Efficient Probabilistic Abstraction for SysML Activity Diagrams

Samir Ouchani[*], Otmane Ait Mohamed, and Mourad Debbabi

Computer Security Laboratory, Hardware Verification Group,
Concordia University, Montreal, Canada
{s_oucha,ait,debbabi}@ece.concordia.ca

Abstract. SysML activity diagrams are OMG/INCOSE standard models for specifying and analyzing systems' behaviors. In this paper, we propose an abstraction approach for this type of diagrams that helps to mitigate the state-explosion problem in probabilistic model checking. To this end, we present two algorithms to reduce the size of a given SysML activity diagram. The first eliminates the irrelevant behaviors regarding the property under check, while the second merges control nodes into equivalent ones. The resulting abstracted model can answer safely the Probabilistic Computation Tree Logic (PCTL) property. Moreover, we present a novel calculus for activity diagrams (NuAC) that captures their underlying semantics. In addition, we prove the soundness of our approach by defining a probabilistic weak simulation relation between the semantics of the abstract and the concrete models. This relation is shown to preserve the satisfaction of the PCTL properties. Finally, we demonstrate the effectiveness of our approach on an online shopping system case study.

Keywords: Abstraction, SysML Activity Diagram, Probabilistic Automata, PCTL.

1 Introduction

Various techniques have been proposed for the verification of software and systems including model checking, type checking, equivalence checking, theorem proving, and dynamic analysis. Particularly, the most popular one used for the assessment of UML and SysML behavioral diagrams is model checking [1,2]. The latter is a formal automatic verification technique for finite state concurrent systems that checks temporal logic specifications on a given model. In addition to qualitative model checking, quantitative verification techniques based on probabilistic model checkers [3] have recently gained popularity. Probabilistic verification offers the capability of interpreting probabilistically the satisfiability of a given property on systems that inherently exhibit probabilistic behavior. Despite its wide use, model checking is generally a resource-intensive process that requires large amount of memory and time processing. This is due to the fact that the systems' state space may grow exponentially with the number of variables combined with the presence of concurrent behaviors and clocks. Moreover, several

[*] Corresponding author.

available model checkers cannot support all systems' features such as buffers, channels, and/or real variables. To overcome these issues, various techniques have been explored [4] for qualitative model checking and then leveraged to the probabilistic case. Among these techniques, several solutions aim at optimizing the employed model checking algorithms by introducing symbolic data structures based on binary decision diagrams, while others target the reduction of the input model. As we are interested in reusing existing model checkers, we concentrate on the second category that includes abstraction approaches.

Abstraction is one of the most relevant technique for addressing the state explosion problem [3,5]. It can be defined as a mapping from a concrete model into a more abstract one that encapsulates the systems' behavior while being of a reduced size. The intuition behind this transformation is to be able to check a property against an abstract model and then to infer safely the same result on the concrete model. Abstraction techniques can be classified in four categories [6]: 1) *Abstraction by state merging* aims at merging states of systems that have similar features. 2) *Abstraction on variables* targets the data in the model and aims at representing a set of values as one symbolic variable. 3) *Abstraction by restriction* operates by forbidding some behavior of the system. 4) *Abstraction by observer automata* restricts system's behaviors to those acceptable by an automaton that observes the system from outside. Our proposed framework takes advantage of the first and the third category.

In this paper, we are interested in the efficient verification of systems' design models expressed as SysML activity diagrams [7]. These diagrams are behavioral and allow for probabilistic behavior specification. Our approach combines two mechanisms of abstraction, the first is based on ignoring the irrelevant action nodes with respect to a given property and the second applies reduction rules that collapse control nodes in the SysML activity diagram. Our approach is depicted in Figure 1. In order to prove the soundness of our algorithm and the preservation of PCTL [3,8] properties satisfaction, we present a new calculus, namely NuAC, that captures the semantics foundation of SysML activity diagrams and we express its operational semantics as probabilistic automata. Furthermore, we demonstrate the practical application of our technique using a case study that would be otherwise difficult to verify. Thus, we use the probabilistic model checker PRISM [9] and we rely on our translation algorithm [2] that maps SysML activity diagrams into PRISM model. Besides, we show significant reduction in the state space and verification time, which makes probabilistic model checking helpful.

The remainder of this paper is organized as follows. Section 2 presents the related work. Section 3 explains the formal representation of SysML activity diagrams. The proposed abstraction approach is detailed in Section 4. Section 5 defines and proves the soundness of our algorithm and the preservation of PCTL satisfaction. Section 6 describes the experimental results. Finally, Section 7 concludes this paper and provides hints on the possible future works.

2 Related Work

In the literature, few works examine the abstraction of UML and SysML diagrams before verification and the majority rely on the implemented abstraction algorithms within the model checker.

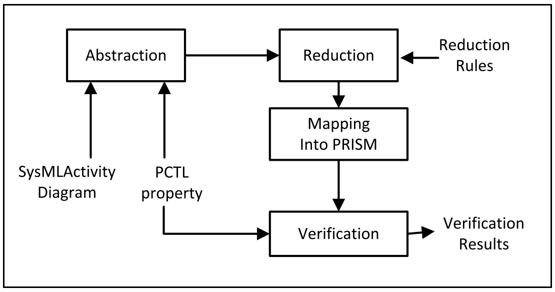


Fig. 1. A Novel Abstraction Approach Overview

Ober et al. [10,11] propose a set of UML model reduction techniques including static analysis, partial order reduction, and model minimization. The abstraction is performed on the semantic model instead of UML diagrams. Westphal [12] exploits the symmetry of UML models in the type of object references to do verification. Shet [13] determines the set of relevant events with respect to the safety property. Daoxi et al. [14] propose an abstraction driven by LTL properties on Promela code of UML behavioral diagrams. Xie and Browne [15,16] propose a verification framework for executable UML (xUML) models. It is based on a user-driven state space reduction procedure. Beek et al. present in [17] a framework called (UMC) for the formal analysis of concurrent systems specified by a collection of UML state machines. It is an on-the-fly based analysis with a user-guided abstraction of the transition system. Gallardo et al. [18] abstract data and events in hierarchical state chart diagrams. They minimize the original access definitions of variables and use a single event name to represent a set of real ones. R. Eshuis [19,20] apply data abstraction on guards and events. In addition, some probabilistic model checker support abstraction, for example PRISM builds the symmetry reduction and LiQuor[1] includes bi-simulation.

