2009 16th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems

On the Meaning of SysML Activity Diagrams

Yosr Jarraya, Mourad Debbabi, and Jamal Bentahar
Computer Security Laboratory,
Concordia Institute for Information Systems Engineering, Concordia University,
Montreal, Quebec, Canada.
Emails:{y_jarray,debbabi,bentahar}@encs.concordia.ca

Abstract

In this paper, we aim to ascribe a meaning to SysML activity diagrams. To this end, we propose a dedicated algebraic-like language, namely activity calculus, and an operational semantics that provides a rigorous and intuitive operational understanding of the behavior captured by the diagram. The semantics covers advanced control flows such as unstructured loops and concurrent control flows. Furthermore, our approach allows non well-formed control flows, with mixed and nested forks and joins. The probabilistic behaviors as specified in SysML are also considered. This formalization allows us to build a sound framework for the verification and validation of systems design expressed in SysML activity diagrams.

1. Introduction

Modern societies intensively use and rely on systems that are more and more complex. This complexity is mainly due to the technological advances and the ever-increasing demand for sophisticated products such as consumer electronics and software-intensive systems. We may observe in today's systems concurrency and parallelism, as well as timed and probabilistic behaviors. The design, development, and guarantee of high reliability and well performance of such systems have become a challenge. Systems Engineering (SE) [1] is the interdisciplinary approach that integrates elements of many disciplines including but not limited to system modeling and simulation, requirements and specification definitions, and software engineering. In order to support the realization of successful systems, the Object Management Group (OMG) and the International Council On Systems Engineering (INCOSE) collaborated on providing a general-purpose systems modeling language, namely SysML [2], that supports specification, analysis, design, verification and validation of complex systems. SysML reuses a subset of the UML 2.0 and adds new diagrams while modifying some others. The SysML's diagrams cover four main perspectives of systems modeling: structure, behavior, requirements, and parametrics.

The research reported in this paper is partially supported by an NSERC scholarship in collaboration with Ericsson Canada and PROMPT Quebec.

In this paper, we focus on the behavioral aspect in systems design, mainly on SysML activity diagrams. They are compared to the Enhanced Functional Flow Block Diagrams (EFFBD) for functional flow modeling, which are widely used by systems engineers [2], [3]. Moreover, activity diagrams support computational and business processes modeling and the use cases detailed specification. The wide utilization of this type of diagram in the design makes its analysis, its rigorous understanding, and its formal assessment a worthy task. In the following, we propose to study the formal semantics of the SysML 1.0 activity diagrams. In the state of the art, some initiatives [4], [5] define formally the UML 1.x activity diagrams semantics, which is tightly related to the semantics of statecharts. However, this was modified in UML 2.x, and hence in SysML, which makes the aforementioned initiatives less applicable. Furthermore, most of the research initiatives within UML 2.0 activity diagrams use existing formalisms with well-established semantic domains. Examples of these formalisms include Communicating Sequential Processes (CSP) in [6] the Interactive Markov Chain (IMC) in [7], and variants of Petri nets formalism such as in [8]–[10]. Referring to the OMG UML 2.0 specification document [11], the activity diagrams were redesigned with Petri nets semantics in mind [11]. Therefore, one might inadvertently assume a straightforward Petri-like semantics for activity diagrams. However, they cannot be fully mapped to one specific variant of Petri nets, as claimed in [12]. Moreover, the algebraic interpretations of activity diagram using existing process algebra are not intuitive and impose unnecessary limitations on the diagram's syntax and semantics.

In the present work, we propose a dedicated formal syntactic and semantic definitions for the activity diagrams. Our main contribution consists of defining a dedicated language, namely *Activity Calculus* (AC), endowed with a formal operational semantics definition based on the Structural Operational Semantics (SOS) [13]. The dedicated language allows for mathematically expressing and analyzing the system's behavior captured by the activity diagrams. This formalization enables us to build a sound framework for the assessment of systems design expressed in SysML activity diagrams. More precisely, the defined formal semantics for a given SysML activity diagram can be input into a

978-0-7695-3602-6/09 $25.00 © 2009 IEEE
DOI 10.1109/ECBS.2009.25

IEEE computer society

Authorized licensed use limited to: CONCORDIA UNIVERSITY LIBRARIES. Downloaded on May 4, 2009 at 09:52 from IEEE Xplore. Restrictions apply.

probabilistic model checker in order to perform probabilistic verification [14]. One of the challenges faced when defining such a syntax is the wide expressiveness of this diagram regarding advanced control flow such as unstructured loops and non well-formed flows. The majority of previous initiatives express the syntax of activity diagrams as a tuple data structures. Very few of them provide a dedicated algebraic-like notation to express activity diagrams [7], [15], where only [7] aims at defining a formal semantics framework. Furthermore, the choice of an SOS-based approach was motivated by the capability to provide rigorous and intuitive operational understanding of the behavior captured by the diagram. To the best of our knowledge, there are no initiatives proposing such a formalism for SysML activity diagrams with a dedicated language and a structural operational semantics, independently of existing formalisms. It is worthwhile to mention also that we did not find any work on the formalization of SysML activity diagrams.

The rest of this paper is organized as follows. Section 2 reports existing initiatives on the formalization of the UML/SysML activity diagrams. Section 3 briefly describes a subset of the concrete syntax of the activity diagram and its corresponding informal semantics according to SysML. Section 4 details the language that we propose and the mapping of activity diagram constructs into expressions using our language. Therein, the corresponding operational semantics is explained. In order to illustrate the usefulness of the formal semantics, a SysML activity diagram case study is presented in Section 5. This is meant to show how the formal operational rules may uncover possible subtle errors in the behavior depicted by the design. Finally, Section 6 concludes this paper by summarizing the main contributions and discussing the foreseeable future work.

2. Related Work

Presently, to the best of our knowledge, there are no proposals on the formal semantics of SysML activity diagrams. Few initiatives are concerned with the V&V of SysML models [14], [16]. Jarraya et al. [14] present a mapping of time-annotated SysML activity diagrams into Discrete-Time Markov Chains (DTMC) for performance evaluation using a probabilistic model checker. Viehl et al. [16] considers time-annotated sequence and UML structured classes/SysML assemblies for performance analysis of System-On-a-Chip (SoC) systems based on simulation. Regarding the formalization of UML activity diagrams, some initiatives such as [4], [5], [17], [18] are within UML 1.x. Others propose a formal semantics for UML 2.0 activity diagrams using a mapping into an existing formalism with well-defined semantics [6]–[10], [19]. The related work can be divided into four distinct approaches: (1) Mapping activity diagrams into a process algebra,(2) mapping activity diagrams into Petri-nets, (3) graph transformation techniques, (4) mapping activity into Abstract State Machines (ASM).

The ASM formalism is proposed in [17], [19]. Böger et al. [17] consider UML 1.3 activity diagrams and define their semantics by mapping their elements into transition rules of a multi-agent ASM (an extensions of ASM with concurrency). Similarly, Sarstedt and Guttmann [19] propose a token flow semantics for a subset of UML 2.0 activity diagrams based on the asynchronous multi-agent ASM model. ASM formalism provides semantics close to the implementation level. It also excludes non well-formed control flows (every fork should be followed by a subsequent join node). However, our approach targets a higher level of abstraction so that limitations imposed by the implementation are avoided. Moreover, we do not restrict the designer to apply only well-formed control flows but we also support nested (and mixed) forks and joins. The approaches in [20], [21] deal with graph transformation techniques of UML 2.0 activity diagrams. Bisztray and Heckel [20] propose an approach that combines CSP process algebra and rule-based graph transformation technique. The mapping is based on the Triple Graph Grammars (TGGs) technique for graph transformations at the meta-model level. However, this approach is closely dependent on the semantic domain of CSP by considering only synchronous parallel composition. Hausmann [21] propose the specification of visual modeling languages semantics based on the Dynamic Meta Modeling (DMM), which is a combination of denotational meta modeling and operational graph transformation rules. However, this technique is quite complex and needs human intervention and understandability of a large set of rules.

Concerning process algebra, Rodrigues [4] considers the formalization of the UML 1.3 activity diagrams using Finite State Processes (FSP). Yang et al. [5] propose a formalization of a subset of UML 1.4 activity diagrams using the π-calculus. For the UML 2.0 activity diagrams, Scuglík [6] proposes CSP as a formal framework. Many activity diagram constructs are covered, however, some constructs such as fork/join and merge have no direct mapping into the CSP syntax. They are handled by a combination of some elements from the CSP domain. Tabuchi et al. [7] propose a stochastic performance analysis of UML 2.0 state machines and activity diagrams annotated with the UML Profile for Schedulability, Performance, and Time. This is done using stochastic process algebraic semantics based on IMC. However, none of these proposed mapping is intuitive, since in most of the cases there is no one-to-one correspondence between activity diagram and process algebra neither syntactically nor semantically. This may also result in the difficulty to refer back to the original activity diagram if one has its corresponding process algebra term.

