StoKlaim: A Stochastic Extension of Klaim [*]

Rocco De Nicola
Università degli Studi di Firenze

Diego Latella
C.N.R. - I.S.T.I.

Joost-Pieter Katoen
RWTH Aachen

Mieke Massink
C.N.R. - I.S.T.I.

Report version V1.3 of May 23, 2006

Abstract

Global (or network-aware) computing entails large-scale networks of computers performing tasks in a cooperative and coordinated manner. Programming and modeling languages such as Klaim focus on key functional aspects of global computing such as distribution awareness, (code and agent) mobility, and privacy aspects. This paper instead concentrates on their performance and dependability aspects. It proposes StoKlaim, an extension of Klaim which permits the description of random phenomena such as spontaneous computer crashes and spurious network hick ups. The operational semantics of StoKlaim is based on continuous-time Markov chains. The approach is illustrated by modeling the spreading of a virus through a network.

1 Introduction

1.1 Global computing

During the last couple of decades, computer systems have changed significantly: stand-alone, static devices executing programs autonomously have evolved to large-scale networks of computing devices performing tasks in a cooperative and coordinated manner. These modern, complex distributed systems—also known as *global or network-aware computers* [10]—are highly dynamic and have to deal with frequent changes of the network environment. The world wide web is a major example of such a global "computer". Features such as distribution awareness and code mobility which were absent or deliberately invisible in previous computer generations play a prominent rôle in global computing. Dedicated programming and specification formalisms have been developed that can deal with issues such as (code and agent) mobility, remote execution, security aspects and data privacy and integrity. Important examples of such languages and frameworks are, among others, Obliq [9], Seal [11], ULM [6] and Klaim (*Kernel Language for Agents Interaction and Mobility* [4]).

1.2 Dependable global computing

Performance and dependability issues are of utmost importance for "network- aware" computing, due to the enormous size of systems—networks typically consist of thousands or even millions of nodes—and their strong reliance on mobility and interaction. Spontaneous computer crashes may easily lead to failure of remote execution or process movement, while spurious network hick ups may cause loss of code fragments or unpredictable delays. The enormous magnitude of computing devices involved in global computing yields failure rates that no longer can be ignored. The

[*]The work presented in this report has been partially supported by EU Project Software Engineering for Service-Oriented Overlay Computers (SENSORIA, contract IST-3-016004-IP-09).

presence of such random phenomena implies that correctness of global computing software and their privacy guarantees are no longer rigid notions like:

> "either it is safe or it is not"

but have a less absolute nature, e.g.:

> "in 99.7% of the cases, privacy can be ensured"

The intrinsic complexity of global computers, though, complicates the assessment of these issues severely. Systematic methods, techniques and tools—all based on solid mathematical foundations i.e., *formal methods*, are therefore needed to establish performance and dependability requirements and guarantees. This paper attempts to make a considerable step into this direction by extending a successful programming and specification formalism for global computing, KLAIM, with random delays.

1.3 Modeling dependable global computing

To facilitate the incorporation of random phenomena in models for network-aware computing, we propose a simple, yet powerful extension of KLAIM [12, 4]. KLAIM is an experimental language for distributed systems that is aimed at modeling and programming mobile code applications, i.e., applications for which exploiting code mobility is the prime distinctive feature. Its distinguishing feature is the explicit use of localities for modeling data or computational resources distribution. It is heavily based on the process algebras CCS and π-calculus as well as on the coordination paradigm of Linda. KLAIM models *networks* as finite collections of sites, each equipped with a (physical) address, where processes can execute and data can reside. Processes and actions—like in traditional process algebras—are key elements in KLAIM and possess the possibility to explicitly refer to and control the spatial structure of the global network at any point of their evolution. Processes are the active computational entities and may run concurrently, either at the same site or at distinct ones. They interact in an asynchronous fashion via multiple distributed *tuple spaces*, a generalization of the well-known single shared tuple space in Linda [18]. Actions in KLAIM explicitly indicate the (possibly remote) site at which they will have effect. KLAIM supports core aspects for global computing such as *process distribution*, *remote evaluation* (a process spawns another process for execution to another site), *code on demand* (a process may download code from a remote site to execute it locally), and *site creation*.