In Table 1, we compare our approach to the existing ones. We observe that few of them formalize SysML activity diagrams and prove the soundness of their proposed abstraction approaches. Moreover, our abstraction approach is efficient as it reduces the size of the model by a considerable rate. Furthermore, our mechanism allows to gain advantage from algorithms built within the tool in use.

3 SysML Activity Diagrams

In this section, we describe the SysML activity diagram notation, and we present an optimized and a modified version of the Activity Calculus (AC) [1], to provide a formal syntax and operational semantics for SysML activity diagrams. This formal semantics is useful to prove the soundness of our abstraction.

SysML reuses a subset of UML packages and extends others with specific systems' engineering features, and it covers four main perspectives of systems modeling: structure, behavior, requirements, and parametric. Particularly, SysML activity diagrams are

[1] `http://www.i1.informatik.uni-bonn.de/baier/projectpages/LIQUOR/LiQuor`

Table 1. Comparison with the Related Work

Approach	Design	Probabilistic	Property	Formalization	Soundness
[10,11]					
[12]	✓				
[13]	✓		✓		
[14]			✓		
[15,16]	✓				
[17]	✓				
[18]	✓				
[19,20]	✓				
Our	✓	✓	✓	✓	✓

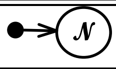
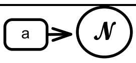
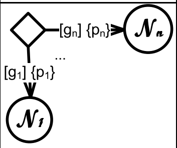
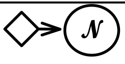
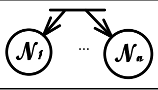
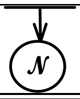
behavioral diagrams used to model system's behavior at various level of abstractions [21]. The main notation of SysML activity diagram can be decomposed into two categories of constructs: activity nodes and activity edges. The former contains three types: activity invocation, object and control nodes. Activity invocation includes receive and send signals, action, and call behavior. Activity control nodes are initial, flow final, activity final, decision, merge, fork, and join nodes. Activity edges are of two types: control flow and object flow. Control flow edges are used to show the execution path through the activity diagram and connects activity nodes. Object flow edges are used to show the flow of data between activity nodes. Concurrency and synchronization are modeled using forks and joins, whereas, branching is modeled using decision and merge nodes. While a decision node specifies a choice between different possible paths based on the evaluation of a guard condition (and/or a probability distribution), a fork node indicates the beginning of multiple parallel control threads. Moreover, a merge node specifies a point from where different incoming control paths follow the same path, whereas a join node allows multiple parallel control threads to synchronize and rejoin.

3.1 Syntax of SysML Activity Diagrams

The UML superstructure specifies basic rules for the execution of the various nodes by explaining textually how tokens (i.e. *locus of control.*) are passed from one node to another [22]. At the beginning, a first token starts flowing from the initial node and moves downstream from one node to another with respect to the foregoing set of control routing rules defined by the control nodes until reaching either an activity final or a flow final node. However, activity diagram semantics as specified in the standard stay informal since it is described informally using textual explanations. Inspired by this concept, we express the Backus-Naur-Form (BNF) of the new version of Activity Calculus (NuAC) that captures the syntax and the execution of activity diagrams. This new version optimizes the syntax presented in [1] and allows multiplicity for fork and decision constructs. Before presenting the NuAC syntax, firstly, we rewrite the SysML activity diagram constructs in the formal way as described in Table 2. The BNF of NuAC is illustrated in Figure 2.

During the execution, the structure of the activity diagram is kept unmodified but the location of the tokens changes. The NuAC syntax was inspired by this idea so that an

Table 2. Mapping Activity Diagram Artifacts into NuAC Syntax

AD Constructs	NuAC Syntax	Description
	$\iota \rightarrowtail \mathcal{N}$	Initial node
◉	$l : \odot$	Activity final node
⊗	$l : \otimes$	Flow final node
	$l : a \rightarrowtail \mathcal{N}$	Action node
 [gₙ] {pₙ} $\mathcal{N}_n$ ... [g₁] {p₁} $\mathcal{N}_1$	$l : Decision($ $(p_1, g_1, \mathcal{N}_1),$ $\cdots,$ $(p_n, g_n, \mathcal{N}_n))$	a Decision node with a convex distribution $\{p_1, \cdots, p_n\}$ with guarded transitions $\{g_1, \cdots, g_n\}$
	$l : Merge(\mathcal{N})$ or l	Merge node specifies the continuation.
	$l : Fork(\mathcal{N}_1, \cdots, \mathcal{N}_n)$	Fork node models the concurrency between n control threads.
	$l : x.Join(\mathcal{N})$ or l	Join node models synchronization. It rejoins a set of input pins. Each pin is specified by an index x.

$$\begin{array}{lllll}
\mathscr{A} & ::= & \varepsilon & \mid \overline{\iota}^k \rightarrowtail \mathcal{N} & \\
\mathcal{N} & ::= & \overline{\mathcal{N}}^n & \mid l : Merge(\mathcal{N}) & \mid l : x.Join(\mathcal{N}) \\
 & \mid & l : Fork(\mathcal{N}, \cdots, \mathcal{N}) & \mid \overline{l : a}^n \rightarrowtail \mathcal{N} & \\
 & \mid & l : Decision((p_1, g_1, \mathcal{N}), \cdots, (p_n, g_n, \mathcal{N})) & & \\
 & \mid & l : \otimes & \mid l : \odot & \mid l
\end{array}$$

Fig. 2. Syntax of New Activity Calculus (NuAC)

NuAC term presents a static structure while tokens are the only dynamic elements. We can distinguish two main syntactic concepts: marked and unmarked terms. A marked NuAC term corresponds to an activity diagram with tokens. An unmarked NuAC term corresponds to the static structure of the diagram. A marked term is typically used to denote a reachable configuration. A configuration is characterized by the set of tokens locations in a given term.

