Petri Nets Based Approach for Modular Verification of SysML Requirements on Activity Diagrams

Messaoud Rahim[1], Malika Boukala-Ioualalen[2], and Ahmed Hammad[1]

[1] FEMTO-ST Institute, UMR CNRS 6174, Besançon, France.
{Lastname.firstname}@femto-st.fr
[2] MOVEP, Computer Science department, USTHB, Algiers, Algeria.
mioualalen@usthb.dz

Abstract. The validation of SysML specifications needs a complete process for extracting, formalizing and verifying SysML requirements. Within an overall approach which considers an automatic verification of SysML designs by translating both requirement and behavioral diagrams, this paper proposes a modular verification of SysML functional requirements on activity diagrams. The contribution of this paper is the proposition of a methodology guided by the relationships between requirements and SysML activities for verifying complex systems with many components. We propose a model-to-model transformation to automatically derive from SysML activities a modular Petri net, then SysML requirements are formalized and verified using the derived Petri net modules. A case study is presented to demonstrate the effectiveness of the proposed approach.

Keywords: SysML, Activity Diagram, SysML Requirements, Requirements Formalization, Modular Verification, Petri nets

1 Introduction

Model-based systems engineering is becoming a promising solution to design complex systems. SysML (System Modeling Language) [1] is a standard modeling language which has been proposed to specify systems that include heterogeneous components. It covers four perspectives on system modeling : structure, behavior, requirement, and parametric diagrams. Particularly, the SysML requirement diagram is used for better organizing requirements at different levels of abstraction, allowing their representation as model elements, and showing explicitly the various kinds of relationships between requirements and design elements [2]. However, one of the main challenge in system design is to ensure that a model meets its requirements. To provide a validation of SysML specifications, existing approaches [3–5] propose to translate SysML behavioral models into formal specification languages, then they verify temporal properties by using model-checking techniques. These approaches ignore systems composition and do not relate system requirements to design elements. The activity diagram is one of

SysML models used to specify the system behavior and where requirements can be verified. Based on using the call behavior action concept, a modular design of complex systems can be obtained by structuring its behaviour in many activities. This provides a compositional specification and enables modular analysis of the specified systems [6]. Requirements can be expressed as properties to verify by an activity diagram during its execution. Unfortunately, the need for formal specifications of properties expressed using logics or automata is a major obstacle for the adoption of formal verification techniques by SysML practitioners [7]. The contribution of this paper is the proposition of a methodology which provides a modular verification of functional SysML requirements captured by activity diagrams. It consists on: *(1)* performing a compositional translation from SysML activity diagrams into modular Petri nets where modular PNML [8] is used as target language. *(2)* proposing a new language (AcTRL : Activity Temporal Requirement Language) to express functional requirements related to activities and showing how AcTRL expressions can be automatically translated into properties expressed as temporal logic formulas. Finally, *(3)* presenting a modular verification algorithm. The compositional translation enables the modular verification by considering the decomposition of activity diagrams into sub-activities and the use of AcTRL avoids the specification of SysML requirements directly as properties of the formal semantic model (Petri nets in our case).
This paper is organized as follows. Section 2 surveys related works. Section 3 presents related concepts. Section 4, introduces our overall methodology. In Section 5, we present a compositional translation from SysML activities to modular Petri nets. In Section 6, we define AcTRL and its grammar. An algorithm for modular verification of requirements will be presented in section 7. In Section 8, we illustrate our approach by a case study. Finally, in Section 9, we conclude and we outline future works.

2 Related Work

Ensuring the correctness of complex and critical systems needs automated approaches for verifying and validating their designs. In [3], authors propose to derive for each SysML behavioral diagram a formal semantic model reflecting its characteristics. In this work, requirements was expressed as temporal properties on the formal semantic model which makes the verification process difficult for SysML practitioners. Linhares et al [9] designed a process for verifying SysML specification by considering block, activity and requirement diagrams and where requirements must be expressed using Linear Temporal Logic(LTL). In [4], authors propose TEPE, a graphical temporal property expression language based on SysML parametric diagram to express system requirements. This work is restricted to state machine diagrams. Regarding activity diagram, a symbolic model checking was proposed in [10], where activity diagram is translated into SMV and the NuSMV model checker was used to verify LTL properties. Data flows were not considered in this work. In [11], the authors present a technique to map the SysML activities to time Petri net for validating the requirements of

real-time systems with energy constraints. This work considers non functional requirements. The work presented in [12] proposes a model-driven engineering approach for simulating SysML activity diagram using Petri net and VHDL-AMS. This work focuses on defining rules to translate SysML diagram elements to Petri net specification but it does not consider compositional structure of activity diagrams. To our knowledge, the present work is the first that considers a modular verification of SysML requirements by taking into account their relations to activities.

