Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 184 (2007) 151–170
www.elsevier.com/locate/entcs

Stochastic Object-Based Graph Grammars

Odorico M. Mendizabal,[1] Fernando L. Dotti,

Pontifícia Universidade Católica do Rio Grande do Sul, Porto Alegre - Brazil
Email: {omendizabal, fldotti}@inf.pucrs.br

Leila Ribeiro[2]

Universidade Federal do Rio Grande do Sul, Porto Alegre - Brazil
Email: leila@inf.ufrgs.br

Abstract

Object-Based Graph Grammar (OBGG) is a formal visual language suited to the specification of asynchronous distributed systems based on message passing. Model-checking of OBGG models is currently supported and a series of case studies have been developed. However, in many situations one has to evaluate non-functional aspects like availability and performance of the system under consideration. In such cases, a stochastic analysis of the system is desired. This paper is a first contribution to the stochastic analysis of OBGG models. OBGG models with occurrence rates associated to rules are translated to Stochastic Automata Networks (SAN). SAN is a Markov Chain equivalent formalism having as advantage its modularity in terms of representation and a compact mathematical solution, allowing the analysis of models with larger state space.

Keywords: Object-based graph grammar, asynchronous system, Stochastics Automata Network

1 Introduction

A very important aspect during the development of a complex system is the ability to asses both functional and non-functional properties about the system as early as possible. This ability often leads to important savings, as well as to enhancements in the quality of the resulting system.

The development of distributed systems is known as a difficult task. Besides dealing with the inherent complexity of concurrent systems, the developer also has to take distribution aspects into account. In such cases, communication latencies and availability of nodes and services, among others, become important aspects to consider which may lead to the success or not of the application.

[1] Author partially sponsored by HP-Brasil/PUCRS agreement.
[2] Author partially sponsored by CNPq, project PLATUS

doi:10.1016/j.entcs.2007.03.020

Object Based Graph Grammars (OBGG)[5] is a graphical formal specification language suitable for the specification of asynchronous distributed systems. Models defined with this formalism can be analyzed using verification (through model checking) [4,12].

Although model-checking is an important analysis method, in many situations one has to evaluate, as early as possible during the development, non-functional aspects like availability and performance of the system under consideration. Moreover, in many classes of applications it is not possible to assure certain properties. In such cases it is important to be able to associate probabilities to satisfaction or not of the property under reasoning.

Stochastic processes allows one to model the interaction of distinct phenomena, each described by a different probability distribution. Among various stochastic processes, Markov Chains [13] have been extensively investigated and used in computer sciences and engineering. Markov Chains are discrete state stochastic processes that can be continuous or discrete time, and have the memoryless property. This property assures that the transition to the next state depends only on the current state of the system and not on the previous ones. The use of exponential and geometric probability distributions associated to the transitions assures the memoryless property for continuous and discrete time Markov Chains, respectively. The solution of a Markov Chain results in the probability of each state of the chain, considering the steady state situation.

Since Markov Chains are transition systems labeled with probability distributions on transitions, they have been used as underlying model for various methods. This is the case for Stochastic Petri Nets (SPN), where the reachability graph of the net is the transition system which is annotated with the probability distributions of the associated transitions [1]. Similarly for Stochastic Process calculi, where the transition system is described by some process calculus, like for instance the π-calculus [11].

In [7] a first step towards the stochastic analysis of graph transformation systems is given. In that contribution, the authors associate (exponential) probability distributions to rules. With this, the transition system obtained from the graph grammar gives raise to a Continuous Time Markov Chain that can be analyzed with existing tools.

In this paper we propose an approach for the stochastic analysis of OBGG models. OBGG is a restricted form of graph grammar and therefore the results of [7] apply to OBGG as well. However, due to the state-space explosion problem, we avoid using Markov Chains and prefer an equivalent method with better scalability. Stochastic Automata Networks (SAN) [10] is a Markov Chain equivalent formalism having as advantage its modularity in terms of representation and the compact mathematical solution, allowing the analysis of models with larger state space, if compared to Markov Chains [6]. Once a model is represented in SAN, it is possible derive the probabilities associated to the states of interest using the PEPS tool (Performance Evaluation of Parallel Systems) [9].

The main contributions of this paper are: (i) the proposal of a stochastic exten-

sion to OBGG; and (ii) the translation of the extended OBGG to SAN, leading to a stochastic semantics of OBGGs.

The paper is organized as follows: the next section presents OBGG and the running example - the model of a token ring network. Section 3 presents the main characteristics of SAN. The extension of OBGG is proposed in Section 4. The translation from OBGG to SAN is discussed in Section 5. Section 6 analyzes the example and final remarks are in Section 7.

2 Object-Based Graph Grammars

In this paper, we consider object-based systems with the following characteristics:

- a system is composed by many objects. The state of each object is defined by its attributes, that may be elements of abstract data types or references to other objects. One object can not read nor modify the attributes of other objects;
- objects are instances of classes, that contain the specifications of the attributes and behavior of the objects belonging to that class;
- objects are autonomous entities that communicate through asynchronous message passing.

The specification of an object-based system is done via an (object-based) graph grammar. We will present the kind of graphs and rules that will be used for the specification of object-based systems. These graphs are called *object-based graphs* and were introduced in [5].

Each graph in an object-based graph grammar may be composed by instances of the vertices and edges shown in Figure 1(a). The vertices represent classes and abstract data types, whereas messages and attributes of classes are modeled as hyperedges (edges with one destination and many source vertices). We defined a distinguished graphical representation for these graphs to increase the readability of the specifications. This representation is shown in Figure 1(b). Elements of abstract data types are allowed as attributes of classes and/or parameters of messages. Note that the graph in Figure 1 defines only a scheme of the kinds of vertices and edges that may occur in a specification, and does not oblige entities or messages to have attributes. For example, this graph specifies that, if a class has attributes, they must be either of type ADT or of type Class.

A rule will express the reaction of an object to the receipt of a message. A rule of an object-based graph grammar consists of:

- *a left-hand side L*: describes the items that must be present in the current state to enable the application of the rule. The restrictions imposed to left-hand sides of rules are:
  - There must be exactly one message vertex, called trigger message (this is the message treated by this rule).
  - Only attributes of the object that is the target of the trigger message may appear.
  - Items of type ADT may be variables, that will be instantiated at the time of

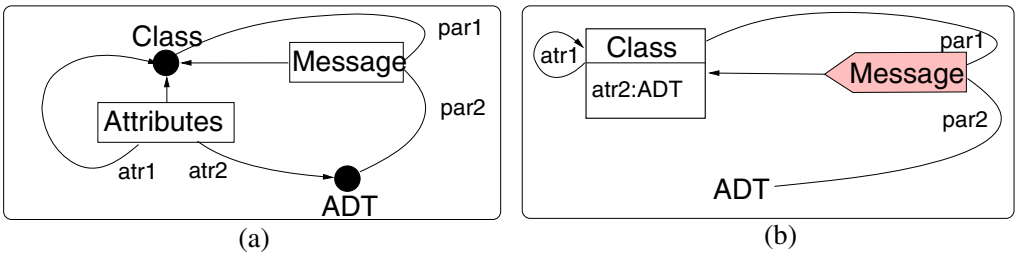


Figure 1. (a) Object-Based Graph Scheme (b) Graphical Representation of Object-Based Graphs

rule application. Operations defined in the ADTs may be used.