In the proposed extension, referred to as STOKLAIM, these actions are assumed to have a random duration governed by a negative exponential distribution. The resulting operational model is therefore a continuous-time Markov chain (CTMC, for short), one of the most popular models for the evaluation of performance and dependability of information processing systems. Our extension is inspired by Markovian extensions of traditional process algebras; for recent surveys see, e.g. [22, 24]. Exponential durations keep the semantics simple but seem to have a somewhat limited expressiveness; however, proper combinations of exponential distributions can approximate general distributions arbitrarily closely. Consequently, they are adequate for many real-life phenomena (such as failure rates and inter-arrival times) and—more importantly—they are the most appropriate stochastic approximation if only the mean duration is known [1]. Our modelling approach is exemplified by modeling the spreading of a virus through a network. A preliminary version of the language STOKLAIM has been published in [15].

1.4 Specifying properties of dependable global computing

Models specified in STOKLAIM thus yield a Markov chain as operational model. To assess dependability aspects, typically long-run or transient probabilities of such chains are determined. We are interested in a more recent technique that determines performance and dependability *guarantees*

[1] Technically speaking, they maximize the entropy when only the mean is known. There is no other continuous distribution function for which this property holds.

in a fully automated manner using model checking. Such guarantees are formulated in a proper stochastic temporal logic for mobility which is the topic of the companion report [13]. In the present paper we instead show how the CTMC obtained from a StoKlaim specification modeling a system can be used for traditional analysis.

1.5 Organization of the paper

Section 2 gives a brief introduction into the basics of continuous-time Markov chains. Section 3 introduces the modeling and programming language StoKlaim, formally defines its syntax and its operational semantics, and details out how Markov chains are obtained from StoKlaim specifications. Section 4 presents the virus spreading example. Section 5 reviews existing work on stochastic languages for mobility and, finally, Section 6 concludes.

2 Markov chains

We briefly introduce continuous-time Markov chains and the notations we use for them. A comprehensive treatment can be found in textbooks such as, e.g., [25].

2.1 Exponential distributions

A continuous random variable X is *exponential* with parameter $\lambda > 0$ if its cumulative distribution function (cdf) is of the form

$$F_X(d) \overset{\text{def}}{=} \mathbb{P}\{X \leq d\} = 1 - e^{-\lambda \cdot d} \quad \text{for} \quad d \geq 0 \quad .$$

The real number λ is called the *rate* of X and uniquely determines an exponential distribution. The expectation of X is $\frac{1}{\lambda}$, i.e., the average duration is $\frac{1}{\lambda}$ time units.

Some important properties of negative exponential distributions are the following. They possess the memory-less property, i.e.,

$$\mathbb{P}\{X > t + d \mid X > t\} \; = \; \mathbb{P}\{X > d\} \quad .$$

For exponential random variables X and Y with rate λ and μ, respectively, we have that the random variable $\mathsf{min}(X, Y)$ is exponentially distributed with rate $\lambda+\mu$. This is sometimes also referred to as the *race condition* of exponential distributions. An important property of such a "race" is that the probability that one of the random variables wins the race is just determined by the rates:

$$\mathbb{P}\{X = \mathsf{min}(X, Y)\} = \frac{\lambda}{\lambda+\mu} \quad .$$

2.2 Continuous-time Markov chains

A continuous-time Markov chain (CTMC) is a tuple $(S, \mathbf{R})$ where S is a countable set of states and $\mathbf{R}$ a *rate matrix* assigning non-negative rates to pairs of states. Roughly speaking, $\mathbf{R}(s, s') = \lambda > 0$ means that the average speed of going from state s to s' is $\frac{1}{\lambda}$. A CTMC is thus a transition system (with unlabeled transitions) where transitions are equipped with continuous probabilities. Let $E(s) = \sum_{s' \in S} \mathbf{R}(s, s')$ the exit rate of state s. State s is called absorbing whenever $E(s) = 0$. It should be noted that we allow self-loops, i.e., $\mathbf{R}(s, s) > 0$ is allowed. The presence of such transitions does not affect standard measures of CTMCs such as transient or steady-state distributions and allows for a natural definition of the operators in the temporal logic [2, 13].

