Hierarchical Colored Petri Nets for the Verification of SysML Designs- Activity-based Slicing Approach

Messaoud Rahim[1,2], Malika Boukala-Ioualalen[2], and Ahmed Hammad[3]

[1] Mathematics and computer sciences Department, MEDEA University, ALGERIA.
messaoud.rahim@univ-medea.dz
[2] MOVEP Laboratory, USTHB, ALGERIA.
mioualalen@usthb.dz
[3] FEMTO-ST Institute, UMR CNRS 6174, Besançon, France.
ahmed.hammad@femto-st.fr

Abstract. Requirements verification at early phases of the design process is one of the main challenges when developing critical and complex systems. In this paper, we focus on the verification of SysML functional requirements on activity diagrams. Our contribution consists in the proposition of a slicing based verification approach guided by the SysML relationships between requirements, blocks, and activities. The objective is to provide a verification methodology for complex systems with many components. Our approach is based on formalizing activity diagrams using hierarchical coloured Petri nets(HCPNs). The proposed slicing permits to alleviate the verification process. For verifying a given requirement, the slicing consists in extracting a minimized fragment (slice) of the HCPNs, which is sound and sufficient to realize the verification. The approach is illustrated by a case study, where we specify and we verify a fire protection system for data-centers.

Keywords: SysML · Activity Diagram · Requirements · Hierarchical Colored Petri Nets· Model Slicing · Formal Verification

1 Introduction

SysML (System Modeling Language) [**?**] is an OMG standard modelling language supported by leading organizations from the systems engineering industry, including the International Council On Systems Engineering(INCOSE). SysML covers four perspectives on system modeling : structure, behavior, requirement, and parametric diagrams. The requirement diagram is one of the most important models provided in SysML. Thanks to this diagram, SysML offers an explicit representation of the different kinds of relationships between requirements and other design artefacts [**?**]. These relationships are of great importance. They can be exploited to guide several aspects related to system design, particularly requirements verification.

The activity diagram allows to specify the dynamic aspects of the system. For describing the full behaviour of complex systems with many components, SysML activity diagram can be composed of several sub-activities. Furthermore, swimlanes can be used to allocate actions to system components (SysML blocks). Despite the various advantages of SysML, it remains a semi-formal language without possibilities of formally verifying models described by it. For this reason, using SysML in conjunction with formal methods is suggested to provide a formal validation for SysML specifications.
Model checking techniques [**?**] are based on automatic examination of all reachable states to check whether a model of a given system satisfies the desired properties. However, as known, the major flaw of model-checking resides in the state explosion problem. Slicing the model according to the property to verify is one of the approaches aiming to alleviate this problem [**?**].
This paper concentrates on the verification of SysML functional requirements linked to behavioral models. Other requirements classes such as performance and availability are not considered. It proposes a methodology based on model-checking to verify requirements on activity diagrams.
In contrast to other verification approaches which are based on translating activity diagrams into Petri nets [**?**,**?**], our approach consider the translation of activity diagrams with swimlanes and sub-activities. This allows to represent structural and functional decomposition of complex systems as they are captured by the activity diagrams. In addition, the verification in our approach is guided by the SysML relationships between requirements, blocks, and activities, which enables the use of a slicing based verification. For verifying a given requirement, the slicing consists in extracting a minimized fragment (slice) of the HCPN model translated from the activity diagram which is sound and sufficient to realize the verification.
The remainder of this paper is organized as follows: Section **??** discusses the related work. Section **??** presents a background about the related concepts. Section **??** introduces our verification methodology. In Section **??**, we illustrate our approach by a case study. Finally, in Section **??**, we conclude and we outline directions for future works.

2 Related Work

The idea of translating SysML specifications to get formal semantic models is interesting for verification purposes [**?**,**?**]. For activity diagrams, several approaches based on their translation into Petri net models were proposed, each one was based on using specific classes of Petri net depending on the proposed goals to achieve. The work in [**?**] introduces an approach using SysML for structural specification and Petri net for expressing the system behavior. In [**?**], a technique was proposed to map SysML activities into Time Petri net for validating the non functional requirements of real-time systems with energy constraints. A bi-directional transformation of UML 2 activities into Petri nets was proposed in [**?**]. In this translation, only the basic activity constructs was considered. To