Among approaches based on Petri net (PN) semantics, Lopez-Grao et al. [18] consider UML activity diagrams as a variant of the UML state machine and propose a mapping

Authorized licensed use limited to: CONCORDIA UNIVERSITY LIBRARIES. Downloaded on May 4, 2009 at 09:52 from IEEE Xplore. Restrictions apply.

into the Labeled Generalized Stochastic Petri Net (LGSPN). Störrle proposes PN-based semantics for UML 2.0 activity diagrams [8]–[10]. [8] handles control flow using a mapping into Procedural Petri Nets (PPN), which is an extension of PN for supporting calling subordinate activities or hierarchy and all kinds of control flow (well-formed or not) but neither data flow nor exception handling are supported. [9] examines exception handling and provides a mapping into an extension of PPN, which is the Exception Petri Nets (EPN). The semantics is denotational and built on top of the semantics of [8]. Recently, Störrle has addressed data flow formalization [10] using Colored Petri Net (CPN). Although the work of Störrle seems to cover the majority of UML 2.0 activity diagram features, some of them have to be investigated more such as streaming and expansion regions. Störrle and Hausmann [12] examined questions related to the appropriateness of the PN paradigm for expressing the UML 2.0 activity diagram semantics. Even though the UML standard claims that activity diagrams are redesigned using a Petri-like semantics, the mapping of some features such as exceptions, streaming, and traverse-to-completion is not so natural and different variants of PN are needed to cover all the features. Moreover, some other problems are hindering the progress of investigations in this direction, which include the absence of analysis tools and theoretical results for a unified formalism, if it exists, combining the domains of all PN variants [12].

Finally, we reviewed some orchestration languages since activity diagram is also used for coordination of behavior. Particularly, Orc [22] is a powerful programming language mainly designed for implementing concurrent programming patterns, workflow patterns, and computation/service orchestration. Although, one might find an intersection between the semantics of both Orc and activity diagrams, none of them can capture the full potential of the other. Moreover, they were designed at a different level of abstraction. Unlike Orc, our AC calculus is designed for different purposes and at different level of abstraction. Our primary purpose of designing AC is to systematically describe the dynamic semantics of activity diagrams, while being implementation independent, in order to validate the design against its requirements using probabilistic model checker.

3. SysML Activity Diagram

SysML has its roots in UML 2, however, it extends and adapts UML to better fit SE practices and methodologies. The activity diagram is the most affected diagram by these extensions. Among new features supported by SysML activity diagram we cite probabilistic behavior. In the following, we describe first a subset of the activity diagrams concrete syntax as specified in the standard followed by our understanding of its underpinning informal semantics.

The activity diagram basic constructs are activity nodes such as action, object, and control nodes as depicted in Fig. 1. Control nodes include fork, join, decision, merge, initial, activity final, and flow final. The initial node defines the start of an activity, whereas the activity final node indicates its completion. Unlike activity final node, flow final node illustrates the completion of a specific control flow inside the activity diagram. Decision and merge nodes are both represented by a diamond notation. Typically, a decision node has two or more outgoing activity edges, labeled with boolean guard conditions, and only one incoming edge, whereas a merge node has only one outgoing activity edge and two or more incoming edges. Furthermore, fork and join nodes are pictured using a bar shape. A fork node has a single incoming edge and many outgoing edges, and it is the reverse for join node. Action nodes and control nodes can be connected with directed control or/and object flow edges, denoting the direction where the control or the object is being passed. Finally, SysML enables the specification of probabilistic behaviors in the activity diagrams in two ways: On edges outgoing from decision nodes and as an extension to some output parameter sets (the set of outgoing edges that hold data output from an action node) [2]. In the present work, we only consider the probabilistic choices. Moreover, we assume a single initial node but this is not a restriction since we can replace all initial nodes by only one node connected to a fork node. Fig. 2 shows a SysML activity diagram for money withdrawal operation from an ATM.

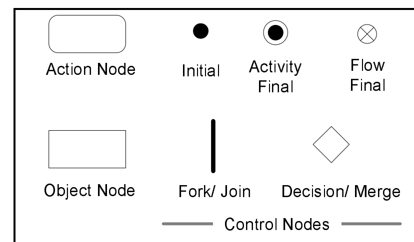

Figure 1. Activity Diagram Basic Constructs

With respect to the informal semantics as described in the standard, activity diagrams specify behaviors composed of a set of actions that are executed with a specific invocation order. The latter is imposed by control flows, optionally emphasizing input and output dependencies using data flows. Actions may be coordinated sequentially or concurrently. Furthermore, the diagram may involve synchronization and/or branching. These features are enabled using the predefined control nodes that support various forms of control routing. 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 two or more possible paths based on the evaluation of a guard condition (and/or a

Authorized licensed use limited to: CONCORDIA UNIVERSITY LIBRARIES. Downloaded on May 4, 2009 at 09:52 from IEEE Xplore. Restrictions apply.

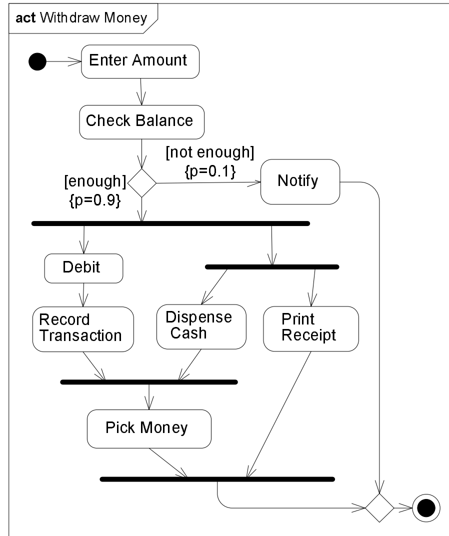

Figure 2. Activity Diagram of Money Withdrawal

probability distribution), a fork node indicates the beginning of multiple parallel flows. Moreover, a merge node is a point of convergence for different incoming flows without the need of synchronization, whereas a join node synchronizes and rejoins multiple parallel flows.

The semantics of activities is based on token flow [11]. From our understanding, we briefly describe the token movement in the activity diagram as follows. First, an initial token starts flowing from the initial node and moving from an action node to the next action(s) with respect to the foregoing set of control routing rules defined by the control nodes until reaching an activity final or a flow final node. In the case of parallel flows, the token is duplicated as many times as there are outgoing edges from the fork node. When tokens reach a join node, they merge into one token that flows downstream on the outgoing edge. The first token that reaches an activity final node stops all the other active flows in the activity diagram. Moreover, any token that reaches a flow final node ends only its corresponding control flow. Finally, when a token reaches a probabilistic decision node, the selection of the propagation flow is made probabilistically. In other words, probabilities express the likelihood that a token will traverse the corresponding edge (or equivalently that the guard is evaluated to true).

4. Syntax and Operational Semantics

In this section, we present the syntax and semantics of Activity Calculus (AC). To the best of our knowledge, this constitutes the first endeavor in defining a dedicated calculus for activity diagrams. The proposed syntax is algebraic-like. In order to improve readability, the syntactic elements are self-descriptive and close to the diagram constructs. Moreover, this is the first proposal for an SOS-based semantics for activity diagrams independently from any other existing language.

4.1. Syntax

From the structural perspective, activity diagram can be viewed as a directed graph with two types of nodes (action and control nodes) connected using directed edges. Alternatively, from the dynamic perspective, the activity diagram behavior amounts to a specifically ordered execution of the actions contained inside the diagram. This order depends on the propagation of the control locus (token) that initially starts from the initial node. When an action receives a token, it becomes active and starts executing. When its execution terminates, it delivers the token to its outgoing edges. During the execution, activity diagram structure remains unchanged, however, the position of the control token changes. Thus, the behavior depicted by the activity diagram (semantics) can be described using a set of progress rules that dictates the tokens movement through the diagram. In the rest of this paper, we will use the word marking (borrowed from the Petri net formalism) to specify the presence of control tokens.We assume that each activity node in the diagram (except initial) has a unique label. Let $\mathcal{L}$ be a collection of labels ranged over by l, l_0, l_l, $\cdots$. We write $l{:}\,N$ to denote an l-labeled activity node N, where N can be any node except initial. Labels serve different purposes. Mainly, a label l is used for uniquely referring an l-labeled activity node in order to model a flow connection to the already defined node. They are useful for connecting multiple incoming flows towards merge and join nodes. Activity calculus terms are derived based on a depth-first traversal of the corresponding activity diagrams. Thus, the mapping of activity diagrams into AC terms is achieved systematically. As a syntactic convention, each time a new merge (or join) node is met, the definition of the node and its newly assigned label are considered. If the node is encountered later in the traversal process, only its corresponding label is used. This convention is important to ensure well-formedness of the AC terms. The translation of the activity constructs into the AC syntax is illustrated in Fig. 4. The syntax of the AC language is defined using Backus-Naur-Form (BNF) notation illustrated in Fig. 3. A marked AC term, typically given by $\mathcal{B}$, corresponds to an active activity diagram with tokens (during its execution). An unmarked AC term, typically given by $\mathcal{A}$, corresponds to the diagram without tokens. The difference between the marked and unmarked expressions consists in the added "overbar" symbol for the marked terms (or subterm) denoting the presence and the location of the tokens. The idea of decorating the syntax was inspired by the work on Petri net algebra in [23]. However, we extended this concept in order to handle multiple tokens. This allows us to consider loops