We conclude this brief introduction by giving the intuitive interpretation of a CTMC. The probability that transition $s \rightarrow s'$ is *enabled* in the time interval $[0, d]$ is

$$1 - e^{-\mathbf{R}(s,s') \cdot d} \quad .$$

Due to the race condition between competing random variables, the probability to move from non-absorbing state s to s' in the interval $[0, d]$ is:

$$\frac{\mathbf{R}(s, s')}{E(s)} \cdot \left(1 - e^{-E(s)\cdot d}\right) \quad .$$

The first component indicates the discrete probability to move from s to s', while the second component characterizes the residence time in state s prior to the move; the probability to take an outgoing transition from s within $[0, d]$ is $1 - e^{-E(s)\cdot d}$.

3 StoKlaim

3.1 Syntax of StoKlaim

This section introduces the ingredients of Klaim and presents a simple extension thereof—inspired by Markovian process algebras—allowing basic activities in Klaim specifications to obey a random duration that is governed by an exponential distribution. The section is organized as follows. We first introduce the syntax of StoKlaim and its formal semantics yielding a labeled transition system. It will be shown how (action-labeled) CTMCs can be obtained from such transition systems. Finally, we give a comparison between our semantics and that of Klaim.

3.1.1 Syntactic categories

To start of with, we distinguish the following basic syntactic categories.

- $\mathcal{V}$, ranged over by $v, v', v_1, \ldots$, is a set of (basic data) *values*;
- $\mathcal{I}$, ranged over by $i, i', i_1, \ldots$, is a set of (physical) *addresses*;
- $\mathcal{L}$, ranged over by $l, l', l_1, \ldots$, is a set of *logical addresses*, also called *localities*; the locality $\mathsf{self} \in \mathcal{L}$;
- $\mathcal{R}$, ranged over by $r, r', r_1, \ldots$, is a set of *rate names*;
- $\mathcal{V}$-var, ranged over by $x, x', x_1, \ldots$, is a a set of *value variables*;
- $\mathcal{L}$-var, ranged over by $u, u', u_1, \ldots$, is set of *locality variables*;
- $\mathcal{P}$-var, ranged over by $X, X', X_1, \ldots$, is a set of *process variables.*

All these sets are countable and are mutually disjoint. Let ℓ, ℓ', ℓ_1 range over $\mathcal{L} \cup \mathcal{L}$-$\mathsf{var}$.

We adopt the $\vec{(\cdot)}$-notation for sequences; e.g., $\vec{l} = l_1, l_2, \ldots, l_n$ denotes a sequence over $\mathcal{L}$ and $\vec{x} = x_1, x_2, \ldots, x_m$ is a sequence over $\mathcal{V}$-var. For sequence $\vec{s} = s_1, \ldots, s_n$, let $\{\vec{s}\}$ denote the set of elements in $\vec{s}$, i.e., $\{\vec{s}\} = \{s_1, \ldots, s_n\}$. One-element sequences and singleton sets are denoted as the element they contain, i.e., $\{s\}$ is denoted as s and $\vec{s} = s'$ as s'.

3.1.2 Nets and processes

Specifications in StoKlaim consist of nets and processes. The most elementary net is the null net, denoted $\mathbf{0}$. A net consisting of a single node with *address* i is denoted $i ::_\rho E$ where ρ is an *allocation environment* and E is a *node element.* Allocation environment ρ maps localities occurring in the processes running at i to addresses. Nets may be composed of the parallel composition of several nodes. Thus, nets are constructed according to the following grammar:

$$\begin{array}{lcll} N & ::= & \mathbf{0} & \text{(null net)} \\ & | & i ::_\rho E & \text{(node)} \\ & | & N \parallel N & \text{(composition)} \end{array}$$