- *a right-hand side R*: describes the items that will be present after the application of the rule. It consists of:
  · Objects: all objects and attributes present in the left-hand side of the rule, as well as new objects (created by the application of the rule). The values of attributes may change, but attributes can not be deleted;
  · Messages to all objects appearing in R.
- *a condition*: that must be satisfied for the rule to be applied. This condition is an equation over the attributes of left- and right-hand sides.

Formally, we use typed attributed hypergraphs and rule is a (partial) graph homomorphism with application conditions. The formal definitions are presented in Section 4.

Now we can define an *object-based system*. It is composed of:

- *a Type Graph*: a graph containing information about all attributes of all classes involved in this system (an attribute may be either of ADT types or a reference to other class) and messages sent/received by each kind of object. This graph can be seen an instantiation of the object-based graph scheme described above.
- *a set of Rules*: these rules specify how the objects behave when receiving messages. For the same kind of message, there may be many rules specifying the intended behavior. Depending on the conditions imposed by these rules (on the values of attributes and/or parameters of the message), they may be mutually exclusive or not. In the latter case, one of them will be chosen non-deterministically to be executed. Note that the behavior of an object when receiving a message is not specified as a series of steps that shall be executed, but rather as an atomic change of the values of the object attributes together with the possible creation of new messages to other (or the same) objects. That is, there is no control structure to govern the application of the rules that specify the behavior of an entity. Our approach is data driven. This has the advantage that unnecessary sequentializations of computation steps are avoided because the specifier only has to care about the causal dependencies between events.
- *an Initial Graph*: this graph specifies the initial values of attributes of the objects, as well as messages that must be sent to these objects when they are created.

The messages in this graph can be seen as triggers of the execution of the object.

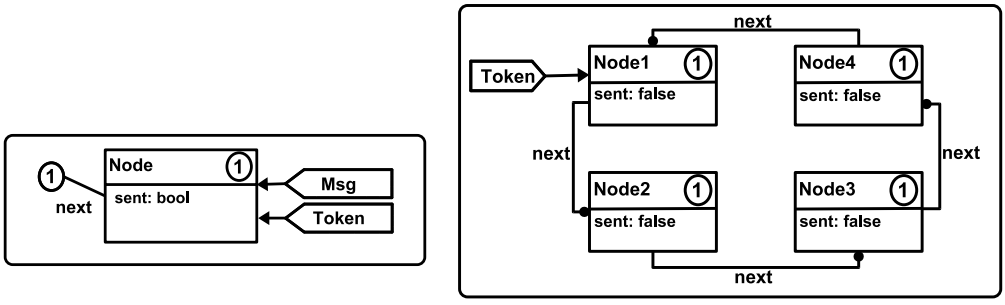
The behavior of an OBGG is given by the state transition system generated by applying rules of the grammar starting in the initial state.

2.1 Example: The Token-Ring Protocol

In this section we exemplify the use of OBGG. The token-ring protocol is relatively simple, allowing the rapid understanding by the reader as well as the exemplification of the translation process in Section 5.

The token-ring protocol is used to control the access of various stations to the shared transmission medium in a ring topology network [14]. According to the protocol, a special bit pattern, called token, is transmitted from station to station in only one direction. When a station wants to send some content through the network, it awaits for the token, holding it, and sends the message on the ring. The frame circulates the ring and the destination station may copy its contents. When the frame completes the cycle, it is received by the originating station. The originating station then removes the frame from the ring and sends the token to the next station, which then may act as already described. Having only one token, only one station may be transmitting in a given time.

Figure 2(a) is a Type Graph and defines the type *Node*. Instances of *Node* have one boolean attribute called *sent* and may receive two kinds of messages: *Msg* meaning a frame of data and *Token* meaning the token. The *link* to the next *Node* is given by the object reference *next* [3] .



(a) (b)

Figure 2. (a) Type Graph for *Node* and (b) Initial Graph for the Token Ring model.

The rules that define the behavior of this model are presented in Figure 3.

If a *Node* receives the *token* it may send a *Msg* (rule *Send*) or pass the *Token* (rule *Token Pass*). If the *Node* decides to send a *Msg*, the attribute *sent* is assigned to *true*.

When a *Msg* is received by a *Node* and it is the originating *Node* (if its attribute *sent* is *true*) then rule *Complete* is applied, removing *Msg* from the ring and gen-

[3] Graphical notation: in Figure 2(a) rectangles are vertices and numbers inside circles are the names of these vertices (those symbols are used to indicate the type of each vertex in Figures 2(b) and 3). The items within a vertex are the vertex attributes. Message symbols that appear in Figures 2(a) and 3 are hyperarcs.

erating the *Token* to the (next) *Node*. If the receiving *Node* is not the originating one (its attribute *sent* is *false*) then rule *Transmit* is applied and *Msg* is passed to the next *Node*.