Authorized licensed use limited to: CONCORDIA UNIVERSITY LIBRARIES. Downloaded on May 4, 2009 at 09:52 from IEEE Xplore. Restrictions apply.

$$
\begin{array}{lcl}
\mathcal{A} & ::= & \epsilon \\
 & | & \iota \rightarrowtail \mathcal{N} \\
 & & \\
\mathcal{N} & ::= & \epsilon \\
 & | & l\colon \otimes \\
 & | & l\colon \odot \\
 & | & l\colon Merge(\mathcal{N}) \\
 & | & l\colon x.Join(\mathcal{N}) \\
 & | & l\colon Fork(\mathcal{N},\, \mathcal{N}) \\
 & | & l\colon Decision_p(\langle g\rangle\ \mathcal{N},\, \langle\neg g\rangle\ \mathcal{N}) \\
 & | & l\colon Decision(\langle g\rangle\ \mathcal{N},\, \langle\neg g\rangle\ \mathcal{N}) \\
 & | & l\colon a \rightarrowtail \mathcal{N} \\
 & | & l
\end{array}
\qquad
\begin{array}{lcl}
\mathcal{B} & ::= & \overline{\mathcal{A}} \\
 & | & \iota \rightarrowtail \mathcal{M} \\
 & | & \overline{\iota} \rightarrowtail \mathcal{N} \\
\mathcal{M} & ::= & \mathcal{N} \\
 & | & l\colon Merge(\mathcal{M}) \\
 & | & l\colon x.Join(\mathcal{M}) \\
 & | & l\colon Fork(\mathcal{M},\, \mathcal{M}) \\
 & | & l\colon Decision_p(\langle g\rangle\ \mathcal{M},\, \langle\neg g\rangle\ \mathcal{M}) \\
 & | & l\colon Decision(\langle g\rangle\ \mathcal{M},\, \langle\neg g\rangle\ \mathcal{M}) \\
 & | & \overline{l\colon a}^{n} \rightarrowtail \mathcal{M} \\
 & | & \overline{\mathcal{M}}^{n}
\end{array}
$$

Figure 3. Unmarked Syntax (left) and Marked Syntax (right) of Activity Calculus

in activity diagrams and so multiple instances of actions. Thus, for example the expression $\overline{\mathcal{N}}^n$ denotes a node (or a flow of node) that is marked with n tokens such that $n \geq 0$. The definition of the term $\mathcal{B}$ is based on $\mathcal{A}$, since $\mathcal{B}$ represents all valid sub-terms with all possible positions of the overbar symbol on top of $\mathcal{A}$ subterms. $\mathcal{N}$ defines an unmarked subterm of $\mathcal{A}$ and $\mathcal{M}$ represents a marked subterm. An activity calculus term $\mathcal{A}$ is either ϵ, to denote an empty activity or $\iota \rightarrowtail \mathcal{N}$, where ι specifies the initial node and $\mathcal{N}$ can be any labeled activity node (or control flows of nodes). The symbol $\rightarrowtail$ is used to specify the control flow edge. Among the basic constructs of $\mathcal{N}$, we have:

- $l\colon \otimes$ (resp. $l\colon \odot$) specifies the flow final node (resp. the activity final node),
- $l\colon Merge(\mathcal{N})$ (resp. $l\colon x.Join(\mathcal{N})$) represents the definition of the merge (resp. join) node. This notation is used only when the corresponding node is firstly encountered during the depth-first traversal of the activity diagram. The parameter $\mathcal{N}$ inside the merge (resp. join) refers to the subsequent destination nodes (or flow) connected to the outgoing edge of the merge (resp. join) node. With respect to the join node, the entity x represents an integer that specifies the number of incoming edges to this specific join node.
- $l : Fork(\mathcal{N}_1,\, \mathcal{N}_2)$ is the construct referring to the fork node. The parameters $\mathcal{N}_1$ and $\mathcal{N}_2$ represent the subterms corresponding to the destination of the outgoing edges of the fork node (i.e. the flows split in parallel).
- $l\colon Decision_p(\langle g\rangle \mathcal{N}_1, \langle\neg g\rangle \mathcal{N}_2)$ (resp. $l\colon Decision(\langle g\rangle \mathcal{N}_1, \langle\neg g\rangle \mathcal{N}_2)$) specifies the probabilistic (resp. non probabilistic) decision node. It denotes a probabilistic (resp. non deterministic) choice between alternative flows $\mathcal{N}_1$ and $\mathcal{N}_2$. For the probabilistic case, the subterm $\mathcal{N}_1$ is selected with a probability p whereas, $\mathcal{N}_2$ is selected with probability $1 - p$.
- $l\colon a \rightarrowtail \mathcal{N}$ is the construct representing the prefix operator: The labeled action $l : a$ is connected to $\mathcal{N}$ using a control flow edge.
- l is a reference to a node labeled with l.

	AD Constructs	AC Syntax
$\mathcal{A}$	● → $\mathcal{N}$	$\iota \rightarrowtail \mathcal{N}$
$\mathcal{N}$	◉	$l\colon \odot$
	$\otimes$	$l\colon \otimes$
	a → $\mathcal{N}$	$l\colon a \rightarrowtail \mathcal{N}$
	[¬g] $\mathcal{N}_2$ [g] $\mathcal{N}_1$	$l\colon Decision(\langle g\rangle\, \mathcal{N}_1,\, \langle\neg g\rangle\, \mathcal{N}_2)$
	[¬g] {1-p} $\mathcal{N}_2$ [g] {p} $\mathcal{N}_1$	$l\colon Decision_p(\langle g\rangle\, \mathcal{N},\, \langle\neg g\rangle\, \mathcal{N})$
	◇ → $\mathcal{N}$	$l\colon Merge(\mathcal{N})$ or l
	$\mathcal{N}_1$ $\mathcal{N}_2$	$l\colon Fork(\mathcal{N}_1,\, \mathcal{N}_2)$
	$\mathcal{N}$	$l\colon x.Join(\mathcal{N})$ or l (x is the number of incoming edges)

Figure 4. Mapping into AC Syntax

For instance, the SysML activity diagram of Fig. 2 can be expressed using the unmarked term $\mathcal{A}_{\texttt{withdraw}}$, such that:

$$
\begin{aligned}
\mathcal{A}_{\texttt{withdraw}} &= \iota \rightarrowtail l_1\colon \texttt{Enter} \rightarrowtail l_2\colon \texttt{Check} \rightarrowtail \mathcal{N}_1 \\
\mathcal{N}_1 &= l_3\colon\ \texttt{Decision}_{0.1}(\langle\texttt{notenough}\rangle\, \mathcal{N}_2, \\
&\qquad \langle\texttt{enough}\rangle\, \mathcal{N}_3) \\
\mathcal{N}_2 &= l_4\colon \texttt{Notify} \rightarrowtail l_5\colon \texttt{Merge}(l_6\colon \odot) \\
\mathcal{N}_3 &= l_7\colon \texttt{Fork}(\mathcal{N}_4,\, l_{13}\colon \texttt{Fork}(l_{14}\colon \texttt{Disp} \rightarrowtail l_{10}, \\
&\qquad l_{15}\colon \texttt{Print} \rightarrowtail l_{12})) \\
\mathcal{N}_4 &= l_8\colon\ \texttt{Debit} \rightarrowtail l_9\colon\ \texttt{Record} \rightarrowtail l_{10}\colon\ 2.\texttt{Join}(\\
&\qquad l_{11}\colon \texttt{Pick} \rightarrowtail l_{12}\colon 2.\texttt{Join}(l_5))
\end{aligned}
$$

The corresponding marked terms, which denote the different positions of the token(s), are derived using the operational rules that will be presented in the sequel.

Authorized licensed use limited to: CONCORDIA UNIVERSITY LIBRARIES. Downloaded on May 4, 2009 at 09:52 from IEEE Xplore. Restrictions apply.