To support multiple tokens, we augment the "overbar" operator with an integer n such that $\overline{\mathcal{N}}^n$ denotes a term marked with n tokens such that $\overline{\mathcal{N}}^1 = \overline{\mathcal{N}}$ and $\overline{\mathcal{N}}^0 = \mathcal{N}$. For the term $\overline{\iota}^k$, k can be either 1 or 0. Multiple tokens are needed when there are loops that encompass in their body a fork node. Furthermore, we use a prefix label for each node (except initial) to uniquely reference it in the case of a backward flow connection. Particularly, labels are useful for connecting multiple incoming flows towards merge and join nodes. Let $\mathscr{L}$ be a collection of labels ranged over by l, l_0, l_1, $\cdots$ and $\mathcal{N}$ be any node (except initial) in the activity diagram. We write $l : \mathcal{N}$ to denote an l-labeled activity node $\mathcal{N}$. The NuAC term $\mathscr{A}$ is built using a depth-first traversal of the activity diagram directed by its activity edges. It is important to note that nodes with multiple incoming edges (e.g. join and merge) are visited as many times as they have incoming edges. Thus, as a syntactic convention, the algorithm uses either the definition term (i.e. $l\colon Merge(\mathcal{N})$ for merge and $l\colon x.Join(\mathcal{N})$ for join) if the current node is visited for the first time or the corresponding label (i.e. l) if the same node is encountered later during the traversal process. We denote by $Decision(-,g,\mathcal{N})$ to express the non-probabilistic decision while p has no value. Also, we denote by $\mathscr{A}[\mathcal{N}]$ to specify $\mathcal{N}$ as a sub term of $\mathscr{A}$.

3.2 Semantics of SysML Activity Diagrams

The execution of SysML activity diagrams is based on token's flow [22]. To give a meaning to this execution, we use structural operational semantics to formally describe how the computation steps of NuAC atomic terms take place. The NuAC semantics rules shown by Figure 3 is based on the informally specified tokens-passing rules defined in [22].

INIT-1 $\quad \overline{\iota} \rightarrowtail \mathcal{N} \xrightarrow{\iota}_1 \iota \rightarrowtail \overline{\mathcal{N}}$

ACT-1 $\quad \overline{l\colon a}^m \rightarrowtail \mathcal{N} \xrightarrow{l}_1 \overline{l\colon a}^{m-1} \rightarrowtail \overline{\mathcal{N}} \quad \forall m>0$

ACT-2 $\quad \overline{\overline{l\colon a}^m \rightarrowtail \mathcal{N}}^n \longrightarrow_1 \overline{l\colon a}^{m+n} \rightarrowtail \mathcal{N} \quad \forall m>0$

FORK-1 $\quad \overline{l\colon Fork(\mathcal{N}_1,\cdots,\mathcal{N}_n)}^m \xrightarrow{l}_1 \overline{l\colon Fork(\overline{\mathcal{N}_1},\cdots,\overline{\mathcal{N}_n})}^{m-1} \quad \forall m>0$

PDEC-1

$$\overline{l\colon Decision((p_1,g_1,\mathcal{N}_1),\cdots,(p_i,g_i,\mathcal{N}_i),\cdots,(p_n,g_n,\mathcal{N}_n))}^m \xrightarrow{l}_{p_i} \overline{l\colon Decision((p_1,g_1,\mathcal{N}_1),\cdots,(p_i,g_i,\overline{\mathcal{N}_i}),\cdots,(p_n,g_n,\mathcal{N}_n))}^{m-1} \quad \forall m>0$$

MERG-1 $\quad \overline{l\colon Merge(\mathcal{N})}^m \xrightarrow{l}_1 \overline{l\colon Merge(\overline{\mathcal{N}})}^{m-1} \quad \forall m>0$

MERG-2 $\quad \mathscr{A}[\overline{l\colon Merge(\mathcal{N})}^m,\overline{l}^k] \longrightarrow_1 \mathscr{A}[\overline{l\colon Merge(\mathcal{N})}^{m+k},l] \; m,k\geq 1$

JOIN-1

$$\mathscr{A}[\overline{l\colon x.Join(\mathcal{N})}^m,\overline{l}_x^k] \xrightarrow{l}_1 \mathscr{A}[\overline{l\colon x.Join(\overline{\mathcal{N}})}^{m+k-1},l_x] \;; x>1 \quad \forall m,k\geq 1$$

FLOWFINAL $\quad \mathscr{A}[\overline{l\colon \otimes}] \xrightarrow{l}_1 \mathscr{A}$

FINAL $\quad \mathscr{A}[\overline{l\colon \odot}] \xrightarrow{l}_1 |\mathscr{A}|$

PROG $\quad \dfrac{\mathcal{N} \xrightarrow{\alpha}_q \mathcal{N}'}{\mathscr{A}[\mathcal{N}] \xrightarrow{\alpha}_q \mathscr{A}[\mathcal{N}']}$

Fig. 3. NuAC Operational Semantic Rules

We define Σ as the set of non-empty actions labeling the transitions (i.e. the alphabet of NuAC, to be distinguished from action nodes in activity diagrams). An element $\alpha \in \Sigma$ is the label of the executing active node. Let Σ^o be $\Sigma \cup \{o\}$ where o denotes the empty action. Let p be probability values such that $p \in \{-\} \cup]0,1]$. The general form of a transition is $\mathscr{A} \xrightarrow{\alpha}_p \mathscr{A}'$. The probability value specifies the likelihood of a given transition to occur and it is denoted by $P(\mathscr{A},\alpha,\mathscr{A}')$. The semantics of SysML activity diagrams is expressed using $\mathscr{A}$ as a result of the defined inference rules that can be described in terms of Probabilistic Automata (PA) [23].

4 The Abstraction Approach

This section describes our approach to abstract a design model expressed as SysML activity diagrams. To do so, we propose essentially two abstraction algorithms.

The first one hides the action nodes of the SysML activity diagram that are not part of the atomic propositions of the PCTL property to be verified à la [5][2]. Initially, SysML activity diagram is an action-based diagram, where actions are the executed entities and guards denote the branching choices between alternative actions. The control nodes are essentially used to coordinate the execution of these actions. Thus, PCTL properties essentially comprise propositions on actions and guards. The atomic propositions of a PCTL property ϕ are formed with a set of independent variables $var(\phi)$ such that $var(\phi) \subseteq \{a_i : i \leq n\} \cup \{g_i : i \leq m\}$ where a_i is a variable corresponding to an action, g_i is a guard variable, and respectively, n and m are the number of actions and guards.

```
$\mathcal{A}bs: NuAC \times Var(\Phi) \rightarrow NuAC$
$\mathcal{A}bs(\mathcal{N}, var(\phi))$ = Case ($\mathcal{N}$) of
        $l : a \rightarrowtail \mathcal{N}'$     ⇒ if $a \in var(\Phi)$ then
                                          $l : a \rightarrowtail \mathcal{A}bs(\mathcal{N}', var(\phi))$
                                  else
                                          $\varepsilon \rightarrowtail \mathcal{A}bs(\mathcal{N}', var(\phi))$
                                  end
    Otherwise     $\mathcal{A}bs(\mathcal{N}, var(\phi))$
```

Listing 1.1. Action Nodes Abstraction Algorithm

The procedure $\mathcal{A}bs$ presented in Listing 1.1 abstracts a given SysML activity diagram with respect to the action variables that are not part of the set $var(\phi)$. It takes as input a NuAC term $\mathcal{N}$ along with $var(\phi)$ and generates an abstract term such that $\mathcal{A}bs(\mathcal{N}, var(\phi)) = \widehat{\mathcal{N}}$ and $var(\widehat{\mathcal{N}}) = var(\phi)$.