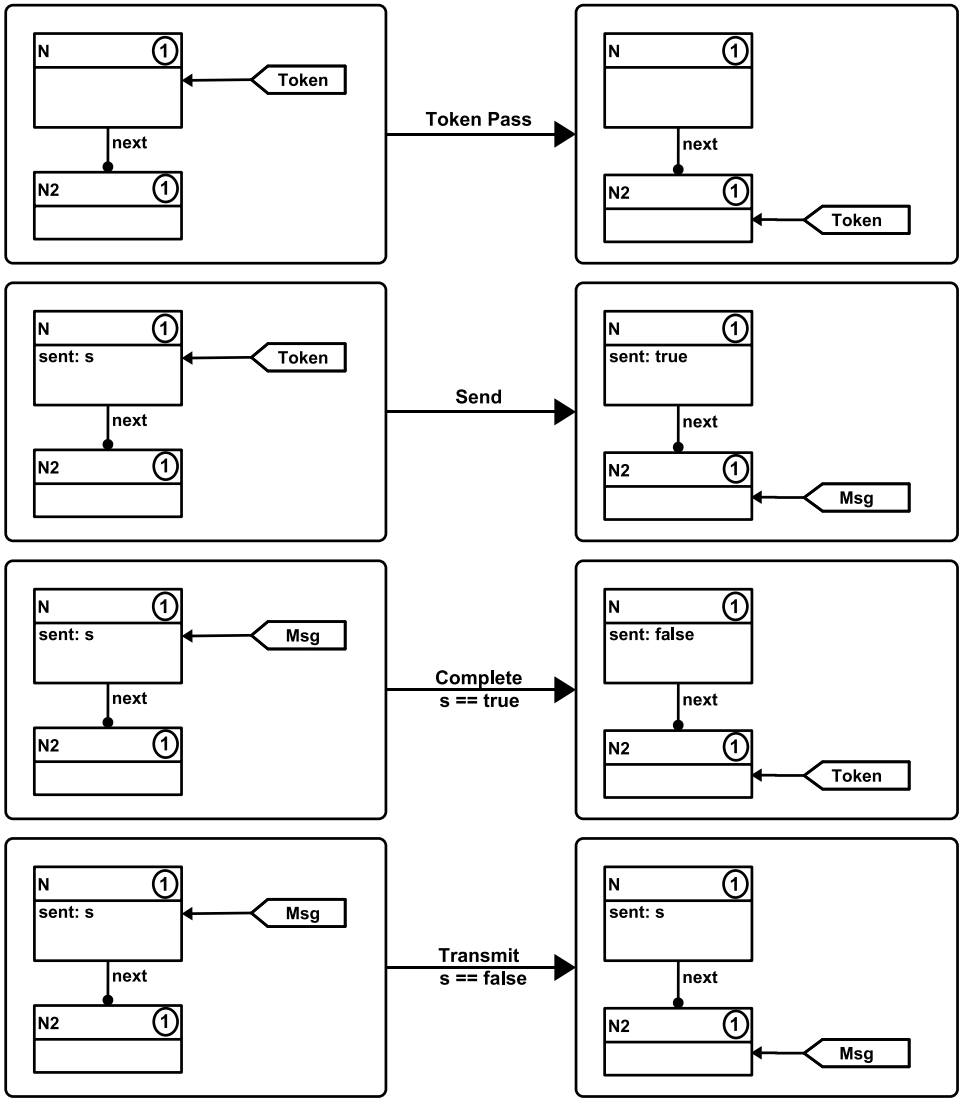


Figure 3. Rules of class *Node*.

The Initial Graph is shown in Figure 2(b), defining the various instances, attributes and messages of the start situation. A ring with four nodes is defined, called *Node1*, *Node2*, *Node3* and *Node4*. The attribute *next* of each instance refers to the next *Node*. All *sent* attributes are initially *false* and only one *Node* (*Node1*) has the *token*. This model is finite-state and generates infinite computations, allowing the steady-state analysis of the generated SAN model.

3 Stochastic Automata Networks

In the Stochastic Automata Network (SAN) formalism, a system is modeled by interacting subsystems which, in turn, are represented by automata that may behave independently or may have dependencies. According to [13,2], SAN has exactly the same application scope as Markov Chains, with the advantage that models are constructed componentwise and the mathematical solution is optimized in terms of state space [10]. SAN models can be discrete-time or continuous-time. In this paper we focus on the continuous-time case. Here we will use a less general definition of SANs, since we do not need all features of this formalism to describe a sthocastic extension of OBGGs.

An automaton is composed by states and transitions labeled with event names. A SAN model is composed by various automata. These automata may evolve independently with local events (that may affect only the local state of the automata participating in this event), whereas synchronizing events are used to model joint evolution of two or more automata. With the association of distribution probabilities to the events, the labeled transition system generated by a SAN gives raise to a Markov Chain and it is possible to calculate the steady state probability of each state of a SAN. More concretely, to each event there is an occurrence rate associated. The inverse of the occurrence rate is the mean value of the exponential distribution function that regulates the time interval between two occurrences of the event.

Definition 3.1 (Automata) An automaton is a tuple $A = (S, T, E, i)$ where S is a finite set of states, E is a finite set of events, $T \subseteq S \times 2^E \times S$ is the transition relation and $i \in S$ is the initial state. Given an automaton A, we denote its components by S_A, T_A, E_A and i_A. Given a state $s \in S$, we denote the set of events that may occur by $outputEvents(s) = \{e | \exists (s, ES, s') \in T \text{ and } e \in ES\}$, and the set of reachable states given an event by $outputStates(s, e) = \{s' | \exists (s, ES, s') \in T \text{ and } e \in ES\}$.

Definition 3.2 (Stochastic Automata Network (SAN)) A stochastic automata network (SAN) is a tuple $SAN = (SE, AL, \tau)$ where SE is a finite set of events, called **synchronizing events**, AL is a list of automata and $\tau : E \to \mathbb{R}^+$ is the **rate function**, with $E = SE \cup \biguplus_{i \in \{1..|AL|\}} (E_{Ai} - SE)$. A state of a SAN is a tuple $s = (s_1, \ldots, s_{|AL|})$, where $s_i \in S_{Ai}$.

A SAN defines the set of events that are used to synchronize the different automata during the execution. The state changes of SANs are possible when all different automata that may engage in some event are in some state in which a transition labeled with this event is possible. Note that, since there may be different transitions labeled with the same event, there may be different reachable state starting with the same state and executing the same event.

Definition 3.3 (Enabled event, State change) Given a SAN $SAN = (SE, AL, \tau)$ with $|AL| = n$ and a state $s = (s_1, \ldots, s_n)$, we say that event e is enabled in s if

(i) $\forall i \in \{1..n\}.e \in outputEvent(s_i)$ or $e \notin E_{Ai}$; and

(ii) $\tau(e) \neq 0$.

If an event e is enabled in state $s = (s_1, \ldots, s_n)$, a **state change** may occur, leading to a state $s' = (s'_1, \ldots, s'_n)$ where $s'_i = \begin{cases} x, \text{ if } x \in outputState(s_i, e) \text{ and } e \in outputEvent(s_i), \\ s_i, \text{ otherwise} \end{cases}$

Figure 4 depicts an illustrative SAN example. There are two automata represented, *Aut1* and *Aut2*. Their initial states are *st1* and *st2* (states showed by gray circles), respectively. Synchronized events are specified by *sync1* and *sync2*, whereas the local ones are specified by *loc1* and *loc2*. Note that *sync2* will be active only if *Aut1* is in *st2* and *Aut2* is in *st1* or *st2*. The local events are independent of the other automata, e.g. *loc1* will be active when *Aut1* is in *st1*, independently of the *Aut2* state.

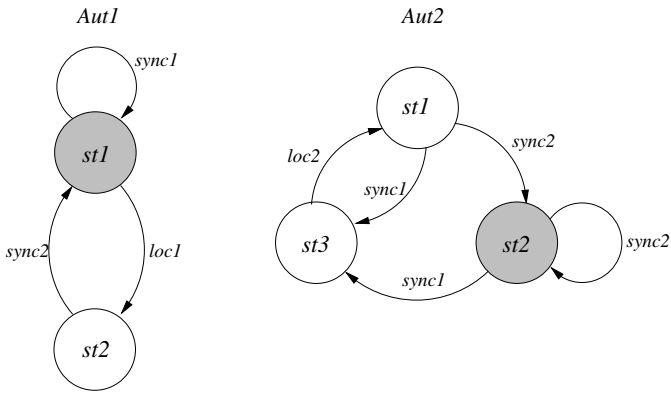


Figure 4. SAN example.