3 Preliminaries

In this section, we present SysML requirement and activity diagrams as described in the OMG standard [1]. In addition, we describe modular PNML language [8] for representing modular Petri nets.

3.1 SysML requirement diagram

Requirements in SysML are defined in an informal way with an identifier and a text. Requirement diagrams are used for specifying requirements and to depict their hierarchy and the exiting relationships between them and other SysML models. As depicted in Figure 1, the <<Verify>> relationship is a dependency between a requirement and a test case that can determine whether a system fulfills the requirement [1]. The <<deriveReqt>> relationship is a dependency between two requirements. It is used to derive a requirement from another. Other relationships exists, we refer to [1] for a detailed description.

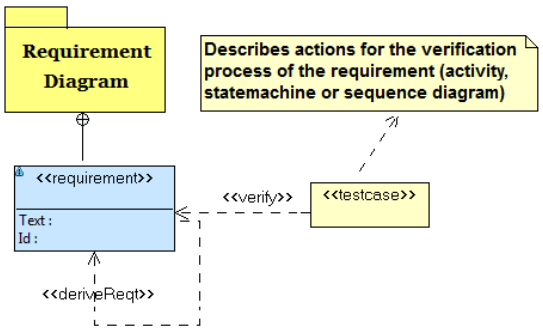


Fig. 1: <<Verify>> and <<deriveReqt>> relationships in requirement diagram

In this paper, we exploit the <<Verify>> relationship, to determine the activities which are used to verify requirements. We derive from functional requirements, more formal requirements described as properties about activity diagram

elements. For tractability purpose, the <<deriveReqt>> relationship will be exploited to relate between natural text and the more formal requirements.

3.2 SysML activity diagram

In this section, we introduce only a brief description of the SysML activity diagram and its elements, more details can be found in [1]. In SysML, an activity is a formalism for describing behaviour that specifies the transformation of inputs to outputs through a controlled sequence of actions. The basic constructs of an activity are actions and control nodes as illustrated in Figure 2. Actions are the building blocks of activities, each action can accept inputs and produces outputs, called tokens. These tokens can correspond to anything that flows such as information or physical item (e.g., water, signal). Control nodes include fork,

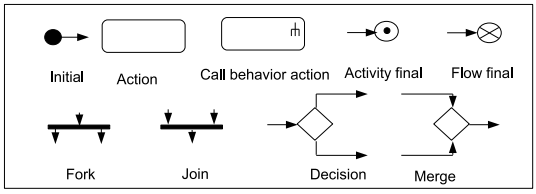


Fig. 2: Activity diagram basic constructs

join, decision, merge, initial, activity final, and flow final.
A specific type of action is the call behavior action. A call behavior action permits to invoke an activity when it starts, and passes the tokens from its input pins to the input parameter nodes of the invoked activity. A call behavior action terminates when its invoked activity reaches an activity final, or when the action receives a control disable. The tokens on the output parameter nodes of the activity are placed on the output pins of the action and a control token is placed on each of the control outputs of the action.

3.3 PNML for Modular Peri nets

The Petri Net Markup Language (PNML) [13] is an interchange format for all kinds of Petri nets. It is currently standardised by ISO/IEC JTC1/SC7 WG 19 as Part 2 of ISO/IEC 15909 [13]. The main features of PNML are its readability which is guaranteed by its XML syntax, its universality to support different Petri net type and its mutuality guaranteed by the use of common principals and common notations [8]. To address real world systems which are too large to be drawn on single page, the PNML provide a net type independent mechanism for structuring large Petri nets. Two mechanisms are proposed, pages and modules. The concept of pages is used with the concept of references to structure the nets