The second algorithm minimizes an activity diagram by merging specific control nodes while preserving the number of tokens and their control paths. This is achieved by preventing the modification of guarded and probabilistic choices. The procedure $\mathcal{M}inim$ presented in Listing 1.2 aims at merging consecutive control nodes of the same type. For that, we propose an equivalence relation inspired by the structural congruence relation defined by Milner [24]. This equivalence relation satisfies the following rules:

1. $l : Fork(\ldots, \mathcal{N}_i, \ldots) \equiv l : Fork(\ldots, \mathcal{N}_k, \ldots, \mathcal{N}_m, \ldots)$ if $\mathcal{N}_i \equiv l' : Fork(\mathcal{N}_k, \ldots, \mathcal{N}_m)$.
2. $l : x.Join(\mathcal{N}') \equiv l : z.Join(\mathcal{N})$ if $\mathcal{N}' \equiv l' : y.Join(\mathcal{N})$ and $z = x + y$.
3. $l : Merge(\mathcal{N}) \equiv l : Merge(\mathcal{N}')$ and $l = l'$ if $\mathcal{N} \equiv l' : Merge(\mathcal{N}')$.
4. $l : Decision(\ldots, (p, g, \mathcal{N}), \ldots) \equiv l : Decision(\ldots, (p \times p_k, g \wedge g_k, \mathcal{N}_k), \ldots, (p \times p_m, g \wedge g_m, \mathcal{N}_m), \ldots)$ if $\mathcal{N} \equiv l' : Decision((p_k, g_k, \mathcal{N}_k), \ldots, (p_m, g_m, \mathcal{N}_m))$.

Basically, $\mathcal{A}bs$ produces a new model that includes mainly the specified actions in the property ϕ where other actions are considered as silent action (Milner [24]). Thus, the

[2] Close to the cone of influence of [5].

```
$\mathcal{M}inim: \ NuAC \rightarrow NuAC$
$\mathcal{M}inim(\mathcal{N})$ = Case ($\mathcal{N}$) of
    $l{:}Merge(\mathcal{N}')$ ⇒ Case ($\mathcal{N}'$) of
            $l'{:}Merge(\mathcal{N}'')$ ⇒ $l{:}Merge(\mathcal{M}inim(\mathcal{N}''))$ and $Rewrite(\mathcal{N}'', l', l)$
            Otherwise ⇒ $l{:}Merge(\mathcal{M}inim(\mathcal{N}'))$
    $l{:}x.Join(\mathcal{N}')$ ⇒ Case ($\mathcal{N}'$) of
            $l'{:}y.Join(\mathcal{N}'')$ ⇒ let
                                z = x+y
                          in
                              $l{:}z.Join(\mathcal{M}inim(\mathcal{N}''))$ and $Rewrite(\mathcal{N}'', l', l_{x+j})$
            Otherwise ⇒ $l{:}Join(\mathcal{M}inim(\mathcal{N}'))$
    $l{:}Fork(\mathcal{N}_1,\cdots,\mathcal{N}_i,\cdots,\mathcal{N}_n)$ ⇒ Case ($\mathcal{N}_i$) of
            $l'{:}Fork(\mathcal{N}_k',\cdots,\mathcal{N}_m')$ ⇒
                      $l{:}Fork(\mathcal{M}inim(\mathcal{N}_1'),\cdots,\mathcal{M}inim(\mathcal{N}_k'),\cdots,$
                            $\mathcal{M}inim(\mathcal{N}_m'),\cdots,\mathcal{M}inim(\mathcal{N}_n))$
            Otherwise ⇒ $l{:}Fork(\mathcal{M}inim(\mathcal{N}_1),\cdots,\mathcal{M}inim(\mathcal{N}_i),\cdots,\mathcal{M}inim(\mathcal{N}_n))$
    $l{:}Decision((p_1,g_1,\mathcal{N}_1),\cdots,(p_i,g_i,\mathcal{N}_i),\cdots,(p_n,g_n,\mathcal{N}_n))$ ⇒ Case ($\mathcal{N}_i$) of
            $l{:}Decision((p_1',g_1',\mathcal{N}_1'),\cdots,(p_j',g_j',\mathcal{N}_j'),\cdots,(p_m',g_m',\mathcal{N}_m'))$ ⇒
                      $l{:}Decision((p_1,g_1,\mathcal{N}_1),\cdots,(p_i\times p_1',g_i\wedge g_1',\mathcal{N}_1'),\cdots,$
                            $(p_i\times p_j',g_i\wedge g_j',\mathcal{N}_j'),\cdots,(p_i\times p_m',g_i\wedge g_m',\mathcal{N}_m'),$
                            $\cdots,(p_n,g_n,\mathcal{N}_n))$
            Otherwise ⇒ $l{:}Decision((p_1,g_1,\mathcal{M}inim(\mathcal{N}_1)),\cdots,(p_n,g_n,\mathcal{M}inim(\mathcal{N}_n)))$
    Otherwise $\mathcal{M}inim(\mathcal{N})$
```

Listing 1.2. Control Nodes Abstraction Algorithm

resulting activity diagram has a reduced number of actions, which increases the occurrence of consecutive control nodes. Consequently, applying $\mathcal{A}bs$ first is more efficient[3] as showed by the following proposition.

Proposition 1 (Application Order). *For a SysML activity diagram "$\mathscr{A}$" and a property "ϕ", we have:* $\mathcal{M}inim(\mathcal{A}bs(\mathcal{M}inim(\mathscr{A}),\phi)) \equiv \mathcal{M}inim(\mathcal{A}bs(\mathscr{A},\phi))$.