4 Stochastic Object-Based Graph Grammars

In this section we will define *stochastic object-based graph grammars*. In a graph grammar, state changes are modeled by rules. Associating occurrence rates to rules (as done for events in SAN and for transitions in SPN, for instance) it is possible to obtain a transition system semantics where each state of the system will have a probability associated, which is the probability of the system being in that state in a steady state situation (i.e. the transition system semantics gives raise to a Continuous Time Markov Chain).

Such information is very useful in the analysis of a concurrent system, and complementary to an analysis based on model checking. Some non-functional aspects can be evaluated through stochastic formalisms, such as performance and dependability. So, these information are useful to guide the developer to adjust bounds for specific requirements in complex systems.

In the following, we extend the definitions of Object-Based Graph Grammars of [5] to associate occurrence rates to rules. It is assumed that the reader is familiar with basic notions of algebraic specification.

We shall define stochastic OBGGs, short SOBGGs, over (typed and attributed) *hyper*graphs, i.e., graphs where edges can be connected to any (finite) number of vertices. Graphically, an edge is depicted as a box (whose shape may vary), and the connections to the vertices are drawn as thin lines, called *tentacles*. The tentacles of an edge are labeled by natural numbers. The main characteristics of object-based graphs are:

- Each object-based graph models a set of objects, in which the internal state of an object (its set of attributes) is described by references to other objects and/or values of pre-defined data types.
- Each object has an associated algebra, carrying the values of pre-defined data types that may be used as attributes of objects.
- The set of vertices is partitioned into two, modeling object identities and data values, respectively.
- The set of (hyper)edges is partitioned into two, modeling messages and attributes of an object, respectively. Each edge has one target (the object that receives the messages/to which the attributes belong) and may have many sources (parameters of the message/attributes of the object).

The definition of OBGGs is based on a category of graphs and *partial* morphisms.

Definition 4.1 (weak commutativity) Given two partial functions $f, f' : A \to B$, we say that f is **less defined** than f' (and we write $f \leq f'$) if $dom(f) \subseteq dom(f')$ and $f(x) = f'(x)$ for all $x \in dom(f)$. Given two partial functions $f : A \to B$ and $f' : A' \to B'$, and two total functions $a : A \to A'$ and $b : B \to B'$, we say that the resulting diagram **commutes weakly** if $f' \circ a \leq b \circ f$.

Now we introduce object-based graphs and partial morphisms. As discussed above, each hyperedge has one target vertex, and may have many source vertices. Source vertices are identified by different numbers of the tentacles, that is, a hyperedge is associated to a *list of vertices*.

Definition 4.2 (object-based graph, object-based graph morphism) Given an algebraic specification *Spec*, an **object-based graph** (OB graph) $G = (V_G, E_G, s^G, t^G, A_G, a^G)$ consists of a set V_G of sets of *vertices* partitioned into sets oV_G and aV_G (of objects and attributes, respectively), a set $E_G = (mE_G, aE_G)$ of sets of *(hyper)edges* partitioned into sets mE_G and aE_G (of messages and attributes, respectively), a total *source* function $s^G : E_G \to V_G^*$, assigning a list of vertices to each edge, a total *target* function $t^G : E_G \to oV_G$ assigning an object-vertex to each edge, an algebra A_G over *Spec*, and an attribution function $a^G : aV_G \to \mathcal{U}(A_G)$, assigning to each attribute-vertex a value from a carrier set of A_G [4].

[4] $\mathcal{U}$: **Alg**(**SPEC**) → **Set** is the forgetful functor that assigns to each algebra the disjoint union of its carrier sets.

A **(partial) OB-graph morphism** $g : G \to H$ is a tuple (g_V, g_E, g_A), where the first components are partial functions $g_V = g_{oV} \cup g_{aV}$ with $g_{oV} : oV_G \to oV_H$ and $g_{aV} : aV_G \to aV_H$ and $g_E = g_{mE} \cup g_{aE}$ with $g_{mE} : mE_G \to mE_H$ and $g_{aE} : aE_G \to aE_H$; and the third component is a total algebra homomorphism which are weakly homomorphic, that is $g_V^* \circ s^G \geq s^H \circ g_E$, $g_V \circ t^G \geq t^H \circ g_E$ and $\mathcal{U}(g_A) \circ a^G \geq a^H \circ g_V$ (if an edge is mapped, the corresponding vertices, if mapped, must have the same sources/target vertex, and if a vertex is mapped, the attributes must be the same). A morphism is called total if both components are total. The category of OB-graphs and partial OB-graph morphisms is denoted by **OBGraphP** (identities and composition are defined componentwise).

$$\begin{array}{ccc} E_G & \xrightarrow{g_E} & E_H \\ {\scriptstyle s^G}\downarrow & \leq & \downarrow{\scriptstyle s^H} \\ V_G^*) & \xrightarrow[g_V^*]{} & V_H^* \end{array} \qquad \begin{array}{ccc} E_G & \xrightarrow{g_E} & E_H \\ {\scriptstyle t^G}\downarrow & \leq & \downarrow{\scriptstyle t^H} \\ V_G & \xrightarrow[g_V]{} & V_H \end{array} \qquad \begin{array}{ccc} E_G & \xrightarrow{g_V} & E_H \\ {\scriptstyle a^G}\downarrow & \leq & \downarrow{\scriptstyle a^H} \\ \mathcal{U}(A_G) & \xrightarrow[\mathcal{U}(g_A)]{} & \mathcal{U}(A_H) \end{array}$$

To distinguish different kinds of vertices and edges, we will use the notion of *typed graphs* [3,8]: every graph is equipped with a morphism *type* to a fixed graph of types [5]. Since the types will constitute the static part of the definition of a class, we will call the graph of types as *class graph*, and some restrictions will be imposed to guarantee that corresponds to a class in the sense of the object paradigm (the first restriction says that there are no data values in a class graph, they are represented by the name of data types, and the second imposes that each class can have exactly one list of attributes).

Definition 4.3 (typed OB-graphs) Let *Spec* be a specification. An OB-graph C is called a **class graph** iff (i) A_C is a final algebra over *Spec*, (ii) for each object vertex $v \in oV_C$ there is exactly one attribute hyperedge ($ae \in aE_C$) with target v. A **typed OB-graph over** C is a pair $OG^C = (OG, type^{OG})$ where OG is an OB-graph called **instance graph** and $type^{OG} : OG \to C$ is a total OB-graph morphism, called the **typing morphism**.

A morphism between typed OB-graphs OG_1^C and OG_2^C is a partial OB-graph morphism $f : OG_1 \to OG_2$ such that $type^{OG_1} \geq type^{OG_2} \circ f$. The category of OB-graphs typed over a class graph C, denoted by **OBGraphP**(**C**), has OB-graphs over C as objects and morphisms between typed OB-graphs as arrows (identities and composition are the identities and composition of partial OB-graph morphisms).