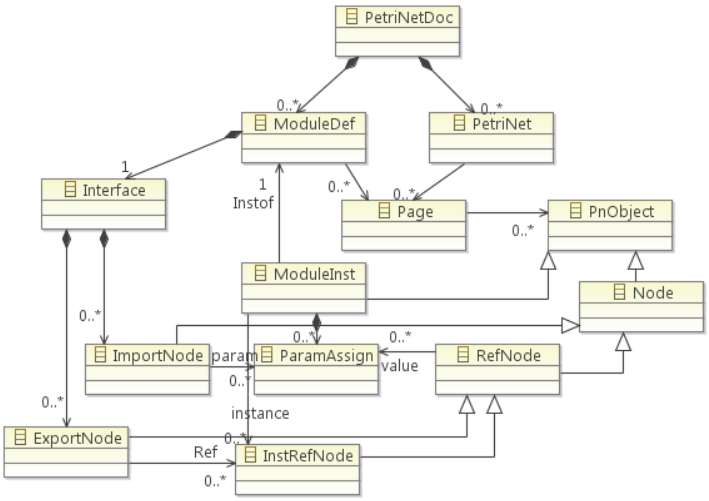


Fig. 3: Extract of the defined meta-model for the modular PNML

on several pages. It is used only for more convivial visual structure of the nets. The concept of modules is supported by modular PNML. Modular PNML as presented in [8] is an extension of the PNML to describe modular Petri nets. It is proposed for defining Petri net modules and for constructing nets from different instances of such modules. A Petri net module is defined as a Petri net with an interface composed by imported and exported nodes. For the transformation to perform in this work, we have extended the PNML (P/T Type) meta-model to support a modular structure of Petri nets. The Figure 3 presents an extract of Modular PNML meta-model which we have inspired from [8]. The presented Modular PNML meta-model extends the core PNML meta-model to allow the definition of modules (ModuleDef) and their instantiation (ModuleInst). Each module includes an interface which contains import and export nodes. A module instance assigns import nodes to reference nodes (ParmAssign) and can contains a reference nodes from other instances(InstRefNode). More explanations can be found in [8].

4 Overall methodology

In this section, we describe our methodology for verifying SysML requirements on activity diagrams. First, the SysML designer creates requirement and activity diagrams to specify the system. Then, he drives from functional requirements, which are related to the activity diagram by a <<Verify>> relationship, temporal requirements described as properties about activity diagram elements. After that, an automatic translation process is used to transform this SysML specification into a formal specification. The SysML activity diagram is translated into

modular Petri net and temporal requirements into formal properties described as temporal logic formulas. A Petri Net tool will be used to check if these requirements are verified in the derived Petri net modules. The verification will be guided by the existing <<Verify>> relationships between requirements and activities. Finally, a feed back is given to the SysML designer to correct his specification. As our approach is modular, in the case of the non satisfaction of a requirement, the generated feed back can give a more accurate indication about the sub activity and the actions which are related to the design error. The Figure 4 summarizes the steps of our methodology.

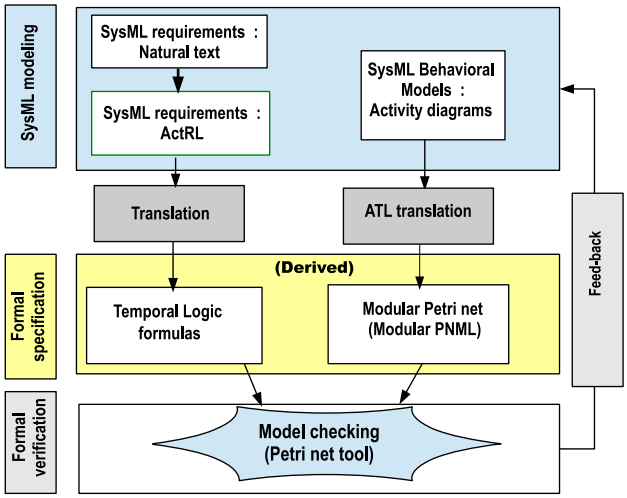


Fig. 4: Overall methodology

5 From activity diagrams to modular Petri nets

In this section, we describe our translation of SysML activity diagrams into modular PNML. We propose to use the activity diagram meta-model defined in the TOPCASED tool [14] as source meta-model and the modular PNML meta-model presented in Section 3.3 as target meta-model. Based on EMF(Eclipse Modeling Framework) with Ecore meta meta-model and ATL language [15], the transformations are defined as semantic and structural mappings based on the respective meta-models. The target model of the transformation is a modular Petri net described in modular PNML which preserve the structure and the semantic of the source SysML activity diagram model.

5.1 Mapping the structure

The transformation must preserve the composite structure of the SysML activity diagram in the target Petri net model. When the SysML activity diagram includes a call behavior actions to define composite activities, the derived Petri net will be composed of modules. Each sub-activity is translated into Petri net module.

The Figure 5 illustrates the derivation of the Petri net modular structure according to the activity decomposition.

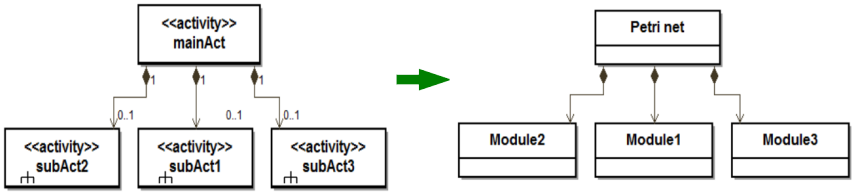


Fig. 5: The structure of the derived Petri net

We create a PNML document and Petri net only for the main activity diagram. For a sub-activity, we create a Petri net module. The ATL rule used to select the main activity is given below:

```
rule mainAct2mpnml {
from
d: ADUML!Activity(not (d.owner.oclIsTypeOf(ADUML!Activity)))
to
 p : MPNML!PetriNetDoc (nets <- f, modulesDef <-
 PNML!ModuleDef.allInstances()),
 s: MPNML!Name(text <- d.name),
 f: MPNML!PetriNet(name <- s)
 .....
}
```

5.2 Translating SysML activity constructs

The translation of basic activity constructs is inspired from the work presented in [5,6]. So, as we are interested to preserve the composite structure of the SysML activity diagram, we have adapted this translations mainly for input and output pins. The Figure 6 presents the used translation rules.