4.2. Operational Semantics

In this section, we present the operational semantics of AC terms in the SOS style. The latter is a well-established approach that provides a framework to give an operational semantics to programming and specification languages [13]. It is also considerably applied in the study of the semantics of concurrent processes. Defining such semantics (small-step semantics) for the AC terms consists in defining a set of axioms and derivation rules that are used to describe the behavior evolution of the studied diagram. These axioms and rules specify the possible transitions that a marked AC term can make during the progress of tokens. Since in some cases we might have more than one token present in the activity, the selection of the one making the progress is performed non-deterministically. The operational semantics is given by a Probabilistic Transition System (PTS) as presented in Definition 1. The general form of a transition is $\mathcal{B} \xrightarrow{\alpha}_p \mathcal{B}'$ or $\mathcal{B} \xrightarrow{\alpha}_p \mathcal{A}$, such that $\mathcal{B}$ and $\mathcal{B}'$ are marked activity calculus terms, $\mathcal{A}$ is unmarked activity calculus term, $\alpha \in \Sigma \cup \{o\}$, the set of actions ranged over by a, a_1, $\cdots$, b, o denotes the empty action, and $p, q \in [0,1]$ are probabilities of transitions occurrences. This transition relation shows the marking evolution and means that a marked term $\mathcal{B}$ can be transformed into another marked term $\mathcal{B}'$ or to an unmarked term $\mathcal{A}$ by executing α with a probability p. If a marked term is transformed into an unmarked term, the transition denotes the loss of the marking. This is the case where a flow final or an activity final node are reached. For simplicity, we omit the label o on the transition relation, if no action is executed, i.e. $\mathcal{B} \longrightarrow_p \mathcal{B}'$ or $\mathcal{B} \longrightarrow_p \mathcal{A}$. The transition relation is defined from Fig. 5 to Fig. 12.

Definition 1. *The probabilistic transition system of the Activity Calculus term $\mathcal{B}$ is the tuple $\mathcal{T}$=($\mathcal{B}$, S, Lab, $\xrightarrow{\alpha}_p$) where:*

- *$\mathcal{B}$ is the initial state,*
- *S is the set of states, ranged over by s, where s is an AC term reachable from $\mathcal{B}$, i.e. if we denote by $\xrightarrow{\alpha}^*_p$ the reflexive transitive closure of $\xrightarrow{\alpha}_p$, S=$\{\mathcal{B}' \mid \mathcal{B} \xrightarrow{\alpha}^*_p \mathcal{B}'\} \cup \{\mathcal{A} \mid \mathcal{B} \xrightarrow{\alpha}^*_p \mathcal{A}\}$,*
- *$Lab \subseteq \Sigma \cup \{o\} \times [0,1]$ is a set of pairs composed of actions (or the empty action) and their corresponding probability,*
- *$\xrightarrow{\alpha}_p$ is the probabilistic transition relation over $S \times Lab \times S$ where $(\alpha, p) \in Lab$. It is the least relation satisfying the AC operational semantics rules.* □

Let e be a marked term and f, f_1, $\cdots$, f_n specify marked (or unmarked) terms. f is a subterm (or a subexpression) of e, denoted by $e[f]$, if f is a valid activity calculus term occurring once in the definition of e. We also use the notation $e[f\{x\}]$ to denote that f occurs exactly x times in the expression e. For simplification $e[f\{1\}]$ = $e[f]$. We may generalize this notation to more than one subterm, i.e. $e[f_1, f_2, \cdots, f_n]$. For instance, given a marked term $\mathcal{B}= \iota \rightarrowtail \overline{l_1 : a_1} \rightarrowtail l_2 : a_2 \rightarrowtail l_3 : \odot$. We write $\mathcal{B}[\overline{l_1 : a_1}]$ to specify that $\overline{l_1 : a_1}$ is a subterm of $\mathcal{B}$. Furthermore, we use the notation $|\mathcal{B}|$ to denotes the unmarked activity calculus term obtained by removing overbars from the marked term $\mathcal{B}$. In the sequel, we present the AC operational semantics.

INIT-1	$\overline{\iota \rightarrowtail \mathcal{N}} \longrightarrow_1 \overline{\iota} \rightarrowtail \mathcal{N}$
INIT-2	$\overline{\iota} \rightarrowtail \mathcal{N} \longrightarrow_1 \iota \rightarrowtail \overline{\mathcal{N}}$
INIT-3	$\dfrac{\mathcal{M} \xrightarrow{\alpha}_q \mathcal{M}'}{\iota \rightarrowtail \mathcal{M} \xrightarrow{\alpha}_q \iota \rightarrowtail \mathcal{M}'}$

Figure 5. Rules for Initial

ACT-1	$\overline{\overline{l:a}^k \rightarrowtail \mathcal{M}}^n \longrightarrow_1 \overline{\overline{l:a}^{k+1} \rightarrowtail \mathcal{M}}^{n-1} \quad \forall n > 0$
ACT-2	$\overline{\overline{l:a}^k \rightarrowtail \mathcal{M}}^n \xrightarrow{a}_1 \overline{\overline{l:a}^{k-1} \rightarrowtail \overline{\mathcal{M}}}^n \quad \forall k > 0$
ACT-3	$\dfrac{\mathcal{M} \xrightarrow{\alpha}_q \mathcal{M}'}{\overline{\overline{l:a}^k \rightarrowtail \mathcal{M}}^n \xrightarrow{\alpha}_q \overline{\overline{l:a}^k \rightarrowtail \mathcal{M}'}^n}$

Figure 6. Rules for Action Prefixing

4.2.1. Rules for Initial.

The first set of rules in Fig. 5 refers to the transitions related to the term $\iota \rightarrowtail \mathcal{N}$. Rule INIT-1 means that the expression $\overline{\iota \rightarrowtail \mathcal{N}}$ can do a transition to $\overline{\iota} \rightarrowtail \mathcal{N}$ with no observable action and with a probability $q=1$. This complies with the specification, which states that an initially activated activity diagram is equivalent to place a control token on the initial node. Rule INIT-2 means that if ι is marked, the marking propagates to the rest of the term, i.e. $\mathcal{N}$, with no observable action and with a probability $q=1$. Rule INIT-3 allows the marking to evolve from $\iota \rightarrowtail \mathcal{M}$ to $\iota \rightarrowtail \mathcal{M}'$ with probability q, by executing the action α if the sub-expression $\mathcal{M}$ can evolve to $\mathcal{M}'$, by the same transition.

4.2.2. Rules for Action Prefixing.

The second set of rules in Fig. 6 concerns action prefixing. These rules illustrate the possible progress of the tokens in the expression $\overline{\overline{l:a}^k \rightarrowtail \mathcal{M}}^n$. Rule ACT-1 concerns the progress of one of the tokens marking the whole expression. As a result of this rule, one token is deleted from the n tokens and it is relayed to the first subterm $\overline{l:a}^k$, thus adding one token to the k tokens. Rule ACT-2 specifies the progress of a token from the subterm $\overline{l:a}^k$ to $\mathcal{M}$. Finally, Rule ACT-3 allows the marking to evolve from $\overline{\overline{l:a}^k \rightarrowtail \mathcal{M}}^n$ to $\overline{\overline{l:a}^k \rightarrowtail \mathcal{M}'}^n$ by executing the action α and with probability q, if $\mathcal{M}$ can evolve to $\mathcal{M}'$.

Authorized licensed use limited to: CONCORDIA UNIVERSITY LIBRARIES. Downloaded on May 4, 2009 at 09:52 from IEEE Xplore. Restrictions apply.

PDEC-1	$\overline{l\colon Decision_p(\langle g\rangle\mathcal{M}_1,\langle\neg g\rangle\mathcal{M}_2)}^n \longrightarrow_p \overline{l\colon Decision_p(\langle tt\rangle\overline{\mathcal{M}_1},\langle ff\rangle\mathcal{M}_2)}^{n-1}\quad \forall n>0$
PDEC-2	$\overline{l\colon Decision_p(\langle g\rangle\mathcal{M}_1,\langle\neg g\rangle\mathcal{M}_2)}^n \longrightarrow_{1-p} \overline{l\colon Decision_p(\langle ff\rangle\mathcal{M}_1,\langle tt\rangle\overline{\mathcal{M}_2})}^{n-1}\quad \forall n>0$
PDEC-3	$\dfrac{\mathcal{M}_1 \xrightarrow{\alpha}_q \mathcal{M}_1'}{\begin{array}{l}\overline{l\colon Decision_p(\langle g\rangle\mathcal{M}_1,\langle\neg g\rangle\mathcal{M}_2)}^n \xrightarrow{\alpha}_q \overline{l\colon Decision_p(\langle g\rangle\mathcal{M}_1',\langle\neg g\rangle\mathcal{M}_2)}^n\\ \overline{l\colon Decision_p(\langle g\rangle\mathcal{M}_2,\langle\neg g\rangle\mathcal{M}_1)}^n \xrightarrow{\alpha}_q \overline{l\colon Decision_p(\langle g\rangle\mathcal{M}_2,\langle\neg g\rangle\mathcal{M}_1')}^n\end{array}}$