Rules define how objects react when receiving messages. Each rule expresses how one particular message will be treated (many rules may be necessary to describe all possible reactions to one message).

Definition 4.4 (rule) Let C be a class graph, *Spec* be a specification and X be a set of variables of sorts of *Spec*. A rule is a pair (r, Eq) where Eq is a set of equations over the specification *Spec* and $r : L \to R$ is a C-typed OB-graph morphism s.t.

[5] Note that, due to the use of partial morphisms, this is not just a comma category construction: the morphism *type* is total whereas morphisms among graphs are partial, and we need weak commutativity instead of commutativity.

(i) L and R are finite;

(ii) a message hyperedge is deleted: $\exists! e \in mE_L$, called *trigger(r)*, $trigger(r) \notin dom(r_E)$;

(iii) only attributes of the target of the message may appear in L: $(aE_L = \emptyset) \vee ((\exists! e \in aE_L) \wedge t^L(e) = t^L(trigger(r)))$;

(iv) attributes of existing objects may not be deleted nor created: $\forall o \in oV_L.(\exists e \in aE_L.t^L(e) = o \Rightarrow \exists e' \in aE_R.t^L(e') = r^V(o))$;

(v) objects may not be deleted: $\forall o \in oV_L.o \in dom(r^V)$;

(vi) the algebra of r is a term algebra $T_{Spec'}(X)$ over the specification $Spec$ including a set of equations Eq;

(vii) attributes appearing in L may only be variables of X: $\forall v \in (aV_L \cup aE_L).a^L(v) \in X$;

(viii) the algebra homomorphism component of r is the identity.

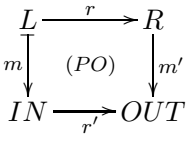
We denote by $Rules(C)$ the set of all rules over a class graph C.

To define a SOBGG, we first define a class graph, modeling the types of objects, messages and attributes that may be present in the system. Then we define the behavior of the system using rules, and the possible initial states (that are graphs containing instances of the types in the class graph).

Definition 4.5 (stochastic object-based graph grammar (SOBGG)) Given a graph of types C. A **stochastic object-based graph grammar** is a tuple $SOBGG = (Spec, X, C, IG, N, n, \rho)$ where $Spec$ is an algebraic specification, X is a set of variables, C is a class graph, IG is graph typed over C, called the **start graph**, N is a set of rule names, $n : N \to Rules(C)$ assigns a rule to each rule name and $\rho : N \to \mathbb{R}^+$ assigns a rate to each rule.

The behavior of a SOBGG is obtained by applying the rules successively to a start graph. Each rule application deletes one message (the trigger of the rule) and may change the value of internal attributes, create new messages and/or objects. Formally, the effect of a rule application is obtained by a pushout in the corresponding category (typed object-based graphs).

Definition 4.6 (derivation step, derivation) Let $SOBGG = (Spec, X, C, IG, N, n, \rho)$ be an SOBGG, $(r : L \to R, Eq) \in n(N)$ be a rule, and IN be a graph typed over C. A **match** for r in IN is a total morphism $m : L \to IN$ in $\mathbf{OBGraphP(C)}$. A **derivation step** $IN \overset{r,m}{\Longrightarrow} OUT$ using rule r and match m is a pushout in the category $\mathbf{OBGraphP(C)}$. The morphism $r' : IN \to OUT$ is called **derived rule**.



A derivation sequence of a $SOBGG$ is a sequence of derivation steps $G_i \overset{r_i,m_i}{\Longrightarrow} G_{i+1}$, $i \in \{0, \ldots, n\}$, $n \in \mathbb{N}$, where $G_0 = IG$ and $r_i \in Rules$ for all $i \in \{0, \ldots, n\}$. The class of all derivation sequences of an $SOBGG$ is denoted by $SDer_{SOBGG}$. The class of all reachable graphs in $SDer_{SOBGG}$ is defined by $State_{SOBGG} = \{G \mid G = IG \vee IG \overset{r,m}{\Longrightarrow} G \in SDer_{SOBGG}\}$

The computations of a SOBGG are exactly the same as the underlying OBGG (grammar without the tax function). The following definition of the behavior semantics describes these computations, without considering the stochastic behavior (that will be considered in section 5.2).

Definition 4.7 (SOBGG Behavior Semantics) Given a stochastic Object-Based Graph Grammar $SOBGG = (Spec, X, C, IG, N, n, \rho)$, its *behavior semantics* $BehSem(SOBGG)$ is defined by the labeled transition system $TS = (S, L, q_0, \rightarrow)$, where:

- $S = State_{SOBGG}$ is the set of states;
- $L = N$ is the set of transition labels;
- $q_0 = IG$ is the initial state;
- $\rightarrow$ is given by following rule:

$$\frac{G \overset{nr,m}{\Longrightarrow} G' \in SDer_{SOBGG}}{G \overset{nr,m}{\longrightarrow} G'}$$

5 Stochastic Semantics of OBGGs

To associate probabilities to the states of the behavior semantics we have to solve the respective stochastic model. To do this, we translate SOBGG to stochastic automata network (SAN), solve the respective SAN, and then finally complete the stochastic semantics of SOBGG.

5.1 Translation of OBGGs into SANs

In the translation from SOBGGs to SANs, attributes and messages of SOBGG objects originate SAN automata states and rules are mapped to transitions, events and rates.

The state of each object is associated with a set of automata: one automaton for each attribute; one automaton for each object reference; one automaton for each type of message that the object may receive. All attributes and messages related to an object are defined by the class graph. Given a class $CG = (V_{CG}, E_{CG}, s^{CG}, t^{CG}, A_{CG}, a^{CG})$, the elements of sets oV_{CG} and aV_{CG} represent the types (of objects and data, respectively) that are allowed in the system. The set mE_{CG} describes the types of messages of the system, and the set aE_{CG} has as components the attribute hyperarcs (that connect each attribute to each object vertex). Elements of mE_{CG} may be deleted or created during the execution of the system, whereas the set aE_{CG} must be stable (because objects may not loose nor gain attributes).

However, since we are dealing with finite state models, mE_{CG} may not grow indefinitely. For each type of message $msg \in mE_{CG}$, let $max(msg)$ denote the maximum number of instances of such messages that may exist in some state of an SOBGG. A further restriction is that we do not allow the creation of objects. This latter restriction could be relaxed by allowing a bounded number of objects to be

created, but this is subject to future work.

Based on the class graph and the initial graph of an SOBGG, we will construct the sets of automata states that will be used to build the corresponding SAN. In the following definitions, we will use as attributes of classes and parameters of messages lists of n elements, but note that n may be zero, leading to an empty list. The initial graph will be used to get the information about the data values (defined in the algebra component) and objects that may exist in the system. We will use a function *states* that, given a vertex, returns the set of values of this type. In case this vertex is the name of a data type, the result is the corresponding carrier set of the algebra. In case it is an object type, it returns the set of vertices of this type (object ids) in the initial graph.