Node elements are either processes executing at a node—*process nodes* in the sequel—or data (represented as a tuple $\vec{f}$, see later on) that is stored at a node:

$$\begin{array}{lcll} E & ::= & P & \text{(running process)} \\ & \mid & \langle \vec{f} \rangle & \text{(stored tuple)} \end{array}$$

Processes are built up from the terminated process **nil**, a set of randomly delayed actions, and standard process algebraic constructors such as prefix, choice, parallel composition and process instantiation. Formally, for action A:

$$\begin{array}{lcll} P & ::= & \mathbf{nil} & \text{(null process)} \\ & \mid & (A, r).P & \text{(action prefix)} \\ & \mid & P + P & \text{(choice)} \\ & \mid & P \mid P & \text{(parallel composition)} \\ & \mid & X(\vec{X'}, \vec{\ell}, \vec{e}) & \text{(process instantiation)} \end{array}$$

where process X and all X' are assumed to be defined in the sequence of process definitions $\vec{D}$ by proper process defining equations of the form:

$$X(!\vec{X}, !\vec{u}, !\vec{x}) \triangleq P \quad .$$

Note that, for syntactical uniformity, *all* binding occurrences of variables are prefixed with '!'. This includes occurrences in node creation and read actions (see below), and formal parameters of process definitions.

The process $(A, r).P$ executes action A with a duration that is distributed exponentially with a rate specified by rate-name r. Rate-names are mapped to rate values by means of a (partial) function, denoted β, from $\mathcal{R}$ to $\mathbb{R}_{>0}$. Thus, the duration of the execution of action A is exponentially distributed with rate $(\beta\, r)$.

A few words are in order concerning rate names. As known from the field of probabilistic and Markovian process algebra, care must be taken to assign the right interpretation to terms of the form

$$(A, \lambda).\mathbf{nil} + (A, \lambda).\mathbf{nil} \quad .$$

A naive approach would yield a single transition labeled with rate λ leading to **nil**, while the race condition interpretation of choice operator requires a single transition labeled with rate $2 \times \lambda$. To avoid this problem several approaches have been proposed such as indexed or proved transitions [19, 27], recursive definitions of measures [29], and multi-sets [22]. The solution taken here is by using rate names, and enforcing that all rate names are unique. This easily combines with the structural congruence. Uniqueness of rate names is established in the inference rules of the operational semantics.

It is convenient to introduce the following notions. Let N be a net. The *site* (with address) i is the collection of nodes in N with address i. The set of processes *running at* site i is the set of processes P such that $i ::_\rho P'$ occurs in N and $P = P'$, or is a proper sub-process of P'. The set of processes (localities, or basic values, respectively) *stored at* site i is the set of processes (localities, or basic values resp.) occurring as fields of tuples $\vec{f}$ such that $i ::_\rho \langle \vec{f} \rangle$ is in N.

3.1.3 Actions

A process can write the value v in repository[2] l by the output action $\mathbf{out}(v)@l$. With an input action $\mathbf{in}(F_1, \ldots, F_n)@l$ a process can withdraw a datum that matches pattern, or *template* $(F_1, \ldots, F_n)$ from repository l. Processes can be written to/withdrawn from a repository as well. Action $\mathbf{read}(F_1, \ldots, F_n)@l$ is similar to $\mathbf{in}(F_1, \ldots, F_n)@l$ except that the datum at l is not deleted

[2]That is, the repository with address i, where i is the address which is bound to l by the allocation environment of the node where the process is running.

from the repository at l. The action $\mathbf{eval}(P)@l$ spawns process P at site l. A locality variable u can be used in place of l in all above actions. Action $\mathbf{newloc}(!u)$ creates a new node. The newly created locality (bound to the address of the newly created node) is referred to by variable u. Actions are built according to the following grammar:

$$\begin{array}{lll} A & ::= & \mathbf{out}(\vec{f})@\ell \quad \text{(output)} \\ & \mid & \mathbf{in}(\vec{F})@\ell \quad \text{(input)} \\ & \mid & \mathbf{read}(\vec{F})@\ell \quad \text{(read)} \\ & \mid & \mathbf{eval}(P)@\ell \quad \text{(process spawning)} \\ & \mid & \mathbf{newloc}(!u) \quad \text{(node creation)} \end{array}$$