Figure 9. Rules for Probabilistic Decision

DEC-1	$\overline{l\colon Decision(\langle g\rangle\mathcal{M}_1,\langle\neg g\rangle\mathcal{M}_2)}^n \longrightarrow_1 \overline{l\colon Decision(\langle tt\rangle\overline{\mathcal{M}_1},\langle ff\rangle\mathcal{M}_2)}^{n-1}\quad \forall n>0$
DEC-2	$\overline{l\colon Decision(\langle g\rangle\mathcal{M}_1,\langle\neg g\rangle\mathcal{M}_2)}^n \longrightarrow_1 \overline{l\colon Decision(\langle ff\rangle\mathcal{M}_1,\langle tt\rangle\overline{\mathcal{M}_2})}^{n-1}\quad \forall n>0$
DEC-3	$\dfrac{\mathcal{M}_1 \xrightarrow{\alpha}_q \mathcal{M}_1'}{\begin{array}{l}\overline{l\colon Decision(\langle g\rangle\mathcal{M}_1,\langle\neg g\rangle\mathcal{M}_2)}^n \xrightarrow{\alpha}_q \overline{l\colon Decision(\langle g\rangle\mathcal{M}_1',\langle\neg g\rangle\mathcal{M}_2)}^n\\ \overline{l\colon Decision(\langle g\rangle\mathcal{M}_2,\langle\neg g\rangle\mathcal{M}_1)}^n \xrightarrow{\alpha}_q \overline{l\colon Decision(\langle g\rangle\mathcal{M}_2,\langle\neg g\rangle\mathcal{M}_1')}^n\end{array}}$

Figure 10. Rules for Non-Deterministic Decision

FLOW-FINAL	$\overline{l\colon\otimes}^n \longrightarrow_1 \overline{l\colon\otimes}^{n-1}\ \forall n>0$
FINAL	$\mathcal{B}[\overline{l\colon\odot}] \longrightarrow_1 \lvert\mathcal{B}\rvert$

Figure 7. Rules for Finals

FORK-1	$\overline{l\colon Fork(\mathcal{M}_1,\mathcal{M}_2)}^n \longrightarrow_1 \overline{l\colon Fork(\overline{\mathcal{M}_1},\overline{\mathcal{M}_2})}^{n-1}\quad \forall n>0$
FORK-2	$\dfrac{\mathcal{M}_1 \xrightarrow{\alpha}_q \mathcal{M}_1'}{\begin{array}{l}\overline{l\colon Fork(\mathcal{M}_1,\mathcal{M}_2)}^n \xrightarrow{\alpha}_q \overline{l\colon Fork(\mathcal{M}_1',\mathcal{M}_2)}^n\\ \overline{l\colon Fork(\mathcal{M}_2,\mathcal{M}_1)}^n \xrightarrow{\alpha}_q \overline{l\colon Fork(\mathcal{M}_2,\mathcal{M}_1')}^n\end{array}}$

Figure 8. Rules for Fork

4.2.3. Rules for Finals. The rules for flow final and activity final are illustrated in Fig. 7. The axiom FLOW-FINAL shows that $\overline{l\colon\otimes}^n$ can do a transition with probability 1 and no action to $\overline{l\colon\otimes}^{n-1}$, which represents the deletion of one token. With respect to activity final, once marked (one token is enough), it imposes the abrupt termination of all the other normal flows in the activity. This is described using rule FINAL stating that if $\overline{l\colon\odot}$ is a subterm of a marked term $\mathcal{B}$, the latter can do a transition with probability $q=1$ and no action resulting in the deletion of all overbars from $\mathcal{B}$.

4.2.4. Rules for Fork. The rules for fork are listed in Fig. 8. The axiom FORK-1 shows the propagation of the tokens to the subterms of the fork if the whole fork expression is marked. Rules FORK-2 illustrates two symmetric rules showing the evolution of the marking in the subterms of the fork expression. According to the activity diagram specification, the fork node generates unrestricted parallelism. Thus, the marking evolves asynchronously according to an interleaving semantics on both left and right subterms.

4.2.5. Rules for Merge. Rules for merge are presented in Fig. 11. Rule MERG-1 states that if $\overline{l}^k$ is a subterm of $\mathcal{B}$ and l corresponds to the merge node $\overline{l\colon Merge(\mathcal{M})}^n$, which is also a subterm of $\mathcal{B}$, then there is an unlabeled transition that results in the expression $\mathcal{B}$ where $\overline{l}^k$ is replaced by $\overline{l}^{k-1}$ and $\overline{l\colon Merge(\mathcal{M})}^n$ replaced by $\overline{l\colon Merge(\mathcal{M})}^{n+1}$ $\forall k\geq 1$. Rule MERG-2 states that the marking on top of the merge expression evolves with probability 1 and no action to the subterm of the merge. Rule MERG-3 allows the marking to evolve in the expression $\overline{l\colon Merge(\mathcal{M})}^n$ if there is a possible transition such that $\mathcal{M} \xrightarrow{\alpha}_q \mathcal{M}'$.

4.2.6. Rules for Decision. The next set of rules concerns the probabilistic decision as illustrated in Fig. 9 and the rules for the non-deterministic decision as illustrated in Fig. 10. Rules PDEC-1 and PDEC-2 are used to propagate the marking through the decision, having the marking on the top of it. Since the latter represents a probabilistic choice (with a guard condition), the marking will propagate either to the first branch with probability p (PDEC-1) or to the second branch with probability $1-p$ (PDEC-2). This meets the standard description of the probabilities on decision nodes since it illustrates the likelihood of the token traversing one of the branches. Rule PDEC-3 groups two symmetric cases that are related to the mark-

Authorized licensed use limited to: CONCORDIA UNIVERSITY LIBRARIES. Downloaded on May 4, 2009 at 09:52 from IEEE Xplore. Restrictions apply.

JOIN-1	$\mathcal{B}[\overline{l\colon x.Join(\mathcal{M})}^n, \overline{l}^{k_x}\{x-1\}] \longrightarrow_1 \mathcal{B}[l\colon x.Join(\overline{\mathcal{M}}), l\{x-1\}]\ x>1, n\geq 1, k_x\geq 1$
JOIN-2	$\overline{l\colon 1.Join(\mathcal{M})}^n \longrightarrow_1 \overline{l\colon 1.Join(\overline{\mathcal{M}})}^{n-1}\ \ n\geq 1$
JOIN-3	$\dfrac{\mathcal{M} \xrightarrow{\alpha}_q \mathcal{M}'}{\overline{l\colon x.Join(\mathcal{M})}^n \xrightarrow{\alpha}_q \overline{l\colon x.Join(\mathcal{M}')}^n}$

Figure 12. Rules for Join

MERG-1

$$\mathcal{B}[\overline{l\colon Merge(\mathcal{M})}^n, \overline{l}^k] \longrightarrow_1 \mathcal{B}[\overline{l\colon Merge(\mathcal{M})}^{n+1},\ \overline{l}^{k-1}]\ \forall k\geq 1$$

MERG-2

$$\overline{l\colon Merge(\mathcal{M})}^n \longrightarrow_1 \overline{l\colon Merge(\overline{\mathcal{M}})}^{n-1}\ \ \forall n\geq 1$$

MERG-3

$$\frac{\mathcal{M} \xrightarrow{\alpha}_q \mathcal{M}'}{\overline{l\colon Merge(\mathcal{M})}^n \xrightarrow{\alpha}_q \overline{l\colon Merge(\mathcal{M}')}^n}$$

Figure 11. Rules for Merge

ing evolution through the decision subterms. If a possible transition $\mathcal{M}_1 \xrightarrow{\alpha}_q \mathcal{M}_1'$ exists and $\mathcal{M}_1$ is a subexpression of $\overline{l\colon Decision_p(\langle g\rangle\mathcal{M}_1, \langle\neg g\rangle\mathcal{M}_2))}^n$, then we can deduce the transition $\overline{l\colon Decision_p(\langle g\rangle\mathcal{M}_1, \langle\neg g\rangle\mathcal{M}_2)}^n \xrightarrow{\alpha}_q \overline{l\colon Decision_p(\langle g\rangle\mathcal{M}_1', \langle\neg g\rangle\mathcal{M}_2)}^n$. For the non-probabilistic decision the rules are almost the same as for the probabilistic decision. The only difference is that the first two axioms define non-deterministic transitions.

4.2.7. Rules for Join. Rules for join are presented in Fig. 12. Rule JOIN-1 and Rule JOIN-2 describe the propagation of a token on the top of the join definition expression, $\overline{l\colon x.Join(\mathcal{M})}^n$ and the referencing labels. Unlike the merge node, the join traversal requires all references to itself to be marked if the node has 2 (or more) incoming flows. This is stated in the UML standard document under the "join specification" requirement. More precisely, all the subterms l corresponding to a given join node in the AC term, including the definition of join itself, have to be marked. The number of occurrence of $\overline{l}$ is known and it correspond to the value of $x-1$. If so, only one control token propagates to the subsequent subterm $\mathcal{M}$ with probability $q=1$. Moreover, according to the standard, if we have more than one token on the same incoming edge they are all combined into one. Rule JOIN-2 corresponds to the special case where $x=1$. There is no restriction in the standard on the use of a join node with a single incoming edge even though this is qualified as not useful. Rule JOIN-3 shows the possible evolution of $\overline{l\colon x.Join(\mathcal{M})}^n$ to $\overline{l\colon x.Join(\mathcal{M}')}^n$, if $\mathcal{M} \xrightarrow{\alpha}_q \mathcal{M}'$.