Proof. Let $M_1 \equiv \mathcal{M}inim(\mathscr{A}), M_2 \equiv \mathcal{A}bs(M_1,\phi)$ and $M_3 \equiv \mathcal{M}inim(M_2)$, we have:

1. $M_1 \equiv \mathcal{M}inim(\mathscr{A}) \Leftrightarrow$ if $\exists l : \mathcal{N}_k \rightarrowtail \mathcal{N}_m \in \mathscr{A}$, then $l : \mathcal{N}_k \rightarrowtail \mathcal{N}_m$ is replaced by $l : \mathcal{N}_{km}$ if one of control merging rules is satisfied.
2. $M_2 \equiv \mathcal{A}bs(M_1,\phi)) \Leftrightarrow \forall a \notin var(\phi) : \mathcal{A}bs(\overline{l : a}^n \rightarrowtail \mathcal{N}, var(\phi)) = \overline{l : \varepsilon}^n \rightarrowtail \mathcal{N}$. In fact, $\mathcal{A}bs$ produces new consecutive control nodes and preserve the diagram structure.

It is clear that the second step has no effect on the first one and vice versa. In addition, applying $\mathcal{M}inim$ two times successively is equivalent to apply it once. Thus, the proposition holds. □

5 Abstraction Soundness and Property Preservation

In this section, we first prove the soundness of our proposed abstraction algorithms. Next, we prove that our algorithms preserve the satisfaction of PCTL properties.

[3] In term of time execution.

5.1 Abstraction Soundness

Our aim is to prove that our abstraction algorithm is sound and preserves PCTL properties. Let $\mathscr{A}$ be a SysML activity diagram and $M_{\mathscr{A}}$ be its corresponding PA constructed by the NuAC operational semantics $\mathcal{S}$ such that $\mathcal{S}(\mathscr{A}) = M_{\mathscr{A}}$. And, let δ be our abstraction composed of $\mathcal{A}bs$ and $\mathcal{M}inim$ algorithms such that $\delta(\mathscr{A}) = \widehat{\mathscr{A}}$, where $\widehat{\mathscr{A}}$ denotes the abstracted SysML activity diagram. Let $M_{\widehat{\mathscr{A}}}$ be its corresponding PA defined using the NuAC operational semantics $\mathcal{S}$ such that $\mathcal{S}(\widehat{\mathscr{A}}) = M_{\widehat{\mathscr{A}}}$. As illustrated in Figure 4, proving the soundness of our algorithm is to find a relation $\mathcal{R}$ between $M_{\mathscr{A}}$ and $M_{\widehat{\mathscr{A}}}$. The formal description of $M_{\mathscr{A}}$ is represented in Definition 1 where $Dist(S)$ is a convex distribution over a set S.

$$\begin{tikzcd}
\mathscr{A} \arrow[r, "\mathcal{S}"] \arrow[d, "\delta"'] & M_{\mathscr{A}} \arrow[d, dashed, "\mathcal{R}"] \\
\widehat{A} \arrow[r, "\mathcal{S}"] & M_{\widehat{\mathscr{A}}}
\end{tikzcd}$$

Fig. 4. Abstraction Correctness

Definition 1 (New Activity Calculus PA). *A probabilistic automata of an activity calculus term $\mathscr{A}$ is a tuple $M_{\mathscr{A}} = (\overline{s},\ \mathrm{L},\ \mathrm{S},\ \Sigma,\ \delta)$ where:*

- *$\overline{s}$ is an initial state, such that $\mathrm{L}(\overline{s})=\overline{\mathscr{A}}$,*
- *L is a labeling function,*
- *S is a finite set of states reachable from $\overline{s}$, such that, $\mathrm{S} = \{s_{i:0\le\ i\ \le\ n} | \mathrm{L}(s_i) \in \{\overline{\mathscr{N}}\}\}$,*
- *Σ is a finite set of actions corresponding to the alphabet of $\mathscr{A}$,*
- *$\delta : \mathrm{S} \times \Sigma \rightarrow Dist(S)$ is a (partial) probabilistic transition function such that, for each $s \in S$ and $\alpha \in \Sigma$ assigns a probabilistic distribution $\mu \in Dist(S)$ such that:*
  - *For each $S' \subseteq S$, $S' = \{s_{i:0\le i\le n} : s \xrightarrow{\alpha}_{p_i} s_i\}$. Each $s \xrightarrow{\alpha}_{p_i} s_i$ satisfies one NuAC semantic rule and $\mu(S') = \sum_{i=0}^{n} p_i = \sum_{i=0}^{n} \mu(s_i) = 1$.*
  - *For each transition $s \xrightarrow{\alpha}_1 s''$ satisfying a NuAC semantic rule, μ is defined such that $\mu(s'') = 1$.*

To define the relation $M_{\mathscr{A}}\ \mathcal{R}\ M_{\widehat{\mathscr{A}}}$, we introduce the notion of weak relation while δ abstracts away from invisible actions. Formally, the probabilistic weak simulation [23] relation defined in Definition 2 introduces the notion of observable action a preceded and followed invisible steps. We denote a weak transition by ($s \overset{a}{\Longrightarrow} P$) where P is the distribution over states reached from s through a sequence of combined steps.

Definition 2 (Weak Probabilistic Simulation). *A weak probabilistic simulation between two probabilistic automata M_1 and M_2 is a relation $\mathscr{R} \sqsubseteq S_1 \times S_2$ such that:*

1. *each start state of M_1 is related to at least one start state of M_2,*
2. *for each pair of states $s_1 \mathscr{R} s_2$ and each transition $s_1 \xrightarrow{a} P_1$ of M_1, there exist a weak combined transition $s_2 \overset{a}{\Longrightarrow} P_2$ of M_2 such that $P1 \sqsubseteq_{\mathscr{R}} P2$.*

Here, $\sqsubseteq_{\mathscr{R}}$ is the lifting of $\mathscr{R}$ to a probability space. It is achieved by finding a weight function [23] that associates each state of M_1 with others in M_2 by a certain probability value. It is defined below.

Definition 3 (Weight Function). *A function $\triangle : S \times S' \to [0,1]$ is a weight function for the two distribution $\mu_1, \mu_2 \in Dist(S)$ w.r.t. $\mathscr{R} \sqsubseteq S \times S'$ iff:*

1. $\triangle(s_1,s_2) > 0 \Rightarrow (s_1,s_2) \in \mathscr{R}$,
2. $\forall\, s_1 \in S : \Sigma_{s_2 \in S} \triangle(s_1,s_2) = \mu_1(s_1)$,
3. $\forall\, s_2 \in S : \Sigma_{s_1 \in S} \triangle(s_1,s_2) = \mu_2(s_2)$ then $s_1 \mathscr{R} s_2$.