consider the semantics of data flows, translating activity diagram into coloured Petri nets was proposed in [**?**]. Activity calling and swimlanes were not considered in these works. Our work considers the semantics of activities with swimlanes, call behavior actions and data flows. These previous approaches intended to verify activity diagrams by model-checking techniques. However, verifying models for complex systems is very challenging task; namely regarding the state space explosion problem. One way to alleviate this problem is model slicing [**?**]. Several works have proposed model slicing approaches, but very few literatures are reported on slicing of activity diagrams based on requirements and almost all of them address test case generation. As examples, Ray et al. [**?**] have used conditioned slicing for test case design through activity diagram. The work in [**?**], proposes a static slicing technique to enhance the comprehension of Use Case Map models. In contrast to these works, we propose model slicing on the formal semantic model translated from the activity diagrams for verification purpose. So, the verification is guided by the SysML specification and the slicing criteria are taken from the SysML requirement diagram.

3 Background

3.1 SysML requirement and activity diagrams

Requirement diagrams are used for specifying requirements and the exiting relationships between them and the other artifact. Our work exploits the "satisfy" and the "Verify" relationships. The "Satisfy" relationship is a dependency between a requirement and a SysML block. It means that the requirement is satisfied by the block. The "Verify" relationship is a dependency between a requirement and a test case. It means that the test case is used to verify the requirement. A test case can be a sequence, a state machine or an activity diagram. Our work considers test cases described by activity diagrams.

The activity diagram is a formalism for describing the system behaviors. The basic constructs of an activity are actions and control nodes. A specific type of action is the call behavior action. A call behavior action permits to invoke an activity when it starts. Activities can be with Swimlanes or activity partitions. Swimlanes are used to allocate actions to SysML blocks.Formally, an activity with swimlanes and call behavior actions is tuple $Act = \langle N, E, Part, Alloc, CAct, call \rangle$ where:

- N is a finite set of nodes partitioned into subsets ($N = N_a \cup N_c \cup N_o$), where N_a is the set of action nodes, N_c is the set of control nodes and N_o is the set of object nodes which represents input and output pins and input and output activity parameters. N_a is partitioned into a set of simple action nodes N_{sa} and a set of call behavior action nodes N_{ca} ($\forall a \in N_{ca}$ and a call $act_i : act_i \in CAct$). N_c is further partitioned into subsets : N_d of decision nodes, N_m of merge nodes, N_f of fork nodes, N_j of join nodes, N_i of initial nodes, N_{ff} of flow final nodes and N_{af} of activity final nodes.

- E is a finite set of edges each connecting either two object nodes with each other or an object node with an action or a control node, i.e., $E \subset (N \times N)$.
- $Part = \{B_1, ...B_n\}$ is a finite set of activity partitions.
- $Alloc : N \rightarrow Part$ is a function that allocates activity nodes to activity partitions.
- $CAct$ is the set of the activity invoked by the call behavior actions.
- $call : N_{ca} \rightarrow CAct$ is a function that associate a called activity to each call behavior action.

3.2 Hierarchical Coloured Petri Nets

Hierarchical Colored Petri Nets (HCPNs) introduce a facility for building a large CPN model by using a number of small CPNs. In HCPNs, the notion of substitution transition is provided to allow the user to relate a transition to a more complex CPN, called subnet. A substitution transition has input and output places called socket places. More formally, a HCPN is a tuple HCPN = $\langle$ S, P, T, A, Σ, St, SA, P_{por}, P_{soc}, PS $\rangle$ where,

- S is the set of pages (subnets), P is the set of places and T is the set of transitions.
- A $\subseteq$ (P $\times$ (T $\cup$ St)) $\cup$ ((T $\cup$ St) $\times$ P): is the set of arcs.
- Σ: is the set of non-empty types, called colour sets.
- St: is the set of substitution transitions, SA: St $\rightarrow$ S is a page assignment function, $P_{por} \subseteq$ P is the set of port places, $P_{soc} \subseteq$ P is the set of socket places, and PS : $P_{por} \rightarrow P_{soc}$ is a port-socket relation function that assigns a port places to a socket places to relate the subnet to the main net.

HCPNs are chosen not only due to their expressive power to capture the semantics of activity diagram constructs including data flows, but also because they allow to represent structural and functional decomposition of complex systems.