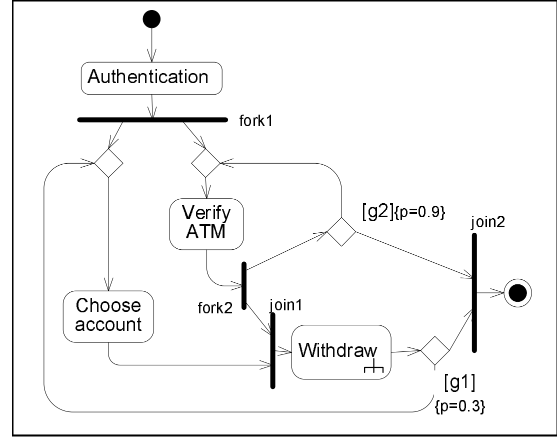

Figure 13. Activity Diagram Example

5. Case Study

In the sequel we present a SysML activity diagram case study used in order to demonstrate the benefit and usefulness of the proposed formal semantics. Apart from ascribing a rigorous meaning to the informally specified diagrams, formal semantics provides us with an effective technique to uncover design errors that could be missed by intuitive inspection. Furthermore, deriving the formal semantics of activity diagrams allows the application of model transformation and model checking. Practically, the informal description of the behavior captured by the activity diagrams does not enable the automation of the validation process. There is a real need to describe this behavior in a mathematically rigorous way. Thus, our formal framework allows the automation of the validation using existing techniques such as probabilistic model checking. Moreover, it allows reasoning about potential relations between activity diagrams from the behavioral perspective and deriving related mathematical proofs.

The SysML activity diagram illustrated in Fig. 13 depicts a hypothetical design of the behavior corresponding to banking operations on an Automated Teller Machine (ATM). The actions therein can be refined using structured activity nodes in order to expand their internal behavior. For instance, the node labeled `Withdraw` is actually a

Authorized licensed use limited to: CONCORDIA UNIVERSITY LIBRARIES. Downloaded on May 4, 2009 at 09:52 from IEEE Xplore. Restrictions apply.

structured node that calls the activity diagram pictured in Fig. 2. The operational semantics defined earlier allows us performing a compositional assessment of the design. First, the detailed activities are abstracted away and the global behavior is validated, then the refined behavior is assessed. The compositionality and abstraction features allow handling real-world systems without compromising the validation process. For instance, we consider the activity diagram of Fig. 13 and assume `Withdraw` to be an atomic action d. Moreover, considering the actions a, b, and c as the abbreviations of the actions `Authentication`, `Verify ATM`, and `Choose account` respectively, the corresponding unmarked term $\mathcal{A}_1$, is as follows:

$\mathcal{A}_1 = \iota \rightarrowtail l_1 \colon a \rightarrowtail l_2 \colon \texttt{Fork}(\mathcal{N}_1, l_{12})$
$\mathcal{N}_1 = l_3 \colon \texttt{Merge}(l_4 \colon b \rightarrowtail l_5 \colon \texttt{Fork}(\mathcal{N}_2, \mathcal{N}_3))$
$\mathcal{N}_2 = l_6 \colon \texttt{Decision}_{0.9}(\langle g2 \rangle\, l_3, \langle \neg g2 \rangle\, l_7 \colon 2.\texttt{Join}(l_8 \colon \odot))$
$\mathcal{N}_3 = l_9 \colon 2.\texttt{Join}(l_{10} \colon d \rightarrowtail \mathcal{N}_4)$
$\mathcal{N}_4 = l_{11} \colon \texttt{Decision}_{0.3}(\langle g1 \rangle\, \mathcal{N}_5, \langle \neg g1 \rangle\, l_7)$
$\mathcal{N}_5 = l_{12} \colon \texttt{Merge}(l_{13} \colon c \rightarrowtail l_9)$

The guard $g1$ denotes the possibility of triggering a new operation if evaluated to $true$ and guard $g2$ denotes the result of evaluating the status of the connection. Applying the operational rules on the marked $\overline{\mathcal{A}_1}$, we can derive a run that leads to a deadlock, which means that we reached a configuration where the expression is marked but no progress can be made (no operational rule can be applied). This derivation may reveal a design error in the activity diagram, which is not obvious using only inspection. Even though one may suspect the `join2` to cause the deadlock due to the presence of a prior decision node, the deadlock actually occurs in anther node (i.e. node `join1`). More precisely, the run consists in the execution of action c twice (because the guard $g1$ is true) and the action b only once ($g2$ evaluated to false). The deadlocked configuration reached by the derivation run consists in the following marked subterms:

$\mathcal{M}_2 = l_6 \colon \texttt{Decision}_{0.9}(\langle g2 \rangle\, l_3, \langle \neg g2 \rangle\, \overline{l_7 \colon 2.\texttt{Join}(l_8 \colon \odot)})$
$\mathcal{M}_5 = l_{12} \colon \texttt{Merge}(l_{13} \colon c \rightarrowtail \overline{l_9})$

The detailed derivation run is provided in Appendix A. In order to proceed to the validation of functional and non-functional requirements, the semantic model (i.e. PTS) has to be entirely derived and then formatted in order to be subjected to probabilistic model checking. Our semantic model presents probabilistic behavior and non-determinism in addition to concurrency, which is the case of Markov Decision Processes (MDP). The latter is supported by various probabilistic model checkers including the Probabilistic Symbolic Model Checker PRISM [24]. Properties to be checked are expressed using the Probabilistic Computation Tree Logic (PCTL) [24]. There are two possible ways to apply probabilistic model checking: either encoding the PTS using the model checker input language or explicitly input the probabilistic transition matrix with the set of states. Due to lack of space, we describe briefly hereafter the second alternative.

Having all reachable states of the semantic model representing all the reachable marking from $\overline{\mathcal{A}_1}$ and the probabilistic transition matrix resulting from the set of probabilistic transitions between states, we have to first process the information into the appropriate accepted file format and then to input both into the model checker. We also input the list of properties to be validated on the model. Finally, properties such as deadlock and reliability can be verified on the model. For instance, the property (1) specifies the eventuality of reaching a deadlock state (with probability $P > 0$) from any configuration starting at the initial state can be expressed as follows:

$$\text{“}init'' \Rightarrow \ \texttt{P} \ > 0 \ [\ \texttt{F} \ \text{“}\texttt{deadlock}'' \ ] \qquad (1)$$

Using PRISM model checker, this property returns true, which confirms our previous finding about the presence of a deadlock configuration.

6. Conclusion

This paper proposes a formal syntax and semantics for SysML activity diagrams. To the best of our knowledge, this is the first paper that explores this topic. Our main contributions consist in defining a formal dedicated language, namely Activity Calculus (AC), endowed with a formal operational semantics. On the one hand, the syntactic elements of the language are self-descriptive in order to improve its readability. Moreover, it is possible to recover the diagrams from their AC expressions. On the other hand, the semantics provides a rigorous, compositional, and intuitive operational understanding of the behavior captured by the diagram. Furthermore, our approach supports advanced control flows such as unstructured loops, concurrent control flows, and multiple instances of actions. In addition, it handles non well-formed control flows, which allows the use of mixed and nested forks and joins. It is important to notice that although non well-formed activity diagrams are allowed by the standard, many reviewed related work do not support them. Moreover, ascribing a meaning to SysML activity diagram enables the rigorous analysis of the design and the uncovering of design errors early in the development process.

As future work, we intend to build on top of the present formalism and to elaborate more on the probabilistic model checking of the SysML activity diagrams. Moreover, we plan to design and implement a practical framework that allows for the automatic derivation of the semantic models and mapping them into the input language accepted by the selected probabilistic model checker.

Authorized licensed use limited to: CONCORDIA UNIVERSITY LIBRARIES. Downloaded on May 4, 2009 at 09:52 from IEEE Xplore. Restrictions apply.

References

[1] INCOSE Technical Board, "INCOSE Systems Engineering Handbook - A "What To" Guide for all Systems Engineering Practitioners," INCOSE, Technical Report TP-2003-016-02, June 2004.

[2] Object Management Group, OMG Systems Modeling Language (OMG SysML) Available Specification 1.0, September 2007.

[3] C. Bock, "UML 2 Activity Model Support for Systems Engineering Functional Flow Diagrams," Journal of the International Council on Systems Engineering, vol. 6, no. 4, pp. 123–137, October 2003.

[4] R. W. S. Rodrigues, "Formalising UML Activity Diagrams Using Finite State Processes," Online Proceedings of UML 2000 Workshop on Dynamic Behaviour in UML Models: Semantic Questions, 2000.

[5] D. Yang and S. sheng Zhang, "Using π-Calculus to Formalize UML Activity Diagram for Business Process Modeling," in the 10th International Conference and Workshop on the Engineering of Computer Based Systems (ECBS), 2003, pp. 47–54.

[6] F. Scuglík, "Relation Between UML 2 Activity Diagrams and CSP Algebra," WSEAS Transactions on Computers, vol. 4, no. 10, pp. 1234–1240, 2005.

[7] N. Tabuchi, N. Sato, and H. Nakamura, "Model-Driven Performance Analysis of UML Design Models Based on Stochastic Process Algebra," in Proceedings of the European Conference on Model Driven Architecture - Foundations and Applications (ECMDA-FA), ser. Lecture Notes in Computer Science, A. Hartman and D. Kreische, Eds., vol. 3748. Springer, 2005, pp. 41–58.