The rules of the graph grammar will give raise to transitions and events of the SAN, and the associated rates will become the rates of the corresponding events.

First, we will define how the event set of the resulting SAN is obtained, and then how each individual automaton is constructed.

Definition 5.1 (translation of rule applications into events) Let $SOBGG = (Spec, X, CG, IG, N, n, \rho)$ be an stochastic object-based graph grammar. We define the set of events E induced by $SOBGG$, denoted as $Events(SOBGG)$, as

$$Events(SOBGG) = \{e \in events(ruleName) | \forall ruleName \in N\}$$

where $events(rn) = \{(rn, dr : IN \rightarrow OUT) | \exists\ CG\text{-typed graph } IN$ with same algebra as IG, $m : L \rightarrow IN$ is a surjective match, and dr is the derived rule of the application of $n(rn)$ to $m\}$.

For each event $e = (rn, dr) \in E$, we define $message(e)$ as the message hyperedge deleted by dr, $msgType(e) = type^G(message(e))$, $object(e) = t^G(message(e))$, $ruleName(e) = rn$, $param(e) = s^G(message(e))$.

In the instances of a class, an automaton will be created for each attribute, the states of each of these automata represent the values that these attributes may assume, and the transitions describe the possible state changes that were performed by some rule application.

Definition 5.2 (generation of attribute automata) Let $SOBGG = (Spec, X, CG, IG, N, n, \rho)$ be an stochastic object-based graph grammar and $obj \in oV_{IG}$ be an object identity. The **attribute automata** generated for obj $AutAttr_{obj}$ is defined as

$$AutAttr_{obj} = \langle AutAttr(v_1)_{obj}, \ldots, AutAttr(v_n)_{obj} \rangle$$

where $s^{IG}(ae) = \langle v_1, .., \ldots, v_n \rangle$, ae is the attribute hyperedge of obj, and each $AutAttr(v_i)_{obj} = (S_i, T_i, E_i, ini_i)$ is an automaton defined as

- $S_i = states(type^{IG}(v_i))$,
- $ini_i = v_i$,

- the set of transitions T_i is obtained as follows: for each event $e = (rn, dr : IN \to OUT) \in Events(SOBGG))$, if $obj = object(e)$ the attribute vertex ae of obj in IN is deleted by dr and re-created as ae' in OUT, with $s^{IN}(ae) = \langle v_1, .., \ldots, v_n \rangle$, and $s^{OUT}(ae') = \langle v'_1, .., \ldots, v'_n \rangle$, the transition $v_i \xrightarrow{\{e\}} v'_i$ is in T_i.
- E_i is the union of the sets of events used as labels of transitions in T_i.

In Figure 5, *sent_Node1* and *next_Node1* attribute automata represent the *sent* and *next* attributes for the object $Node1$ of the Token Ring OBGG model according to our translation approach (the initial state of each automata is the gray circle). The names of events used to label transitions are composed of name of applied rule (that gave raise to this event), a list of attribute name and the respective value needed to build the match, and the object that receives the message. Having a rule name and a match uniquely specifies a rule application, that is, an event, and therefore we will not use the derived rule in the graphical representation. The other automata depicted in this figure will be explained along this section.

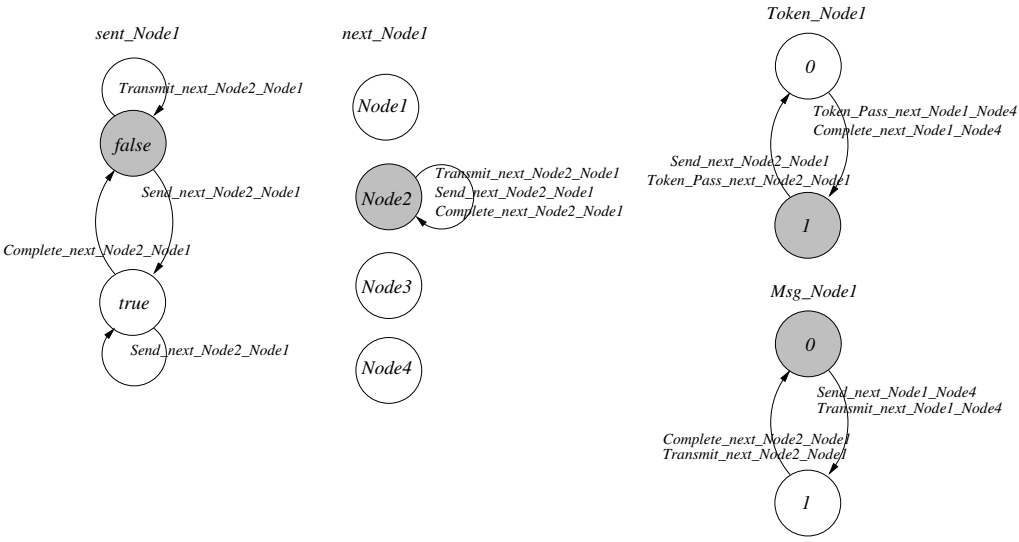


Figure 5. Token Ring model translated.

In the generated SAN, there will be an automaton for each kind of message and parameter value that the object may receive, and the states of each of these automata will represent the number of messages of each kind present in the current state of the system. Transitions will model deletion and creation of messages.

Definition 5.3 (generation of message automata) Let $SOBGG = (Spec, X, CG, IG, N, n, \rho)$ be an stochastic object-based graph grammar and $obj \in oV_{IG}$ be an object identity. The **message automata** generated for obj $AutMsg_{obj}$ is defined as

$$AutMsg_{obj} = ((A_i)_{i \in MAttr})_{obj}$$

where $MAttr = \{msg\} \times states(tv_1) \times \ldots \times states(tv_n)$, $msg \in \{m \in mE_{CG} | t^{CG}(m) = class, type^{IG}(obj) = class\}$ (that is, msg is a message type that may be received by object obj), $s^{CG}(msg) = \langle tv_1, .., \ldots, tv_n \rangle$. Each automata $A_i = (S_i, T_i, E_i, ini_i)$, with $i = (msg, v_1, \ldots, v_n)$, is defined as follows:

- $S_i = \{0..max(msg)\}$,
- ini_i is the number of messages of type msg and parameters $\langle v_1, .., \ldots, v_n \rangle$ present in IG;
- the set of transitions T_i is obtained as follows:
  - events deleting messages of type $(msg, v_1, \ldots, v_n)$: for each event $e = (rn, dr : IN \to OUT) \in SE$, with $msgType(e) = msg$, $object(e) = obj$, parameters $param(e) = \langle v_1, .., \ldots, v_n \rangle$, the following transitions are added to T_i:

$$\{i \xrightarrow{e} i - 1 | 0 < i \leq max(msg)\}$$

  - events creating message of type $(msg, v_1, \ldots, v_n)$: for each event $e = (rn, dr : IN \to OUT) \in SE$, such that e creates x messages of type $(msg, v_1, \ldots, v_n)$, the following transitions are added to T_i:

$$\{i \xrightarrow{e} i + x | 0 < i \leq max(msg)\}$$

In figure 5, *Token_Node1* and *Msg_Node1* are message automata for object $Node1$, corresponding to messages *Token* and *Msg*, respectively. Since these messages do not have parameters, only these message automata will be generated for each object of type *Node*.