Translating call behavior actions Three principal steps are considered when translating call behavior actions :

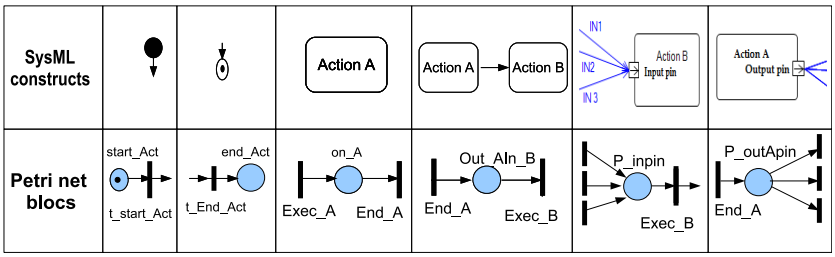


Fig. 6: Translation of basic SysML AD constructs

– Step 1: pass input flows from call behavior action to the called activity.
– Step 2: execute the called activity.
– Step 3: pass output flows from the called activity to the call behaviour action.

The mapping of a call behavior action A that invokes an activity Act with one input and one output control flow, n input pins and m output pins is as presented in Figure 7. The PNML code related to this translation includes definitions of

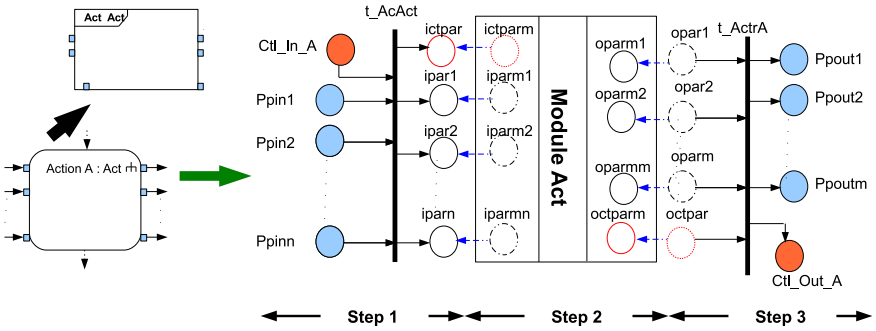


Fig. 7: Translation of call behavior action

transitions, places and arcs related to step 1 and step 3. It must also include an instance of the Petri net module defined for the activity Act (see the next section). This instance is defined like in the following listing :

```
<instance id="A_Act" ref=URI#Act>
<Paramassign parameter="ictparm" ref="ictpar"/>
<Paramassign parameter="iparm1" ref="ipar1"/>
......
<Paramassign parameter="iparmn" ref="iparn"/>
</instance>
```

We signal that the nodes $octpar, opar1, opar2, ...oparn$ (step 3) are instance reference places. They are defined in modular PNML like :

```
<InstRefPlace id="octpar" instance="A_Act" ref="octparm"/>
<InstRefPlace id="opar1"  instance="A_Act" ref="oparm1"/>
.....
<InstRefPlace id="oparm" instance="A_Act" ref="oparmm"/>
```

5.3 Mapping Sub-Activities

As described in Section 5.1, sub-activities are translated into PNML modules. Activity parameters are for accepting inputs to an activity and providing outputs from it. An activity with input and output parameters is translated into PNML module as illustrated in Figure 8. Input activity parameters are translated into

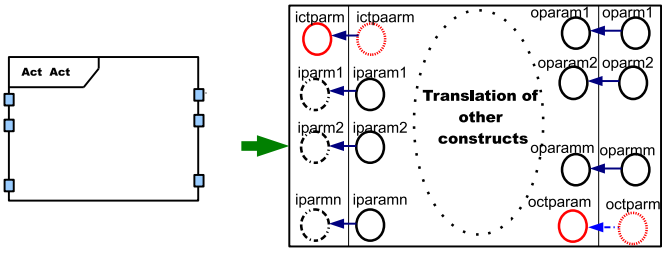


Fig. 8: Translation of Sub-activities

reference places. Output activity parameters are translated into places. The interface of the PNML module is composed of import places and export places. Import places are refereed by the reference places representing input activity parameters. Export places refer to places representing output activity parameters.