[8] H. Störrle, " Semantics of Control-Flow in UML 2.0 Activities," in IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC). Rome, Italy: IEEE Computer Society, 2004, pp. 235–242.

[9] H. Storrle, "Semantics of Exceptions in UML 2.0 Activities," Ludwig-Maximilians-Universität München, Institut für Informatik, Tech. Rep. 0403, 2004.

[10] H. Störrle, "Semantics and Verification of Data Flow in UML 2.0 Activities," Electronic Notes in Theoretical Computer Science, vol. 127, no. 4, pp. 35–52, 2005.

[11] Object Management Group, OMG Unified Modeling Language: Superstructure 2.1, April 2006.

[12] H. Störrle and J. H. Hausmann, "Towards a Formal Semantics of UML 2.0 Activities," in Software Engineering, ser. LNI, P. Liggesmeyer, K. Pohl, and M. Goedicke, Eds., vol. 64. GI, 2005, pp. 117–128.

[13] G. D. Plotkin, "A Structural Approach to Operational Semantics," University of Aarhus, Technical Report DAIMI FN-19, 1981.

[14] Y. Jarraya, A. Soeanu, M. Debbabi, and F. Hassaïne, "Automatic Verification and Performance Analysis of Time-Constrained SysML Activity Diagrams," in the Proceedings of the 14th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems(ECBS'07), 26-29 March, Tucson, AZ, U.S.A. IEEE Computer Society, 2007, pp. 515–522.

[15] D. Flater, P. A. Martin, and M. L. Crane, "Rendering UML Activity Diagrams as Human-Readable Text," National Institute of Standards and Technology (NIST), Technical Report NISTIR 7469, November 2007.

[16] A. Viehl, T. Schänwald, O. Bringmann, and W. Rosenstiel, "Formal Performance Analysis and Simulation of UML/SysML Models for ESL Design," in the Proceedings of the Conference on Design, Automation and Test in Europe (DATE). European Design and Automation Assoc., 2006, pp. 242–247.

[17] E. Börger, A. Cavarra, and E. Riccobene, "An ASM Semantics for UML Activity Diagrams," in the Proceedings of the 8th International Conference on Algebraic Methodology and Software Technology (AMAST). London, UK: Springer-Verlag, 2000, pp. 293–308.

[18] J. P. López-Grao, J. Merseguer, and J. Campos, "From UML Activity Diagrams to Stochastic Petri Nets: Application to Software Performance Engineering," ACM SIGSOFT Software Engineering Notes, vol. 29, no. 1, pp. 25–36, 2004.

[19] S. Sarstedt and W. Guttmann, "An ASM Semantics of Token Flow in UML 2 Activity Diagrams," in Ershov Memorial Conference, ser. Lecture Notes in Computer Science, I. Virbitskaite and A. Voronkov, Eds., vol. 4378. Springer, 2006, pp. 349–362.

[20] B. Dénes and R. Heckel, "Rule-Level Verification of Business Process Transformations using CSP," in Proceedings of the Sixth International Workshop on Graph Transformation and Visual Modeling Techniques (GT-VMT), K. Ehrig and H. Giese, Eds., vol. 6, United Kingdom, May 2007, pp. 13–27.

[21] J. H. Hausmann, "Dynamic Meta Modeling: A Semantics Description Technique for Visual Modeling Languages," Ph.D. dissertation, University Paderborn, 2005.

[22] J. Misra and W. R. Cook, "Computation Orchestration," Software and Systems Modeling, vol. 6, no. 1, pp. 83–110, 2007.

[23] E. Best, R. Devillers, and M. Koutny, Petri Net Algebra. New York, NY, USA: Springer-Verlag New York, Inc., 2001.

[24] M. Kwiatkowska, G. Norman, and D. Parker, "Quantitative Analysis with the Probabilistic Model Checker PRISM," Electronic Notes in Theoretical Computer Science, vol. 153, no. 2, pp. 5–31, 2005.

Authorized licensed use limited to: CONCORDIA UNIVERSITY LIBRARIES. Downloaded on May 4, 2009 at 09:52 from IEEE Xplore. Restrictions apply.

Appendix

1. Case Study Derivation Run

In the sequel, we present a specific derivation run that leads to a deadlocked configuration. The derivation run is presented in Fig. 14 and Fig. 15. This has been obtained by applying the AC operational semantics rules on the term $\overline{\mathcal{A}_1}$, which corresponds to the initial state of the probabilistic transition system. It represents a single path in the probabilistic transition system corresponding to the semantic model of the activity diagram of Fig. 13. Informally, the deadlock occurs because both join nodes `join1` and `join2` are waiting for a token that will never be delivered on one of their incoming edges. There is no possible token progress since no rule can be applied.

$$\overline{\mathcal{A}_1}$$
$$\longrightarrow_1 \bar{\iota} \rightarrowtail l_1{:}\, a \rightarrowtail l_2{:}\, \texttt{Fork}(\mathcal{N}_1, l_{12})$$
$$\longrightarrow_1 \iota \rightarrowtail \overline{l_1{:}\, a \rightarrowtail l_2{:}\, \texttt{Fork}(\mathcal{N}_1, l_{12})}$$
$$\longrightarrow_1 \iota \rightarrowtail \overline{l_1{:}\, a} \rightarrowtail l_2{:}\, \texttt{Fork}(\mathcal{N}_1, l_{12})$$
$$\xrightarrow{a}_1 \iota \rightarrowtail l_1{:}\, a \rightarrowtail \overline{l_2{:}\, \texttt{Fork}(\mathcal{N}_1, l_{12})}$$
$$\longrightarrow_1 \iota \rightarrowtail l_1{:}\, a \rightarrowtail l_2{:}\, \texttt{Fork}(\overline{\mathcal{N}_1}, \overline{l_{12}})$$
$$\longrightarrow_1 \iota \rightarrowtail l_1{:}\, a \rightarrowtail l_2{:}\, \texttt{Fork}(l_3{:}\, \texttt{Merge}(\overline{l_4{:}\, b \rightarrowtail l_5{:}\, \texttt{Fork}(\mathcal{N}_2, \mathcal{N}_3)}), \overline{l_{12}})$$
$$\longrightarrow_1 \iota \rightarrowtail l_1{:}\, a \rightarrowtail l_2{:}\, \texttt{Fork}(l_3{:}\, \texttt{Merge}(\overline{l_4{:}\, b} \rightarrowtail l_5{:}\, \texttt{Fork}(\mathcal{N}_2, \mathcal{N}_3)), \overline{l_{12}})$$
$$\xrightarrow{b}_1 \iota \rightarrowtail l_1{:}\, a \rightarrowtail l_2{:}\, \texttt{Fork}(l_3{:}\, \texttt{Merge}(l_4{:}\, b \rightarrowtail \overline{l_5{:}\, \texttt{Fork}(\mathcal{N}_2, \mathcal{N}_3)}), \overline{l_{12}})$$
$$\longrightarrow_1 \iota \rightarrowtail l_1{:}\, a \rightarrowtail l_2{:}\, \texttt{Fork}(l_3{:}\, \texttt{Merge}(l_4{:}\, b \rightarrowtail l_5{:}\, \texttt{Fork}(\overline{\mathcal{N}_2}, \overline{\mathcal{N}_3})), \overline{l_{12}})$$
$$\longrightarrow_{0.1} \iota \rightarrowtail l_1{:}\, a \rightarrowtail l_2{:}\, \texttt{Fork}(l_3{:}\, \texttt{Merge}(l_4{:}\, b \rightarrowtail l_5{:}\, \texttt{Fork}(l_6{:}\, \overline{\texttt{Decision}_{0.9}(\langle g2\rangle\, l_3, \langle \neg g2 = tt\rangle\, l_7{:}\, 2.\texttt{Join}(l_8{:}\, \odot))}, \overline{\mathcal{N}_3})), \overline{l_{12}})$$
$$\longrightarrow_1 \iota \rightarrowtail l_1{:}\, a \rightarrowtail l_2{:}\, \texttt{Fork}(l_3{:}\, \texttt{Merge}(\overline{l_4{:}\, b \rightarrowtail l_5{:}\, \texttt{Fork}(l_6{:}\, \texttt{Decision}_{0.9}(\langle g2\rangle\, l_3, \langle \neg g2\rangle\, \overline{l_7{:}\, 2.\texttt{Join}(l_8{:}\, \odot)})}), \overline{l_9{:}\, 2.\texttt{Join}(l_{10}{:}\, d \rightarrowtail l_{11}{:}\, \texttt{Decision}_{0.3}(\langle g1\rangle\, \overline{l_{12}{:}\, \texttt{Merge}(l_{13}{:}\, c \rightarrowtail l_9)}, \langle \neg g1\rangle\, l_7))}), l_{12})$$