From the initial graph, the initial state of the SAN is derived. This is done by setting the initial state of each automaton of the resulting SAN to represent the attributes, messages and parameters in the initial graph. Using the PEPS tool, this is done by declaring a partial reachability function which defines the initial state of each automaton. For instance, we translate the initial graph depicted in Figure 2(b) into the following partial reachability function:

```
partial reachability =
(st sent_Node1==false) && (st Token_Node1==1) && (st Msg_Node1==0) &&
(st sent_Node2==false) && (st Token_Node2==0) && (st Msg_Node2==0) &&
(st sent_Node3==false) && (st Token_Node3==0) && (st Msg_Node3==0) &&
(st sent_Node4==false) && (st Token_Node4==0) && (st Msg_Node4==0);
```

Definition 5.4 (Translation of SOBGG into SAN) Let $SOBGG = (Spec, X, CG, IG, N, n, \rho)$ be an stochastic object-based graph grammar. We define the generated SAN $SAN = (SE, AL, \tau)$ inductively as follows:

(i) $SE = events(SOBGG)$;

(ii) $\tau(e) = \rho(rn)$, where $rn = ruleName(e)$;

(iii) $AL = (Aut_{obj})_{obj \in oV_{IG}}$ where $Aut_{obj} = (AutAttr_{obj}, AutMsg_{obj})$, and $AutAttr_{obj}$ is the attribute automaton of obj and $AutMsg_{obj}$ is the message automaton of obj.

5.2 Stochastic Semantics of SOBGGs

Now, given a graph typed over a class model graph, that is, a state of an OBGG system, this state can be translated to a set of automata according to the following definition.

Definition 5.5 (translation of OB-graphs to SAN states) Let $SOBGG = (Spec, X, CG, IG, N, n, \rho)$ be an stochastic object-based graph grammar and $SAN = (SE, AL, \tau)$ be the corresponding SAN. Then any reachable state G of $SOBGG$ can be translated to a global state GS as follows:

- for all object obj in G: let ae be the attribute edge of obj and $s^G(ae) = \langle v_1, \ldots, v_n \rangle$. The state of each attribute automata $AutAttr(v_i)$ of obj must be v_i.
- for all messages msg with parameters $\langle v_1, \ldots, v_n \rangle$ in G: let $tmsg$ be the type of message msg in the class graph. The state of each message automata $AutMsg_{(tmsg, v_1, \ldots, v_n)}$ must be the number of messages of type $tmsg$ with exactly the same parameter values as msg in graph G.

To define the semantics of a stochastic object-based graph grammar, we just have to translate it to a SAN, solve this SAN, and associate the corresponding probability to each OB-graph.

Definition 5.6 (stochastic semantics of SOBGG) Let $SOBGG = (Spec, X, CG, IG, N, n, \rho)$ be an stochastic object-based graph grammar and $SAN = (SE, AL, \tau)$ be the corresponding SAN. The **stochastic semantics of SOBGG** is the transition system $stoST = (S, L, q_0, \rightarrow, \tau)$, where $(S, T, q_0, \rightarrow)$ is the behavior semantics of $SOBGG$ and $\tau : S \rightarrow \mathbb{R}^+$ associates a probability to each state $s \in S$ by translating this state to a SAN global state and checking the probability of this state.

6 Model Analysis

This section presents results obtained from the steady state analysis of the Token Ring translated model. We applied the translation steps as stated in Section 5, obtaining a SAN model and evaluated this model with PEPS tool.

A token ring network with four nodes was modeled, having a static topology, *i.e.* a node does not change its neighbors. Due to the static topology of the example, the automata *next_Node1* to *next_Node4*, representing the *next* attribute of the instances are not necessary. Therefore, each node was modeled with three automata. One for the attribute *sent* and two for the possible input messages. There are two states in each automaton. This results in 12 automata and a product state space of 4096 states. However, considering the initial state as described, only 20 states are reachable.

Assigning rates to the rules of the model and solving the corresponding SAN, we obtain the probability associated to each state of the model. In this example, the rates defined for the rules are illustrative. With the results obtained we can analyze the model with respect to some functional and non-functional requirements. As

example of functional analysis, some scenarios that are not expected in a Token Ring protocol are evaluated. For example:

- *property 1*: it is impossible that at the same time more than one node receives a token;
- *property 2*: it is impossible that at the same time more than one node have messages transmitted in the ring;

In order to analyze such cases, we define integration functions with the PEPS tool. An integration function is an expression using the calculated probabilities of the states of the SAN model. For *property 1*, we observe that the probability of more than one node having Tokens simultaneously is equal to 0.0%. The corresponding integration function is:

```
more_than_one_token_received =
    (nb [Token_Node1 .. Token_Node4] 1) > 1;
```

In this case, the function *nb* returns the number of autamata, from *Token_Node1* to *Token_Node4*), simultaneously in state *1*. The whole expression returns the probability of more than one node having a token. This probability is 0.0%.

For *property 2* the analysis is analogous but we look at the probability of attribute *sent* being true for more then one node. As expected, this probability also is 0.0%. The corresponding integration function is:

```
more_than_one_token_sent =
    (nb [sent_Node1 .. sent_Node4] true) > 1;
```

Although we can evaluate some functional properties using the probabilities associated to the states, for some properties it is necessary to evaluate the causal dependence of the rule applications. In such cases, model checking OBGG models [4] is a better choice than solving SOBGG models. On the other hand, in order to evaluate non-functional properties SOBGG is needed.

For the quantitative analysis of the Token Ring model we have analyzed the impact of the rates of the various rules in the probabilities of situations of interest [6]. We associate rates $t1$, $t2$, $t3$, and $t4$ to rules Token_Pass, Send, Complete and Transmit, respectively. Figure 6 presents the rule Complete for the SOBGG model. Note that a rate $t3$ was added to the original OBGG rule.

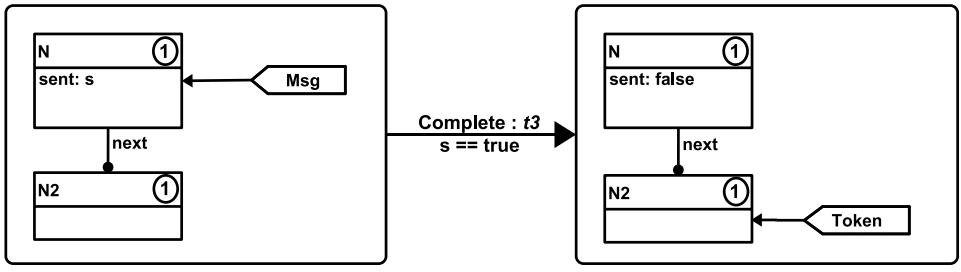


Figure 6. SOBGG version to rule Complete.