4 Methodology

Our methodology for verifying SysML system design aims to verify SysML functional requirements. The requirements of the system are described in a SysML requirement diagram and the system behavior is described by an activity diagram. This latter will be structured with activity partititions. Each one represents a SysML block. Also, it can contain call behavior actions that call other activities for modelling complex behaviors. We verify if the SysML functional requirements are verified by the activity diagram.
For that, we define, as detailed in the figure Fig.**??**, a slicing based verification approach to overcome the state space explosion problem. The first step of the proposed verification consists in translating the SysML activity diagram into a HCPN. Then, we extract the blocks and the activities which satisfy (respectively. verify) requirements. This step exploits "satisfy" and "verify" relationships. After that, requirements will be translated into temporal logic formulas

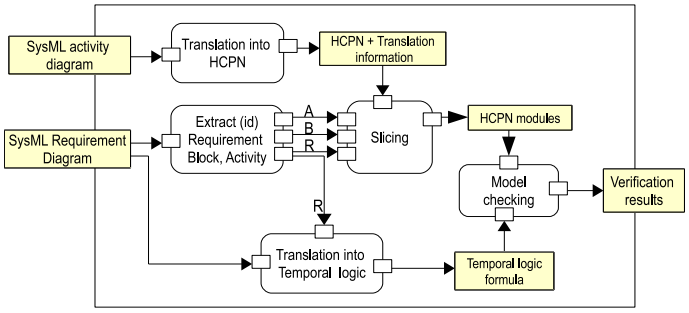

Fig. 1. The verification methodology

and information about blocks and activities will be used to slice the HCPN. To avoid to perform many translations, the slicing will be applied in the HCPN. Finally, we check the temporal logic formulas in the sliced HCPN and we return the verification results.

4.1 Translation of activity diagrams

The translations of basic activity constructs are done as in our previous works [**?**,**?**]. So, in this section, we only present the translation of call behavior actions and activity partitions (swimlanes).
The main idea of our translation is as follow : let $AD = \langle setAct \rangle$ be an activity diagram consisting on a set of activities (main activity and sub-activities), where each activity $AName = \langle N, E, Part, Alloc, CAct, call \rangle$ can be partitioned into set of activity partitions $Part$ and can contain call behavior actions. The translation of AD into HCPN = $\langle$ S, P, T, A, Σ, St, SA, P_{por}, P_{soc}, PS $\rangle$ consists in translating each activity $Act \in setAct$ into a HCPN page $Act\_page \in S$ and each call behavior action $ActionA$ that call the activity Act into a substitution transition $SubTAction\_A \in st$, where $SA(SubTAction\_A) = Act\_page$. If $AName$ is partitioned, each activity partition $B \in Part$ is translated into a substitution transition $TB \in St$ and a HCPN page $B\_page \in S$, where $SA(TB) = B\_page$.
The translation of activity diagrams into HCPNs is fully automated. It is performed based on model-driven approach using the ATL transformation language.

4.2 The slicing step

Let R be a requirement to verify in an activity Act partitioned in activity partitions $Part = \{B_1, ...B_n\}$ and R is satisfied by a block $B_i \in Part$. The idea of the proposed slicing consists in determining, for the requirement R, the activity partitions which are needed to its verification. Because R has "satisfy" relationship with B_i, B_i is one of the needed activity partitions. However, in some cases, the actions allocated to B_i are not sufficient to verify R. This is due to their connections with other actions allocated to other activity partitions. Namely, when the

actions allocated to B_i require, in their inputs, outputs from actions allocated to other activity partitions. For that, we compute the set of activity partitions from which B_i has a direct or indirect dependency. Let $dep(B_i, Act)$ be the function used to perform this computation. Each activity partition $B_j \notin dep(B_i, Act)$ is not needed for the verification of R. The slicing is done on the HCPN model by deleting, for each $B_j \notin dep(B_i, Act)$, its corresponding substitution transition TB_j.
For computing the direct and indirect dependencies between the activity partitions, we propose to construct a Dependency Graph Between Activity Partitions $DGBAP = \langle Part, A \rangle$, where, $DGBAP$ is a directed graph which has the elements of $Part$, as vertices and the elements of $A \subset Part \times Part$ as directed edges($(B_i, B_j) \in A$ if the activity partition B_j has at least one input link from the activity partition B_i). So, computing $dep(B_i, Act)$ will consists in performing the function $compute\_dep(B_i, DGBAP)$ as listed in the Algorithm **??**. Given

Algorithm 1 $compute\_dep(B_i, DGBAP)$
Inputs : (B_i: activity partition, $DGBAP = \langle Part, A \rangle$)
Output: (DL: a list of activity partitions)

```
1: DL ← {};Stack P; Push(P, B_i);
2: while (Not empty(P)) do
3:    B_j ← Pop(P); DL ← DL ∪ {Bj};
4:    for all (B_k, B_j) ∈ A) do
5:       if (B_k ∉ DL) then
6:          Push(P, B_k);
7:       end if
8:    end for
9: end while
```

$DGBAP$ and an activity partition B_i, the algorithm allows to compute a list of activity partitions from which B_i is dependent.