For our proof, we stipulate herein the abstraction relation denoted by $\mathscr{A} \sqsubseteq_{\mathscr{R}'} \widehat{\mathscr{A}}$ between SysML activity diagrams $\mathscr{A}$ and $\widehat{\mathscr{A}}$.

Definition 4 (Abstraction Relation). *An abstraction relation is a weak probabilistic relation between a SysML activity diagram $\mathscr{A}$ and its abstracted model $\widehat{\mathscr{A}}$ by applying δ algorithm.*

In the following, we present the soundness of our algorithm. Let $\mathscr{M}_{\mathscr{A}}$ be a PA representing the semantic of the NuAC term $\mathscr{A}$, $\mathscr{M}_{\widehat{\mathscr{A}}}$ is the PA representing the semantics of $\widehat{\mathscr{A}}$ such that $\widehat{\mathscr{A}} = \delta(\mathscr{A}, \phi)$. Proving that δ is sound means proving there exists a weak probabilistic simulation between $\mathscr{M}_{\mathscr{A}}$ and $\mathscr{M}_{\widehat{\mathscr{A}}}$, i.e. $\mathscr{M}_{\mathscr{A}} \sqsubseteq_{\mathscr{R}} \mathscr{M}_{\widehat{\mathscr{A}}}$.

Theorem 1 (Soundness). *The abstraction algorithm δ is sound.*

Proof. The proof follows a structural induction on NuAC terms. In an inductive way, we select the $\overline{a} \rightarrowtail \mathscr{N}$ case to prove the soundness for $\mathcal{A}bs$ procedure procedure. The remaining cases can be proved similarly for both $\mathcal{A}bs$ and $\mathcal{M}inim$ functions.

Let $L(s_1) = \overline{a} \rightarrowtail \mathscr{N} \Rightarrow \exists\, s_1' : s_1 \to s_1'$ by applying ACT-1 rule such that: $s_1' = a \rightarrowtail \overline{\mathscr{N}} \Rightarrow \mu_1(s_1') = 1$. By considering s_2 as the abstracted state of s_1, $L(s_2) = Abs(L(s_1))$, we will have two cases:

1. $a \in var(\phi) : L(s_2) = Abs(\overline{a} \rightarrowtail \mathscr{N}) = \overline{a} \rightarrowtail Abs(\mathscr{N})$. By applying ACT-1, $\exists\, s_2' : s_2 \to s_2'$ such that: $L(s_2') = a \rightarrowtail \overline{Abs(\mathscr{N})} \Rightarrow \mu_2(s_2') = 1$. Then, it exists a weight function $\triangle$ for $\mathscr{R} = (s_1', s_2')$ such that:
   (a) $\triangle(s_1',s_2') = 1 \Rightarrow \triangle(s_1',s_2') = \mu_1(s_1')$, and
   (b) $\triangle(s_2',s_1') = 1 \Rightarrow \mu_2(s_2') = \triangle(s_2',s_1')$, then
   (c) $\triangle(s_1,s_2) > 0 \Rightarrow s_1 \sqsubseteq_{\mathscr{R}} s_2$
2. $a \notin var(\phi)$:
   $L(s_2) = Abs(s_1) = Abs(\overline{a} \rightarrowtail \mathscr{N}) = \overline{\varepsilon} \rightarrowtail \underline{Abs(\mathscr{N})} \Rightarrow \exists\, s_2' : s_2 \to s_2'$.
   By applying ACT-1 rule such that $L(s_2') = \varepsilon \rightarrowtail \overline{\mathscr{N}} \Rightarrow \mu_2(s_2') = 1$.
   It exist a weight function $\triangle$ for $\mathscr{R} = (s_1', s_2')$ such that:
   (a) $\triangle(s_1',s_2') = 1 \Rightarrow \triangle(s_1',s_2') = \mu_1(s_1')$, and
   (b) $\triangle(s_2',s_1') = 1 \Rightarrow \mu_2(s_2') = \triangle(s_2',s_1')$, then
   (c) $\triangle(s_1,s_2) > 0 \Rightarrow s_1 \sqsubseteq_{\mathscr{R}} s_2$

It is clear that, the marked NuAC term $\overline{\mathscr{A}}$ is the unique initial state of $M_{\mathscr{A}}$ corresponding to the unique initial state in $M_{\widehat{\mathscr{A}}}$. By following the same style of proof, we find: $\overline{\mathscr{A}} \sqsubseteq_{\mathscr{R}} M_{\widehat{\mathscr{A}}}$, which confirms that Theorem 1 holds. □

5.2 Property Preservation

In order to perform model-checking, a property should be specified. We selected PCTL to express such property. Formally, its syntax is given by the following BNF grammar:

$$\begin{array}{l} \phi ::= \top \mid a \mid \phi \wedge \phi \mid \neg\phi \mid \mathrm{P}_{\bowtie p}[\psi] \\ \psi ::= \mathrm{X}\phi \mid \phi \mathrm{U}^{\leq k}\phi \mid \phi \mathrm{U}\phi \end{array}$$

Where a is an atomic proposition, $k \in \mathbb{N}$, $p \in [0,1]$, and $\bowtie \in \{<, \leq, >, \geq\}$. Also, other useful operators can be derived such as:

- Future: $F\phi \equiv \top\ U\ \phi$ or $F^{\leq\, k}\phi \equiv \top\ U^{\leq\, k}\ \phi$.
- Generally: $G\phi \equiv \neg(F\neg\phi)$ or $G^{\leq\, k}\phi \equiv \neg(F^{\leq\, k}\neg\phi)$.

