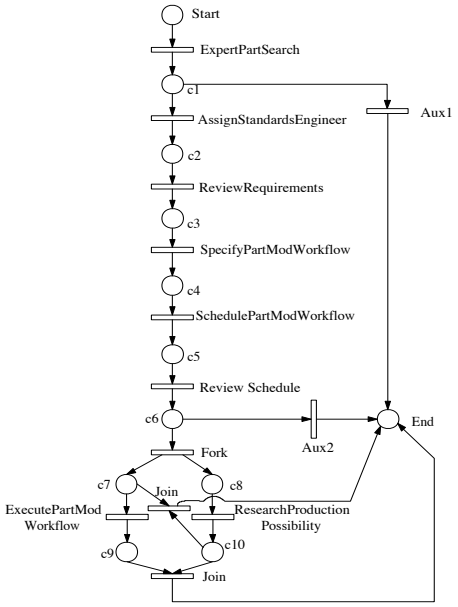

Figure 10. Sound WF-net that models the Workflow Process.

As a future work, it would be interesting to enhance the software support to permit an UML Activity Diagram be modelled graphically and to show the correspondent WF-net graphically too. It also would be interesting to extend the metamodel of the UML Activity Diagrams presented in this paper considering, for example, the object nodes and to propose a new set of transformation rules to some high-level Petri net, permitting the verification of other interesting properties.

References

Atlas_Group. ATL: Atlas Transformation Language. `http://www.eclipse.org/m2m/atl/`.

Diaz, M. (2009). *Petri Nets: Fundamental Models, Verification and Applications*. Wiley-IEEE Press.

Dumas, M. and ter Hofstede, A. H. M. (2001). UML activity diagrams as a workflow specification language. In *Proc. of the 4th Intl. Conf. on The Unified Modeling Language*, pages 76–90, London, UK. Springer-Verlag.

Gehrke, T., Goltz, U., and Wehrheim, H. (1998). The dynamic models of UML: Towards a semantics and its application in the development process. Technical report, Institut für Informatik, Universität Hildeshein.

Girard, J.-Y. (1987). Linear logic. *Theoretical Computer Science*, 50(1):1–102.

Jouault, F., Allilaire, F., Bézivin, J., and Kurtev, I. (2008). ATL: A model transformation tool. *Science Compututer Programming*, 72(1-2):31–39.

Leymann, F. and Roller, D. (1997). Workflow-based applications. *IBM Systems Journal*, 36(1):102–123.

Meena, H. K., Saha, I., Mondal, K. K., and Prabhakar, T. V. (2005). An approach to workflow modeling and analysis. In *Eclipse'05: Proceedings of the 2005 OOPSLA Workshop on Eclipse technology eXchange*, pages 85–89, New York, NY, USA. ACM.

Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580.

OMG (2008). *OMG Unified Modeling Language Specification – Version 2.2*. Object Management Group. `http://www.omg.org/spec/UML/2.2/Superstructure`.

Passos, L. M. S. and Julia, S. (2009). Qualitative analysis of workflow nets using linear logic: soundness verification. In *Proceedings of the 2009 IEEE international conference on Systems, Man and Cybernetics*, SMC'09, pages 2843–2847, Piscataway, NJ, USA. IEEE Press.

Petri, C. A. (1962). *Kommunikation mit Automaten*. PhD thesis, Institut für instrumentelle Mathematik, Bonn.

Riviere, N., Valette, R., Pradin-Chezalviel, B., and Ups, I. A. . (2001). Reachability and temporal conflicts in t-time Petri nets. In *PNPM '01: Proc. of the 9th Intl. Workshop on Petri Nets and Performance Models (PNPM'01)*, page 229, Washington, DC. IEEE.

Soares Passos, L. M. and Julia, S. (2009). Análise qualitativa e quantitativa de workflow nets utilizando lógica linear. In *SBSI 2009: Proceedings of the V Simpósio Brasileiro de Sistemas de Informação*. SBC.

Störrle, H. and Hausmann, J. H. (2005). Towards a formal semantics of UML 2.0 activities. In Liggesmeyer, P., Pohl, K., and Goedicke, M., editors, *Software Engineering*, volume 64 of *LNI*, pages 117–128. GI.

Tričković, I. (2000). Formalizing activity diagram of UML by Petri nets. *Novi Sad Journal of Mathematics*, 30(3):161–171.

van der Aalst, W. M. P. (1998). The application of Petri nets to Workflow Management. In *The Journal of Circuits, Systems and Computers*, pages 21–66.

van der Aalst, W. M. P. and van Hee, K. (2004). *Workflow Management: Models, Methods, and Systems*. The MIT Press.

van der Aalst, W. M. P., van Hee, K. M., ter Hofstede, A. H. M., Sidorova, N., Verbeek, H. M. W., Voorhoeve, M., and Wynn, M. T. (2011). Soundness of workflow nets: classification, decidability, and analysis. *Form. Asp. Comput.*, 23:333–363.

Fragment of ATL code for mapping UML Activity Diagrams into WF-nets

```
1  OUT: WorkflowNet from IN :ActivityDiagram;
2  AD2WN = ad: ActivityDiagram -> wn: WorkflowNet
3  Action2Transition =
4        a: Action -> t : Transition ( name <- a.name, net <- a.ad)
5  DecisionNode2Place = dn: DecisionNode -> p: Place (name <- dn.name, net <- dn.ad)
6  MergeNode2Place = mn: MergeNode -> p: Place (name <- mn.name, net <- mn.ad)
7  ControlFlow = cf: ControlFlow
8         -> t2p: TransitionToPlace (from  <- cf.source, net <- cf.ad)
9         -> p2t:  PlaceToTransition (to <- cf.target, net <- cf.ad)
10        -> p: Place (name <- '')
11     where  cf.source.Type() <> InitialNode  and cf.target.Type() <> FinalNode and
12        cf.target.Type() <> DecisionNode and cf.source.Type() <> DecisionNode and
13        cf.target.Type() <> MergeNode and cf.source.Type() <> MergeNode)
14     do t2p.to <- p; p2t.from <- p;  p.incomingEdge <- t2p; p.outgoingEdge <- p2t;
15        p.net  <- thisModule.globalWorkflowNet;
```