6 AcTRL: Activity Temporal Requirement language

As presented in Section 3.1, SysML requirements are described using an Id and natural text. To address this limitation, we propose AcTRL(Activity temporal requirement language) which can be used by SysML designers to express requirements to verify on activity diagrams. First, we define a high level representation of the activity diagram operational semantic as states/transitions system. Then, we define a set of predicate expressions which can be formulated about the states of activity diagram elements. To express temporal requirements related to the execution of an activity diagram, predicate expressions about activity elements are temporally quantified using the property specification pattern system proposed in [16].

6.1 Operational semantic of SysML activity diagram

During execution, at each instant, an activity has a specific state. This state is defined among others by : the activity status (not started, started, finished),

the states of all its actions, the states of its input and output parameters and the value of all the local variables used to express guards defined to control the tokens flows. The state of an action is defined by : its status (active, not active), the states of its incoming and outgoing control flows and the states of its input and output pins. According to this description, the operational semantic of an activity diagram can be represented by a high level states/transitions system as depicted in Figure 9.

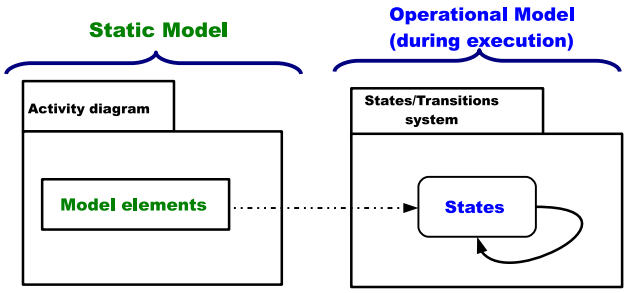


Fig. 9: Operational semantic of activity diagram

6.2 Predicate expressions about activity diagram elements

In this section, we present a sub-set of predicate expressions which can characterize the elements of an activity diagram during its execution. Let $ActivityName$ an activity, examples of such predicate expressions are :

1. If $actionName$ is action from activity $ActivityName$, then $[ActivityName].[actionName].isActive()$ is a valid predicate expression. Its value is True on a given state if $actionName$ is on execution.
2. If $actionName$ is action from activity $ActivityName$ and $ctlfName$ an incoming control flow of $actionName$, then $[actionName].incoming[ctlfName].isNotEmpty()$ is a valid predicate expression. The same expression can be defined for an output control flow.
3. If $actionName$ is action from activity $ActivityName$ and $PinName$ is its input pin, then $[actionName].input[PinName].isNotEmpty()$ is a valid predicate expression. The same expression can be defined for an output pin.
4. All the boolean OCL expressions about the objects manipulated in the activity $ActivityName$ are valid predicate expressions.
5. All the boolean expressions about the local variables used in $ActivityName$ are valid predicate expressions.
6. If $actExp$ is a valid predicate expression, then $not\ actExp$ is valid predicate expression.

7. If $actExp1$, $actExp2$ are valid expressions, then $actExp1$ *and* $actExp2$ and $actExp1$ *or* $actExp2$ are valid expressions.

6.3 Temporal expressions

The idea of property specification pattern system [16] is to allow users to construct complex properties from basic, assuredly correct building blocks by providing generic specification patterns (left side of Figure 10) encoding certain elementary properties: existence, absence, universality, bounded existence, precedence (chains), and response (chains), each specialized for a set of different scopes (right side of Figure 10) : globally, before R, after Q, between Q and R, after Q until R.

Given an activity diagram, requirements about its execution are interpreted as

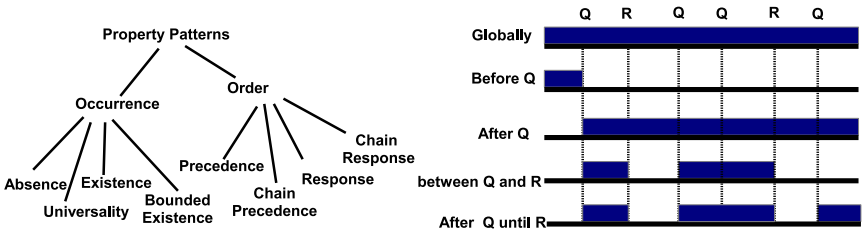


Fig. 10: Temporal pattern and scopes

properties over the states of its elements. These states can be characterized by the predicate expressions defined above. The SysML designer derives from functional requirements the elementary predicate expressions, then, he formalizes requirements by quantifying the predicate expressions by the necessary patterns and scopes to get temporal requirements about the activity execution. In the following, is given the grammar of the AcTRL (<Pred-exp> is predicate expression as defined above):

```
<AcTRL>::= <pattern> <scope>
<pattern>::= always <Pred-exp>
           | never  <Pred-exp>
           | eventually <Pred-exp>
           | <Pred-exp> precededing <Pred-exp>
           | <Pred-exp> following <Pred-exp>
 <scopes> ::= globally
           | before <Pred-exp>
           | after <Pred-exp>
           | between <Pred-exp> and <Pred-exp>
           | after <Pred-exp> until <Pred-exp>
```