4.3 Verification of requirements

In order to allow the verification of SysML requirements, we propose to express them as temporal properties about the activity diagram elements using the ActTRL language [**?**,**?**] and then we translate them into ASK-CTL formulas as in [**?**]. Finally, the CPNTools is used as model-checking tool for analysing the generated ASK-CTL formulas on the HCPN slices.

5 Case study : fire protection System in data-center

We consider a fire protection system in data-center as a case study. A fire protection system includes an automatic fire detection system, fire alarms, and a fire suppression system. Automatic fire detection system is designed to detect fire

in its earliest stages. It is based on heat detectors which detect fire and notify information to detection central. To confirm a fire, a cross-zoned system can be adopted. It permits to reduce unwanted and nuisance alarms caused by detectors. In cross-zoned system, a fire is confirmed only if two detectors notify a fire to the detection central which activates the fire suppression system, the internal and the external alarms. A detection by a single detector will only be notified to the system administrator. In data-centers, fire suppression system must be based on gas extinguishing. When a fire is confirmed, the detection central activates an auto actuation device to suppress the fire automatically.

5.1 SysML specification

The aim of this step is to produce a SysML specification. The SysML Block Definition Diagram in figure Fig.**??** describes the structure of the system. The

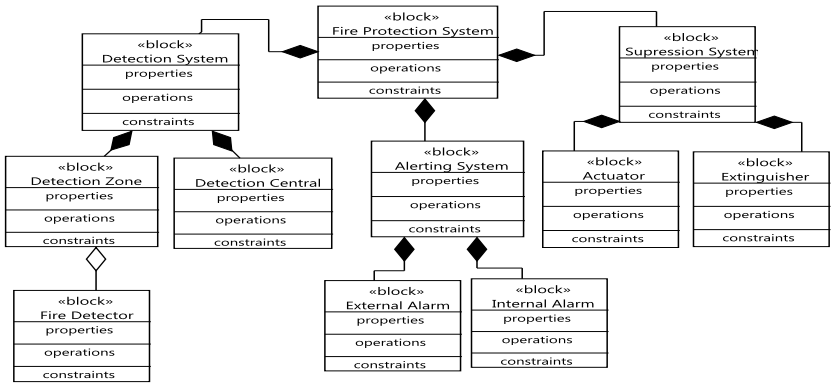

Fig. 2. Block Definition Diagram for the fire protection system

requirements of the fire protection system are specified in a SysML requirement diagram. So, in this paper, we consider only a sub-set of requirements. As shown in figure Fig.**??**, the main requirement of the system is to protect the data-center from fires. This requirement is composed of three requirements (R0, R3, R4). The requirement R0 itself is composed of two requirements (R1 and R2). The requirement R1 deals with the detection of fires. It is satisfied by the "Fire Detector" block and verified by the "Detect Fire" activity. The requirement R2 deals with the confirmation of fire detection. It is satisfied by the "Detection System" Block. The requirement R3 is to ensure the sounding of alarms when the detection of a fire is confirmed. It is satisfied by the "Alerting System" block. The requirement R4 deals with the suppression of fires. It is satisfied by the "Suppression System" block. The requirement R2, R3 and R4 are to verify by the "Protect from fire" activity (see Fig.**??**).