To specify a satisfaction relation of a PCTL formula in a state s, a class of adversaries (Adv) has been defined [8] to solve the nondeterminism decision. Hence, a PCTL formula should be satisfied under all adversaries. The satisfaction relation ($\models_{Adv}$) of PCTL formula is defined as follows:

- $s \models_{Adv} \top$
- $s \models_{Adv} a \Leftrightarrow a \in L(s)$
- $s \models_{Adv} \phi_1 \wedge \phi_2 \Leftrightarrow s \models_{Adv} \phi_1 \wedge s \models_{Adv} \phi_2$
- $s \models_{Adv} \neg\phi \Leftrightarrow s \not\models_{Adv} \phi$
- $s \models_{Adv} \mathrm{P}_{\bowtie p}[\psi] \Leftrightarrow P(\{\pi \in IPath_{M,s} | \pi_{Adv} \models \psi\}) \bowtie p$
- $\pi \models_{Adv} X\phi \Leftrightarrow \pi(1) \models_{Adv} X\phi$
- $\pi \models_{Adv} \phi_1\ U^{\leq\, k}\ \phi_2 \Leftrightarrow \exists\, i \geq k.(\pi(i) \models_{Adv} \phi_2 \wedge \pi(j) \models_{Adv} \phi_1\ \forall\, j < i)$
- $\pi \models_{Adv} \phi_1\ U\phi_2 \Leftrightarrow \exists\, k \geq 0.\ \pi \models_{Adv} \phi_1\ U^{\leq\, k}\ \phi_2$

Here, we prove by induction on the structure of the PCTL grammar, except for the neXt operator ($PCTL_{\backslash X}$), that a formula (ϕ) holds in the concrete model if it holds in the abstracted model as stated in Theorem 2.

Theorem 2 (PCTL Preservation). *For two models M and $\widehat{M}$ such that $M \sqsubseteq_{\mathscr{R}} \widehat{M}$. If ϕ is a $PCTL_{\backslash X}$ property, then we have: $(\widehat{M} \models \phi) \Rightarrow (M \models \phi)$.*

Proof. To prove the preservation of PCTL properties, we follow an inductive reasoning on the PCTL structure. □

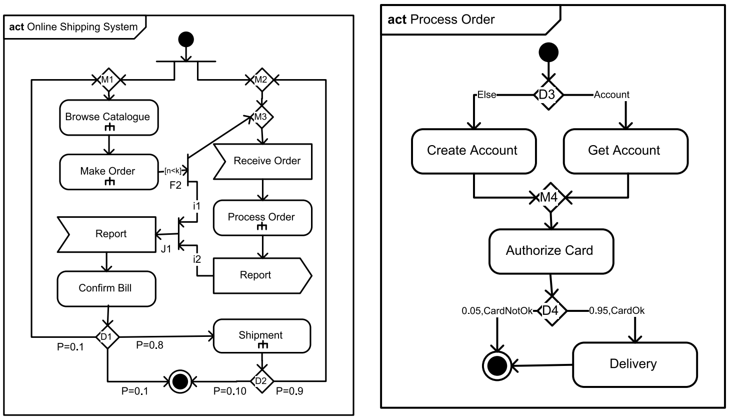
6 Experimental Results

In this section, we apply our abstraction algorithm on an online shopping system case study [25]. In order to show our abstraction efficiency, we use our translation algorithm [2] to map SysML activity diagrams into PRISM input language where we verify PCTL properties on both: the concrete and the abstract models. This is done in the purpose of providing experimental results demonstrating the efficiency and the validity of our abstraction.

To this end, we compare the results perspective of the verification cost (β) in terms of time verification and the abstraction efficiency (η) in terms of model's time construction. With respect to the verification cost, we measure both the time required to construct the model, denoted by T_c, and the time required for verifying the property, denoted by T_v. The verification cost is given by $\beta = 1 - \frac{|Tv(\widehat{M})|}{|Tv(M)|}$. Concerning the abstraction efficiency, we measure the number of transitions ($\#t$) for both concrete and abstract diagrams. It is given by $\eta = 1 - \frac{|Tc(\widehat{M})|}{|Tc(M)|}$. The result of the verification of a property is denoted by Res.

6.1 Model Description

The online shopping system aims at providing services for purchasing online items. Figure 5a illustrates the corresponding SysML activity diagram[4]. It contains four call-behavior actions[5], which are: "Browse Catalogue", "Make Order", "Process Order" and "Shipment". As example, Figure 5b expands the call behavior action "Process Order".



(**a**) Online Shopping System. (**b**) Process Order.

Fig. 5. The Concrete SysML Activity Diagram

6.2 Property Specification

In order to verify the functional requirements of the online shopping system, we propose the following properties and its related PCTL expressions.

1. For each order, what is the minimum probability value to make a delivery? From this expression, it is clear that only the main diagram (M) and "Process Order" behavior (M_3) are affected. We express this property in PCTL as follows, where n is the order number and K is the maximum allowed number to make an order.
$$Pmin = ?[(n \leq K)\ U\ (Delivery)]$$

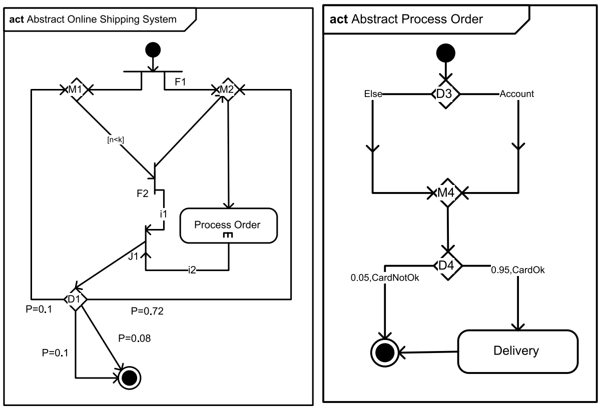
[4] This diagram is not symmetric which mean that we can not benefit from the symmetry reduction built within PRISM.

[5] Each call-behavior action is represented by its proper diagram.

2. After browsing the catalogue, what is the minimum probability value to ship a selected item? The propositions of this property belong to the main diagram (M) and both behaviors: "Browse Catalogue", "Process Order" and "Shipment" where n and m representing the order and the shipment numbers, respectively. Its corresponding PCTL expression is:
$Pmin = ?[((SelectItem \wedge m = n \wedge m \leq K) \Rightarrow F(Delivery)) \Rightarrow F(Shipment)]$.

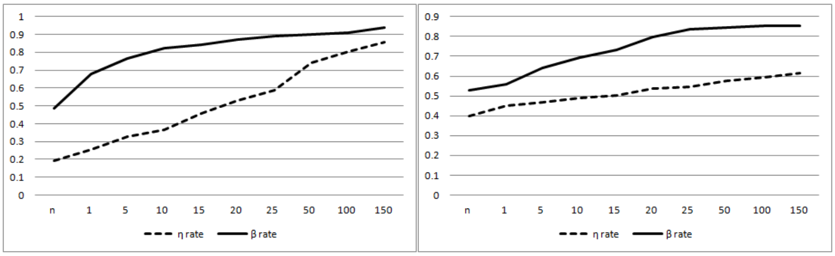
6.3 Verification Result

After applying our abstraction algorithm, we obtain the abstract SysML activity diagrams shown in Figure 6 for property #1. Due to the lack of space, we are not providing the abstract SysML activity diagram related to property #2.