6.4 Translation into CTL/LTL formulas

Functional requirements described using AcTLR can automatically translated into temporal logic formulas. A complete library is provided in [17], which propose translations into many formalisms (LTL, CTL, QREs, ...). So, we have to give a semantic for the defined predicate expressions according to the translation of activity diagram into Petri nets. As example, we consider the following predicate expression: $[ActivityName].[actionName].isActive()$, according to the translation of an action (Figure 6) will be translated into $(Marking(on\_A) =< 1 >)$, a proposition which means : the place on_A contains the mark $< 1 >$. As second example, the predicate expression $[actionName].output[PinName].isNotEmpty()$, according to the translation of output pins (Figure 6), will be translated into $(Marking(P\_OutApin)! =<>)$, which means : the place $P\_OutApin$ is marked.

7 Modular verification of requirements

Considering the compositional structure of activity diagrams can reduce the spatial and temporal complexity of their verification by model-checking. In this section, we propose an algorithm that guides the modular verification of SysML requirements on activities. We consider the verification of functional SysML requirements on a composite activity diagram where a set of call behavior actions is used to call set of activities. The proposed modular verification concerns the sub-activities and the composite activities according to their relations with SysML requirements. We assume that all functional requirements related to activities are expressed in AcTRL and depicted in a requirement diagram. By exploiting the $<< verify >>$ relationships, we generate a set of associations Set-Req-act containing all couples (ReqId, ActName), where ReqId is a requirement related to the activity ActName by a $<< verify >>$ relationship.
When performing the translation of an activity diagram into a modular Petri net, we generate a set of associations Set-ACT-PNM containing all couples (ActName, PNMname), where PNMname is a Petri net module translated from the activity ActName. When performing the translation of functional requirements expressed in AcTRL into temporal logic formulas, we generate a set of association Set-Req-LFor, which contains all couples (ReqId, lform), where Lform is a temporal logic formula derived from the requirement ReqId.
The verification of requirements on activities is achieving according to the algorithm 1. Changing the design of a sub-activity may influence the verification of a requirement related to its main activity. For this reason, the requirements related to a composite activity are verified after the validation of all the requirements related to their sub-activities. The algorithm begins by verifying the requirements related to sub-activities, then it process the verification of requirements related to composite activities until it reaches the main activity. The algorithm has complexity depending on the complexity of the model-checking (the function check()). It process N check; where N is the number of requirements.

Algorithm 1 Verify(ActName)

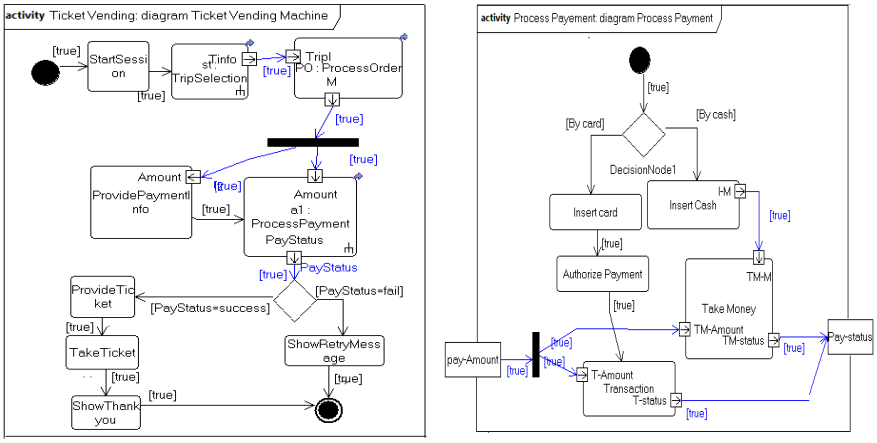
```
if (ActName does not contain call behavior actions) then
  Foreach (couple (ReqId, ActName)∈ Set-Req-act
  if ( (ActName, PNMname)∈ Set-ACT-PNM and (ReqId, lform) ∈ Set-Req-LFor)
  then
    check (lform, PNMname);
  end if
  EndForeach
else
  Foreach (sActName ∈ Sub-activities (ActName))
  Verify(sActName);
  EndForeach
  Foreach (couple (ReqId, ActName)∈ Set-Req-act
  if ( ActName, PNMname)∈ Set-ACT-PNM and (ReqId, lform) ∈ Set-Req-LFor)
  then
    check (lform, PNMname);
  end if
  EndForeach
end if
```