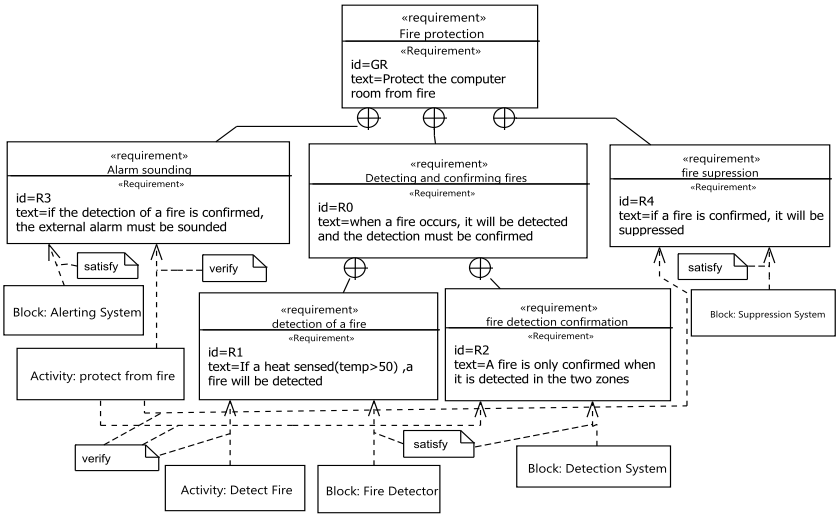

Fig. 3. A requirement diagram for the fire protection system

We specify the indented behavior of the fire protection system by the activity diagram presented in the figure Fig.**??**. We create three activity partitions, the

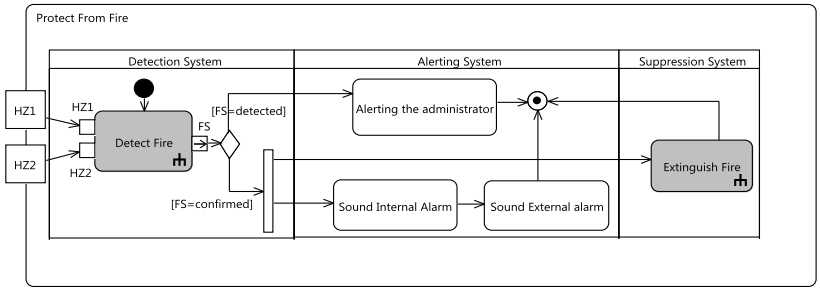

Fig. 4. Activity diagram for the fire protection system

first represents the detection system block, the second represents the alerting system block and the third represents the suppression system block. If a fire occurs, it will be detected by the fire detection system. For this, the "Detection System" activity partition contains the "Detect Fire" call behavior action which allows to sense the heat by the fire detectors(in Zone 1 and in Zone 2) and then, determines if a fire is occurred. The activity diagram representing the "Detect Fire" sub-activity is given in Fig.**??**. For the actions assigned to the "Alerting System" activity partition, two cases are modelled : fire detected but

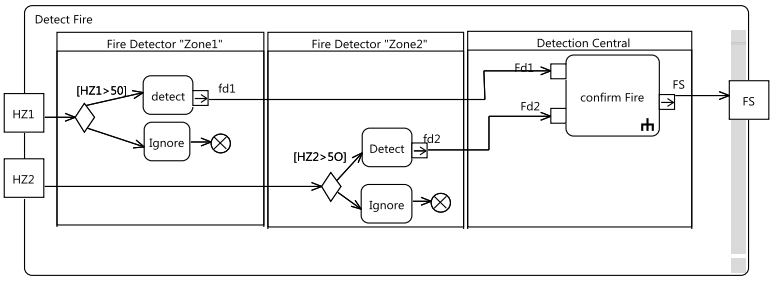

Fig. 5. Activity diagram for the "Detect Fire" sub-activity

not confirmed and fire confirmed. In the first case, we only produce an alert for the system administrator. For the second case, we sound the internal and the external alarms. Also, the call behavior action "extinguish fire" assigned to the "suppression System" activity partition is activated in the second case. The models related to the sub-activity called by the "extinguish fire" call behavior action are not presented in this paper.

5.2 Translation into HCPN

We show in the figure Fig.**??** the prime page of the HCPN model derived from the activity "Protect from fire". As explained in Sect.**??**, the HCPN model in its prime page contains three substitution transitions.

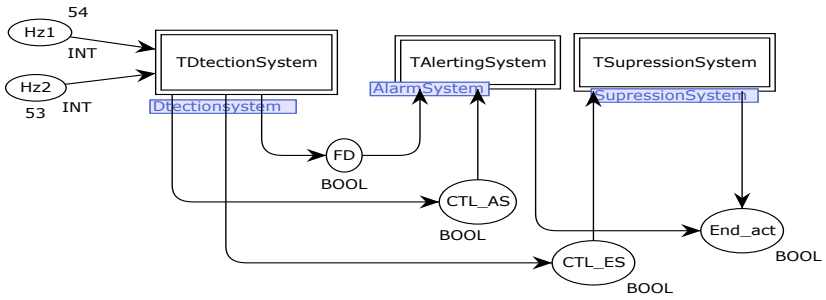

Fig. 6. The HCPN model derived from the "Protect from fire" activity

As example, in the figure Fig.**??**, we show the HCPN sub-page assigned to the "TDetectionSystem" substitution transition. It contains the translation of the fork node, the decision node and the "Detect Fire" call behavior action allocated to "Detection System" activity partition (see Fig.**??**). The "Detect Fire" call