(a) Online Shopping System. (b) Process Order.

Fig. 6. The Abstract SysML Activity Diagram



(a) Property #1. (b) Property #2.

Fig. 7. Shopping Online System Abstraction Rate

Figure 7a illustrates both abstraction rates in term of model size and computation time for the verification of property #1. For the verification of property #2, Figure 7b shows the evolution of the abstraction rate in terms of model size and computation time. In summary, the results demonstrate that the abstraction efficiency is important especially when the model' size is growing. Furthermore, they show that our abstraction algorithm actually preserves the verification results.

7 Conclusion

In this paper, we presented an automatic abstraction approach to improve the scalability of probabilistic model-checking in general and more especially for the verification of SysML activity diagrams. Also, we proposed a calculus dedicated to these diagrams. We have proved the soundness of our algorithm by defining a probabilistic weak simulation relation between the semantics of the abstract and the concrete models. In addition, the preservation of the satisfaction of $\text{PCTL}_{\setminus X}$ properties is proved. Finally, we demonstrated the effectiveness of our approach by applying it on an online shopping system application.

As future work, we would like to extend our approach by investigating several directions. First, we intend to integrate our algorithm within PRISM model checker. Second, we plan to apply our proposed abstraction on a composition of SysML activity diagrams. Next, we explore other abstraction approaches especially data abstraction targeting events and guards reduction. Finally, we intend to investigate reducing the property within the model at the same time to check $\widehat{\text{M}} \models \widehat{\text{P}}$ instead of $\text{M} \models \text{P}$.

References

1. Debbabi, M., Hassane, F., Jarraya, Y., Soeanu, A., Alawneh, L.: Verification and Validation in Systems Engineering - Assessing UML / SysML Design Models. Springer (2010)
2. Ouchani, S., Jarraya, Y., Mohamed, O.A.: Model-based systems security quantification. In: PST, pp. 142–149 (2011)
3. Baier, C., Katoen, J.P.: Principles of Model Checking, MIT Press (May 2008)
4. Xin-feng, Z., Jian-dong, W., Xin-feng, Z., Bin, L., Jun-wu, Z., Jun, W.: Methods to tackle state explosion problem in model checking. In: Proceedings of the 3rd Int. Conf. on IITA, pp. 329–331. IEEE Press, NJ (2009)
5. Clarke, E.M., Grumberg, O., Peled, D.A.: Model Checking. MIT Press (1999)
6. Bérard, B., Bidoit, M., Finkel, A., Laroussinie, F., Petit, A., Petrucci, L., Schnoebelen, P.: Systems and Software Verification. Springer (2001)
7. OMG, OMG Systems Modeling Language (OMG SysML) Specification, Object Management Group, OMG Available Specification (September 2007)
8. Forejt, V., Kwiatkowska, M., Norman, G., Parker, D.: Automated Verification Techniques for Probabilistic Systems. In: Bernardo, M., Issarny, V. (eds.) SFM 2011. LNCS, vol. 6659, pp. 53–113. Springer, Heidelberg (2011)
9. P. Team. PRISM - Probabilistic Symbolic Model Checker, `http://www.prismmodelchecker.org` (last visited November 2011)
10. Ober, I., Graf, S., Ober, I.: Model Checking of UML Models via a Mapping to Communicating Extended Timed Automata. In: SPIN 2004. LNCS, vol. 2989 (2004)
11. Ober, I., Graf, S., Ober, I.: Validating Timed UML models by simulation and verification. In: Workshop SVERTS, San Francisco (October 2003)

12. Westphal, B.: LSC Verification for UML Models with Unbounded Creation and Destruction. Electronic Notes in Theoretical Computer Science 144, 133–145 (2006)
13. Prashanth, C.M., Shet, K.C.: Efficient Algorithms for Verification of UML Statechart Models. Journal of Software 4, 175–182 (2009)
14. Daoxi, C., Guangquan, Z., Jianxi, F.: Abstraction framework and complexity of model checking based on the Promela models. In: 4th International Conference on Computer Science Education, ICCSE 2009, pp. 857–861 (July 2009)
15. Xie, F., Browne, J.C.: Integrated State Space Reduction for Model Checking Executable Object-Oriented Software System Designs. In: Kutsche, R.-D., Weber, H. (eds.) FASE 2002. LNCS, vol. 2306, pp. 64–79. Springer, Heidelberg (2002)
16. Xie, F., Levin, V., Browne, J.C.: ObjectCheck: A Model Checking Tool for Executable Object-Oriented Software System Designs. In: Fundamental Approaches to Software Engineering, pp. 331–335 (2002)
17. ter Beek, M.H., Fantechi, A., Gnesi, S., Mazzanti, F.: A State/Event-Based Model-Checking Approach for the Analysis of Abstract System Properties. Sci. Comput. Program. 76, 119–135 (2011)
18. Yang, H.: Abstracting UML Behavior Diagrams for Verification. In: Software Evolution With Uml And Xml, pp. 296–320. IGI Publishing, Hershey (2005)
19. Eshuis, R.: Symbolic Model Checking of UML Activity Diagrams. ACM Transactions on Software Engineering and Methodology 15 (2006)
20. Eshuis, R., Wieringa, R.: Tool support for verifying uml activity diagrams. IEEE Transactions on Software Engineering 30 (2004)
21. Holt, J., Perry, S.: SysML for Systems Engineering. Institution of Engineering and Technology Press (January 2007)
22. OMG, OMG Unified Modeling Language: Superstructure 2.1.2, Object Management Group (November 2007)
23. Segala, R.: A Compositional Trace-Based Semantics for Probabilistic Automata. In: Lee, I., Smolka, S.A. (eds.) CONCUR 1995. LNCS, vol. 962, pp. 234–248. Springer, Heidelberg (1995)
24. Milner, R.: Communicating and mobile systems - the Pi-calculus. Cambridge University Press (1999)
25. Gomaa, H.: Software Modeling and Design: UML, Use Cases, Patterns, and Software Architectures. Cambridge University Press (2011)