Figure 14. The Derivation Run Leading to a Deadlock

$$\longrightarrow_1 \iota \rightarrowtail l_1{:}\, a \rightarrowtail l_2{:}\, \texttt{Fork}(l_3{:}\, \texttt{Merge}(l_4{:}\, b \rightarrowtail l_5{:}\, \texttt{Fork}(l_6{:}\, \texttt{Decision}_{0.9}(\langle g2\rangle\, l_3, \langle \neg g2\rangle\, \overline{l_7{:}\, 2.\texttt{Join}(l_8{:}\, \odot)}), \overline{l_9{:}\, 2.\texttt{Join}(l_{10}{:}\, d \rightarrowtail l_{11}{:}\, \texttt{Decision}_{0.3}(\langle g1\rangle\, l_{12}{:}\, \texttt{Merge}(\overline{l_{13}{:}\, c \rightarrowtail l_9}), \langle \neg g1\rangle\, l_7))}), l_{12})$$
$$\longrightarrow_1 \iota \rightarrowtail l_1{:}\, a \rightarrowtail l_2{:}\, \texttt{Fork}(l_3{:}\, \texttt{Merge}(l_4{:}\, b \rightarrowtail l_5{:}\, \texttt{Fork}(l_6{:}\, \texttt{Decision}_{0.9}(\langle g2\rangle\, l_3, \langle \neg g2\rangle\, \overline{l_7{:}\, 2.\texttt{Join}(l_8{:}\, \odot)}), \overline{l_9{:}\, 2.\texttt{Join}(l_{10}{:}\, d \rightarrowtail l_{11}{:}\, \texttt{Decision}_{0.3}(\langle g1\rangle\, l_{12}{:}\, \texttt{Merge}(\overline{l_{13}{:}\, c} \rightarrowtail l_9), \langle \neg g1\rangle\, l_7))}), l_{12})$$
$$\xrightarrow{c}_1 \iota \rightarrowtail l_1{:}\, a \rightarrowtail l_2{:}\, \texttt{Fork}(l_3{:}\, \texttt{Merge}(l_4{:}\, b \rightarrowtail l_5{:}\, \texttt{Fork}(l_6{:}\, \texttt{Decision}_{0.9}(\langle g2\rangle\, l_3, \langle \neg g2\rangle\, \overline{l_7{:}\, 2.\texttt{Join}(l_8{:}\, \odot)}), \overline{l_9{:}\, 2.\texttt{Join}(l_{10}{:}\, d \rightarrowtail l_{11}{:}\, \texttt{Decision}_{0.3}(\langle g1\rangle\, l_{12}{:}\, \texttt{Merge}(l_{13}{:}\, c \rightarrowtail \overline{l_9}), \langle \neg g1\rangle\, l_7))}), l_{12})$$
$$\longrightarrow_1 \iota \rightarrowtail l_1{:}\, a \rightarrowtail l_2{:}\, \texttt{Fork}(l_3{:}\, \texttt{Merge}(l_4{:}\, b \rightarrowtail l_5{:}\, \texttt{Fork}(l_6{:}\, \texttt{Decision}_{0.9}(\langle g2\rangle\, l_3, \langle \neg g2\rangle\, \overline{l_7{:}\, 2.\texttt{Join}(l_8{:}\, \odot)}), l_9{:}\, 2.\texttt{Join}(\overline{l_{10}{:}\, d \rightarrowtail l_{11}{:}\, \texttt{Decision}_{0.3}(\langle g1\rangle\, l_{12}{:}\, \texttt{Merge}(l_{13}{:}\, c \rightarrowtail l_9), \langle \neg g1\rangle\, l_7)})))), l_{12})$$
$$\longrightarrow_1 \iota \rightarrowtail l_1{:}\, a \rightarrowtail l_2{:}\, \texttt{Fork}(l_3{:}\, \texttt{Merge}(l_4{:}\, b \rightarrowtail l_5{:}\, \texttt{Fork}(l_6{:}\, \texttt{Decision}_{0.9}(\langle g2\rangle\, l_3, \langle \neg g2\rangle\, \overline{l_7{:}\, 2.\texttt{Join}(l_8{:}\, \odot)}), l_9{:}\, 2.\texttt{Join}(\overline{l_{10}{:}\, d} \rightarrowtail l_{11}{:}\, \texttt{Decision}_{0.3}(\langle g1\rangle\, l_{12}{:}\, \texttt{Merge}(l_{13}{:}\, c \rightarrowtail l_9), \langle \neg g1\rangle\, l_7)))), l_{12})$$
$$\xrightarrow{d}_1 \iota \rightarrowtail l_1{:}\, a \rightarrowtail l_2{:}\, \texttt{Fork}(l_3{:}\, \texttt{Merge}(l_4{:}\, b \rightarrowtail l_5{:}\, \texttt{Fork}(l_6{:}\, \texttt{Decision}_{0.9}(\langle g2\rangle\, l_3, \langle \neg g2\rangle\, \overline{l_7{:}\, 2.\texttt{Join}(l_8{:}\, \odot)}), l_9{:}\, 2.\texttt{Join}(l_{10}{:}\, d \rightarrowtail \overline{l_{11}{:}\, \texttt{Decision}_{0.3}(\langle g1\rangle\, l_{12}{:}\, \texttt{Merge}(l_{13}{:}\, c \rightarrowtail l_9), \langle \neg g1\rangle\, l_7)}))), l_{12})$$
$$\longrightarrow_{0.3} \iota \rightarrowtail l_1{:}\, a \rightarrowtail l_2{:}\, \texttt{Fork}(l_3{:}\, \texttt{Merge}(l_4{:}\, b \rightarrowtail l_5{:}\, \texttt{Fork}(l_6{:}\, \texttt{Decision}_{0.9}(\langle g2\rangle\, l_3, \langle \neg g2\rangle\, \overline{l_7{:}\, 2.\texttt{Join}(l_8{:}\, \odot)}), l_9{:}\, 2.\texttt{Join}(l_{10}{:}\, d \rightarrowtail l_{11}{:}\, \texttt{Decision}_{0.3}(\langle g1\rangle\, \overline{l_{12}{:}\, \texttt{Merge}(l_{13}{:}\, c \rightarrowtail l_9)}, \langle \neg g1\rangle\, l_7)))), l_{12})$$
$$\longrightarrow_1 \iota \rightarrowtail l_1{:}\, a \rightarrowtail l_2{:}\, \texttt{Fork}(l_3{:}\, \texttt{Merge}(l_4{:}\, b \rightarrowtail l_5{:}\, \texttt{Fork}(l_6{:}\, \texttt{Decision}_{0.9}(\langle g2\rangle\, l_3, \langle \neg g2\rangle\, \overline{l_7{:}\, 2.\texttt{Join}(l_8{:}\, \odot)}), l_9{:}\, 2.\texttt{Join}(l_{10}{:}\, d \rightarrowtail l_{11}{:}\, \texttt{Decision}_{0.3}(\langle g1\rangle\, l_{12}{:}\, \texttt{Merge}(\overline{l_{13}{:}\, c \rightarrowtail l_9}), \langle \neg g1\rangle\, l_7)))), l_{12})$$
$$\longrightarrow_1 \iota \rightarrowtail l_1{:}\, a \rightarrowtail l_2{:}\, \texttt{Fork}(l_3{:}\, \texttt{Merge}(l_4{:}\, b \rightarrowtail l_5{:}\, \texttt{Fork}(l_6{:}\, \texttt{Decision}_{0.9}(\langle g2\rangle\, l_3, \langle \neg g2\rangle\, \overline{l_7{:}\, 2.\texttt{Join}(l_8{:}\, \odot)}), l_9{:}\, 2.\texttt{Join}(l_{10}{:}\, d \rightarrowtail l_{11}{:}\, \texttt{Decision}_{0.3}(\langle g1\rangle\, l_{12}{:}\, \texttt{Merge}(\overline{l_{13}{:}\, c} \rightarrowtail l_9), \langle \neg g1\rangle\, l_7)))), l_{12})$$
$$\xrightarrow{c}_1 \iota \rightarrowtail l_1{:}\, a \rightarrowtail l_2{:}\, \texttt{Fork}(l_3{:}\, \texttt{Merge}(l_4{:}\, b \rightarrowtail l_5{:}\, \texttt{Fork}(l_6{:}\, \texttt{Decision}_{0.9}(\langle g2\rangle\, l_3, \langle \neg g2\rangle\, \overline{l_7{:}\, 2.\texttt{Join}(l_8{:}\, \odot)}), l_9{:}\, 2.\texttt{Join}(l_{10}{:}\, d \rightarrowtail l_{11}{:}\, \texttt{Decision}_{0.3}(\langle g1\rangle\, l_{12}{:}\, \texttt{Merge}(l_{13}{:}\, c \rightarrowtail \overline{l_9}), \langle \neg g1\rangle\, l_7)))), l_{12})$$

Figure 15. The Derivation Run Leading to a Deadlock - Continued

Authorized licensed use limited to: CONCORDIA UNIVERSITY LIBRARIES. Downloaded on May 4, 2009 at 09:52 from IEEE Xplore. Restrictions apply.