The probabilities we look at are:

[6] In this context, situations of interest means scenarios with different workload.

- (i) the probability of the network being used by *Node1* (*i.e.* the probability of attribute *sent_Node1* being true);
- (ii) the probability of the network not being used by *Node1* (should be a complement of (i));
- (iii) the probability of having a data message being transmitted by any node (*i.e.* the probability of any of the automata *sent_Node1* to *sent_Node4* being in state true) - since all nodes have the same rates, the ring should be shared equally and therefore this value should be 4 times the value of (i);
- (iv) the probability of having no data message under transmission (should be a complement of (iii));
- (v) the probability of having a token under transmission (should be same as (iv));

Table 6 shows the results for these cases, presenting also the integration function for each situation. The operator *st* applied to an automaton name means the state of the mentioned automaton.

Formula	**Probability (%)**	
	t1 = 10, t2 = 5, t3 = 7, t4 = 4	t1 = 14, t2 = 3, t3 = 7, t4 = 4
(i) st sent_Node1 == true	20.42	18.20
(ii) st sent_Node1 == false	79.58	81.80
(iii) (nb [sent_Node1 .. sent_Node4] true) > 0	81.70	72.81
(iv) (nb [sent_Node1 .. sent_Node4] false) == 4	18.30	27.18
(v) (nb [Token_Node1 .. Token_Node4] 1) > 0	18.30	27.18

Table 1
Quantitative analysis of the Token Ring translated model.

These rates mean that the nodes pass the token more often than they send messages. Further, note that when we just decrease the rate associated to sending messages, and increase the rate of passing the token, the occupation of the ring by data messages decreases, as expected.

Other scenarios could be specified changing the values of the rates. This could be useful, for instance, to guide a system developer to predict the network throughput (considering various workloads). The definition of rates to represent a specific reality (a real network) will be addressed in future work.

7 Final Remarks

In this paper we introduce Stochastic Object-Based Graph Grammars (SOBGG), an extension of Object-Based Graph Grammars (OBGG) to allow the stochastic analysis of the system being modeled. This type of analysis is suitable to evaluate non-functional properties of OBGG systems such as performance and dependability. levels.

In order to solve the stochastic model, we map SOBGG to Stochastic Automata Network (SAN). The solution of the corresponding SAN model with existing tools allows to associate probabilities to the reachable states of the SOBGG. SAN were

preferred due to the compact mathematical solution, allowing to solve models with larger state space if compared to Markov Chains, and also due to the modular presentation.

The mapping of SOBGG to SAN was described. Each instance of an object from the OBGG will generate various automata, one for each attribute and one for each possible input message type. The events and transitions of the automata are given by the rules of the SOBGG. The synchronization offered by SAN events is suitable to express atomicity in rule applications.

Once the corresponding SAN model has been generated and solved, the probabilities of each global state can be used for analysis. PEPS further offers integration functions which allow to obtain the probabilities associated to particular states of an automaton, and not only of global states.

As could be noticed, in order to analyse the SOBGG model the user has to know the generated SAN model to extract results. More concretely, the user writes integration functions about the states of the various SAN automata and their probability. One important future work should be to allow the analysis of the model based on the SOBGG abstractions and not on the generated SAN, which should be, ideally, hidden from the user.

Various optimizations can be made on the SAN model obtained from this translation, comprising important topics of future work. For instance, in models with static topology it is possible to eliminate automata for the object references, saving space. It should also be noticed the importance of reducing the state space of each element of the model such that it becomes treatable by a computational tool.

In the case studies carried out so far, we could notice that SOBGG models, when translated to SAN, tend to generate a large product state space but a reduced reachable state space. The PEPS tool, in the current version, first calculates the product state space and then solves the system, assigning probabilities to the reachable states. Therefore our models are restricted in the product state space. A new version of the PEPS tool is being developed whereby the product state space is avoided and the reachable state space is calculated directly. This enhancement will allow the stochastic analysis of models with considerable size.

References

[1] F. Bause and P.S. Kritzinger. *Stochastic Petri Nets*. Vieweg Verlag, 2 edition, 2002.

[2] L. Brenner, P. Fernandes, and A. Sales. The need for and the advantages of generalized tensor algebra for kronecker structured representations. In 20^{th} *Annual UK Performance Engineering Workshop*, pages 48–60, 2004.

[3] A. Corradini, U. Montanari, and F. Rossi. Graph processes. *Fundamenta Informaticae*, 26(3/4):241–265, 1996.

[4] F. L. Dotti, L. Foss, L. Ribeiro, and O. M. Santos. Verification of object-based distributed systems. In *6th International Conference on Formal Methods for Open Object-Based Distributed Systems*, volume 2884 of *LNCS*, pages 261–275, France, 2003. Springer-Verlag.

[5] F. L. Dotti and L. Ribeiro. Specification of mobile code systems using graph grammars. In *4th International Conference on Formal Methods for Open Object-Based Distributed Systems*, volume 177, pages 45–63. IFIP Conference Proceedings, Kluwer Academic Publishers, 2000.

[6] P. Fernandes, B. Plateau, and W. J. Stewart. Numerical evaluation of stochastic automata networks. In *Proceedings of the Third International Workshop on Modeling, Analysis, and Simulation of Computer and Telecommunication Systems*, pages 179–183, 1995.

[7] R. Heckel, G. Lajios, and S. Menge. Stochastic graph transformation systems. In H. Ehrig, G. Engels, F. Parisi-Presicce, and G. Rozenberg, editors, *Proc. 2nd Intl. Conference on Graph Transformations (ICGT 2004)*, volume 3256 of *Lecture Notes in Computer Science*, pages 210–225. Springer, 2004.

[8] M. Korff. True concurrency semantics for single pushout graph transformations with applications to actor systems. In R. J. Wieringa and R. B. Feenstra, editors, *Information Systems - Correctness and Reusability*, pages 33–50. World Scientific, 1995.

[9] B. Plateau and K. Atif. Peps: a package for solving complex Markov models of parallel systems. In *Proceedings of the 4th International Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, 1988.

[10] Brigitte Plateau. On the stochastic structure of parallelism and synchronization models for distributed algorithms. In *SIGMETRICS*, pages 147–154, 1985.

[11] C. Priami. The stochastic pi-calculus. *The Computer Journal*, 38:578–589, 1995.

[12] O. M Santos, F. L. Dotti, and L. Ribeiro. Verifying object-based graph grammars. *Eletronic Notes in Theoretical Computer Science*, 109:125–136, 2004.

[13] W. J. Stewart. *Introduction to the numerical solution of Markov chains*. Princeton University Press, 1994.

[14] A. S. Tanenbaum. *Computer Networks*. Prentice-Hall, Englewood Cliffs, New Jersey, third edition, 1996.