8 Application : A Ticket Vending Machine case study

In this section, we consider a Ticket Vending Machine(TVM) case study to illustrate our methodology. A TVM can be used to dispense tickets to passengers at a railway station. The behavior of the machine is triggered by passengers who need to buy a ticket. When passenger starts a session, TVM will request trip information from commuter. Passengers use the front panel to specify their boarding and destination place, details of passengers (number of adults and children) and date of travel. Based on the provided trip info, TVM will calculate payment due and display the fare for the requested ticket. Then, it requests payment options. Those options include payment by cash, or by credit or debit card. After that, the passenger chooses a payment option and processes to payment. After a success payment, the TVM prints and provides a ticket to passenger.
We specify the function of TVM by the activity diagram shown in Figure 11a. The activity diagram describes a composite activity which calls another activities. As example, we present the "process payment" sub-activity in the Figure 11b.

We specify the requirements to verify by activities using requirement diagrams. As example, two requirements are presented in Figure 12. They are expressed in AcTRL and related by a $<< verify >>$ relationship to activities. In this diagram extract, Set-Req-act = {(DREQ1, TicketVending), (DREQ2, Process Payment)}. From AcTRL expressions, we derive two logic formulas F1 and F2. As consequence, Set-Req-LFor = {(DREQ1, F1), (DREQ2, F2)}.

The activity diagram is translated into Petri net modules described in PNML. The running of the implemented ATL rules produces a XMI serialisation of the modular PNML document. The Figure 13 shows the structure of the derived



(a) A main activity diagram for TVM (b) A sub-activity representing the payment process

Fig. 11: Activities for TVM

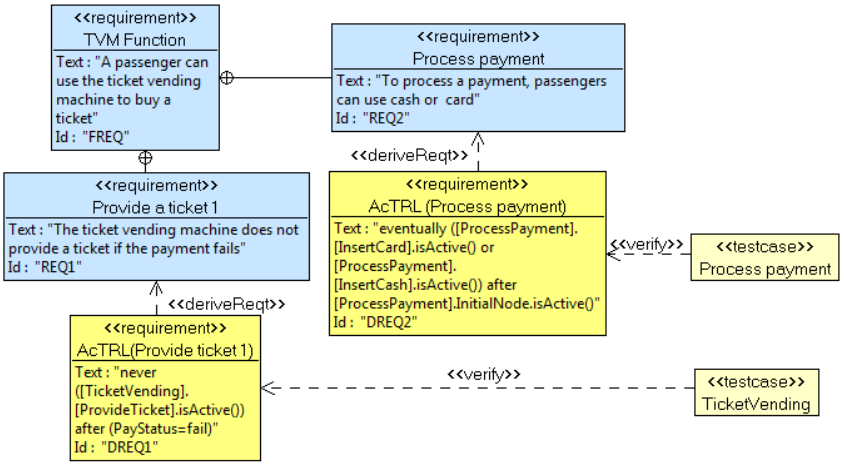


Fig. 12: A requirement diagram for TVM that uses AcTRL

PNML document. As the SysML activity diagram contains three sub-activities, the derived PNML document will be composed of a main Petri net and a Petri net module for each sub-activity. The set Set-ACT-PNM contains (TicketVending, Petri net TicketVending) and (Process Payment, Module ProcessPayment). By applying the algorithm 1, F2 will be checked in "Module ProcessPayment" then F2 will be checked in "Petri net TicketVending".

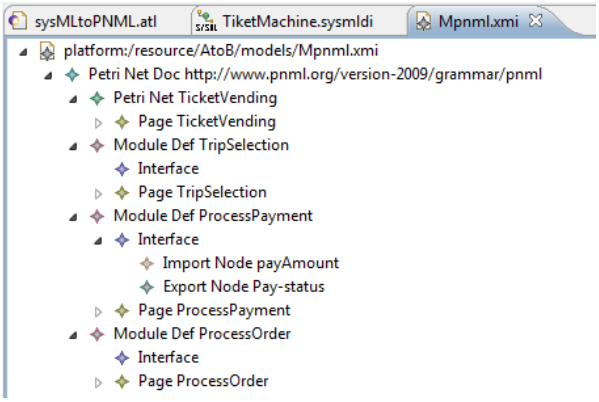


Fig. 13: Structure of the derived modular PNML

9 Conclusion and Future work

In this paper, we presented a methodology that proposes a modular verification of SysML specifications. The proposed methodology considers both requirements and activity diagrams. It consists on translating a composite activity diagram into modular Petri net. Then, it proposes a formalization of requirements related to activities by the proposition of AcTRL, which can be used by SysML designers and their expressions are translatable into temporal logics. Finally, an algorithm is proposed to guide the modular verification of SysML requirements. The translation from SysML activities to modular Petri net was fully automated using model to model transformation with ATL language. To illustrate the effectiveness of the proposed methodology, a practical case study was given.
As future work, we plan to automatize the translation of AcTRL expressions according to the translation of the activity diagram. Also, we plan to implement our methodology into complete framework. By completing these tasks, a SysML specification with requirements and activity diagram can be automatically verified using a Petri net tool. The next step will be the feedback of analysis results and their interpretation on SysML models.