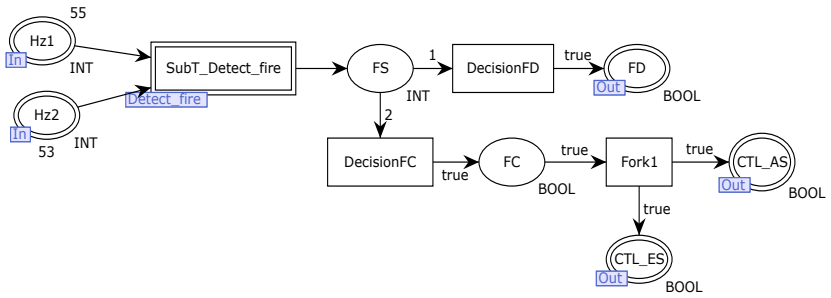

Fig. 7. The HCPN "Detection system" sub-page

behavior action is translated into a substitution transition ("Detect_fire"). The HCPN sub-page assigned to this latter is shown in figure Fig.**??**, it is obtained from the translation of the activity "Detect Fire" (see Fig.**??**) called by the "Detect Fire" call behavior action (see Fig.**??**). In the "Detect Fire" activity, we

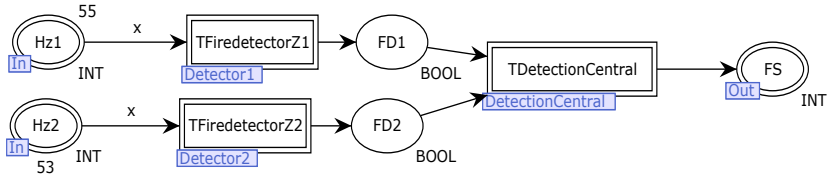

Fig. 8. The HCPN model derived from the "Detect fire" activity

have three activity partitions. They are translated into substitution transitions.

5.3 Slicing examples and verification of requirements

After the translation step, we can begin the verification of requirements using our slicing technique. This section presents and explains some slicing examples when verifying the requirements of the fire protection system. From the requirement diagram, the requirements R2, R3 and R4 are to verify by the activity "Protect from Fire". First, we construct the DGBAP related to this activity (see Sect.**??** and Fig.**??**).

As the requirement R2 is satisfied by the block "Detection System", according to the constructed DGBAP, its verification is independent from the other activity partitions. So, the slicing consists in conserving only the substitution transition related to the "Detection System" activity partition.
The requirement R3 is satisfied by the block "Alerting System", according to the constructed DGBAP, its verification is dependent from the "Detection System"

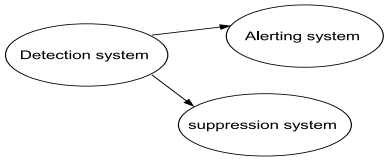

Fig. 9. The DGBAP for the activity "Protect from fire"

and "Alerting System" activity partitions. The slicing consists in deleting the substitution transition related to the "Suppression System" activity partition.
The requirement R4 is satisfied by the block "Suppression System", according to the constructed DGBAP, its verification is dependent from the "Detection System" and "Suppression System" activity partitions. The slicing consists in deleting the substitution transition related the "Alerting System" activity partition.

6 Conclusion

In this paper, we presented a slicing based approach for the verification of SysML requirements on activity diagrams. The verification is based on translating activity diagrams into HCPNs. The proposed slicing approach is guided by the relationships between requirements, activities and blocks. The slicing can be applied, for a given requirement, to verify on an activity diagram with activity partitions. The slicing is done for such requirements on the HCPN translated from the activity diagram. The effectiveness of the proposed technique depends on the activity partitions dependencies and differs from a requirement to another. The approach was illustrated by a case study, where a SysML activity diagram with activity partitions and call behavior actions was translated into HCPN. The HCPN was then sliced to verify SysML requirements.
As future work, it is important to consider the improvement of the proposed slicing technique which can be achieved through the application of our approach on more complicated case studies.