References

1. OMG: OMG Systems Modeling Language (OMG SysML^{TM}) Version 1.2. (2010) downloadable from http://www.omg.org.
2. Nejati, S., Sabetzadeh, M., Falessi, D., Briand, L., Coq, T.: A SysML-based approach to traceability management and design slicing in support of safety certification: Framework, tool support, and case studies. Information and Software Technology **54** (2012) 569 – 590
3. Debbabi, M., Hassaine, F., Jarraya, Y., Soeanu, A., Alawneh, L.: Verification and Validation in Systems Engineering: Assessing UML/SysML Design Models. 1st edn. Springer-Verlag New York, Inc., New York, NY, USA (2010)

4. Knorreck, D., Apvrille, L., de Saqui-Sannes, P.: TEPE: a SysML language for time-constrained property modeling and formal verification. ACM SIGSOFT Software Engineering Notes **36** (2011) 1–8
5. Foures, D., Vincent, A., Pascal, J.: ACTIVITYDIAGRAM2PETRINET : Transformation-Based Model In Accordance With The Omg SySML Specifications. In: Proceedings of the Eurosis, The 2011 European Simulation and Modelling Conference. (2011) 429–434
6. Rahim, M., Hammad, A., Ioulalen, M.: Modular and Distributed Verification of SysML Activity Diagrams. In: MODELSWARD 2013, 1st Int. Conf. on Model-Driven Engineering and Software Development, Barcelona, Spain. (2013) 202–205
7. Klein, F., Giese, H.: Joint structural and temporal property specification using timed story scenario diagrams. In: Fundamental Approaches to Software Engineering. Springer (2007) 185–199
8. Michael, W., Ekkart, K.: The Petri net markup language. In: Petri Net Technology for Communication-Based Systems. Springer (2003) 124–144
9. Linhares, Marcos Vinicius and de Oliveira, Rômulo Silva and Farines, J-M and Vernadat, François: Introducing the modeling and verification process in SysML. In: Emerging Technologies and Factory Automation (ETFA) IEEE Conference, IEEE (2007) 344–351
10. Eshuis, R.: Symbolic model checking of UML activity diagrams. ACM Transactions on Software Engineering and Methodology (TOSEM) **15** (2006) 1–38
11. Andrade, E., Macie, P., Callou, G., Nogueira, B.: A Methodology for Mapping SysML Activity Diagram to Time Petri Net for Requirement Validation of Embedded Real-Time Systems with Energy Constraints. In: Third International Conference on Digital Society, ICDS'09. (2009) 266–271
12. Foures, D., Albert, V., Pascal, J.C., Nketsa, A.: Automation of SysML activity diagram simulation with model-driven engineering approach. In: Proceedings of the 2012 Symposium on Theory of Modeling and Simulation - DEVS Integrative M&S Symposium. TMS/DEVS '12, San Diego, CA, USA, Society for Computer Simulation International (2012) 11:1–11:6
13. PNML.org: (Reference site for the implementation of Petri Net Markup Language (PNML)) url : http://www.pnml.org.
14. Farail, P., Goutillet, P., Canals, A., Le Camus, C., Sciamma, D., Michel, P., Crégut, X., Pantel, M.: The TOPCASED project: a toolkit in open source for critical aeronautic systems design. Ingenieurs de l'Automobile (2006) 54–59
15. Allilaire, F., Bézivin, J., Jouault, F., Kurtev, I.: ATL-eclipse support for model transformation. In: Proceedings of the Eclipse Technology eXchange workshop (eTX) at the ECOOP 2006 Conference, Nantes, France. Volume 66., Citeseer (2006)
16. Dwyer, M.B., Avrunin, G.S., Corbett, J.C.: Patterns in property specifications for finite-state verification. In: Proceedings of the International Conference on Software Engineering, IEEE (1999) 411–420
17. Alavi, H., Avrunin, G., Corbett, J., Dillon, L., Dwyer, M., Pasareanu, C.: (Specification Patterns) url: http://patterns.projects.cis.ksu.edu.