3.1.4 Tuples and templates

Tuple fields can be processes, localities, locality variables and value expressions. For the sake of simplicity, process fields are restricted to process instantiations only, as in the case of process actual parameters of process instantiations. We assume a standard way for building value expressions from values, value variables and operators and do not discuss these in any further detail here. The grammar for tuple fields is:

$$\begin{array}{llll} f & ::= & X(\vec{X'}, \vec{\ell}, \vec{e}) & \text{(process)} \\ & \mid & \ell & \text{(locality or locality variable)} \\ & \mid & e & \text{(value expression)} \end{array}$$

Template fields can be tuple fields, or *binders*, which are variables prefixed with an exclamation mark. Binders indicate the binding occurrences of related variables; their *scope* will be defined in Section 3.2. Template fields obey the following syntax:

$$\begin{array}{llll} F & ::= & f & \text{(tuple field)} \\ & \mid & !X & \text{(process binder)} \\ & \mid & !u & \text{(locality binder)} \\ & \mid & !x & \text{(value binder)} \end{array}$$

3.1.5 StoKlaim specifications

A StoKlaim *specification* is a triple $(\beta, N, \vec{D})$ where $\beta : \mathcal{R} \to \mathbb{R}_{>0}$ is a rate-mapping, i.e., a function from rate-names to rates. N and $\vec{D}$ are, respectively, a net modeling the behaviour of a system and the process definitions for the processes used in N.

3.2 Semantics of StoKlaim

The semantics of StoKlaim is defined in a similar way as that of Klaim, and consists of a structural congruence on terms that allows the simplication of terms using some simple axioms, together with an operational semantics mapping terms onto—in our case— action-based Markov chains. First, we restrict ourselves to well-formed specifications by imposing some (straightforward) static semantics constraints. Notice that StoKlaim is a typed language; type-checking StoKlaim is out of the scope of the present paper, where type-correctness of StoKlaim specifications is assumed.

3.2.1 Well-formed specifications

Free and bound variables are defined in the usual way: in processes of the form $(\mathbf{newloc}(!u), r).P$, binder $!u$ binds all free occurrences of variable u in P. Similarly, in process $(\mathbf{in}(\vec{F})@\ell, r).P$ or $(\mathbf{read}(\vec{F})@\ell, r).P$ a binder occurring in $\vec{F}$ binds all free occurrences of the variable with the same name in P. In a process definition like $X(!\vec{X'}, !\vec{u}, !\vec{x}) \triangleq P$, a binder occurring in the formal parameter list $!\vec{X'}, !\vec{u}, !\vec{x}$ binds all free occurrences of the variable with the same name in P. In these cases, P is called the *scope* of the binder at hand.

Let $(\mathsf{dom}\,\beta)$ be the domain of β, $(\mathsf{dom}\,\rho)$ the domain of allocation environment ρ, $(\mathsf{rng}\,\rho)$ is the *range* of ρ, $(\mathsf{Loc}\,N)$, $(\mathsf{Adr}\,N)$ and $(\mathsf{Rat}\,N)$ the set of localities, addresses and rate names, respectively, occurring in N.

Definition 3.1 A StoKlaim *specification* $(\beta, N, \vec{D})$ is *well-formed* if and only if it is type-correct and:

- all rate-names occurring in N or $\vec{D}$ are distinct.
- $(\mathsf{dom}\,\beta) = (\mathsf{Rat}\,N)$
- each allocation environment ρ in N satisfies:
  - **(i)** $\mathsf{self} \in (\mathsf{dom}\,\rho)$;
  - **(ii)** $(\mathsf{dom}\,\rho) \setminus \{\mathsf{self}\} \subseteq (\mathsf{Loc}\,N)$
  - **(iii)** $(\mathsf{rng}\,\rho) \subseteq (\mathsf{Adr}\,N)$
  - **(iv)** for all nodes $i_1 ::_{\rho_1} E_1$ and $i_2 ::_{\rho_2} E_2$, if $i_1 = i_2$ then ρ_1 and ρ_2 are *compatible*, i.e., for all l in $(\mathsf{dom}\,\rho_1) \cap (\mathsf{dom}\,\rho_2)$ we have $\rho_1\, l = \rho_2\, l$.
- The only free variables occurring in N are process variables defined in $\vec{D}$.
- All process variables used in the left-hand-side of defining equations in $\vec{D}$ are distinct. In every defining equation $X(!\vec{X}', !\vec{u}, !\vec{x}) \triangleq P$ in $\vec{D}$ all binders occurring in the formal parameter list $!\vec{X}', !\vec{u}, !\vec{x}$ are distinct and all free (process) variables occurring in P, which are not bound by the binders in $!\vec{X}', !\vec{u}, !\vec{x}$, are defined in some defining equation in $\vec{D}$. Finally, for each process formal parameter $!X'$ there is at most *one* free occurrence of X' in P.
- All processes instantiations $X(\vec{X}', \vec{\ell}, \vec{e})$ are *guarded*, i.e. they occur in the context of an action prefix $(A, r).X(\vec{X}', \vec{\ell}, \vec{e})$, for some A and r.
- In processes of the form $(\mathbf{in}(\vec{F})@\ell, r).P$ or $(\mathbf{read}(\vec{F})@\ell, r).P$, all binders occurring in $\vec{F}$ are distinct; moreover for each process binder $!X$ occurring in $\vec{F}$ there is at most one free occurrence of X in P.

$\diamond$

In the remainder of this paper we assume specifications to be *well-formed*.

3.2.2 Structural congruence

StoKlaim specifications will be mapped by the operational semantics definition onto labeled transition systems, in our case action-labeled Markov chains. The states of these structures are called *configurations*, i.e., tuples (I, L, β, N), often denoted as $I, L, \beta, \vdash N$, where $I \subseteq \mathcal{I}$ and $L \subseteq \mathcal{L}$ are the set of addresses and localities, respectively, in the net N, and β is the rate-mapping. Configurations are considered modulo the *structural congruence* relation defined in Table 1. (For the sake of simplicity we have omitted the components I, L and β when they are unaffected.) Compared to the structural congruence laws of Klaim, the laws (CO+), (AS+), and (NE+), and (REN) have been added. Law (REN) states that the rate names occurring in N can be replaced by means of a rate-name substitution θ. We adopt the usual notation for syntactical substitution, namely $N\theta$, where θ is a total function in $\mathcal{R} \to \mathcal{R}$. It is required that (i) the substitution does not interfere with the current rate-mapping (i.e., $(\mathsf{dom}\,\beta) \cap (\mathsf{rng}\,\theta) = \varnothing$), (ii) uniqueness of rate-names is preserved in $N\theta$ (i.e., θ is injective), and (iii) the rate-mapping in the configuration resulting from applying the substitution is defined for the new names and gives the same rates as for the old ones (i.e. the new mapping is the composition of the old mapping and the inverse of θ). Finally,

(NE$\parallel$)	$N \equiv N \parallel \mathbf{0}$
(CO$\parallel$)	$N_1 \parallel N_2 \equiv N_2 \parallel N_1$
(AS$\parallel$)	$N_1 \parallel (N_2 \parallel N_3) \equiv (N_1 \parallel N_2) \parallel N_3$
(NE+)	$i ::_\rho P \equiv i ::_\rho P + \mathbf{nil}$
(CO+)	$i ::_\rho P_1 + P_2 \equiv i ::_\rho P_2 + P_1$
(AS+)	$i ::_\rho P_1 + (P_2 + P_3) \equiv i ::_\rho (P_1 + P_2) + P_3$
(NE$\mid$)	$i ::_\rho P \equiv i ::_\rho P \mid \mathbf{nil}$
(CLO)	$i ::_{\rho_1 \bowtie \rho_2} P_1 \mid P_2 \equiv i ::_{\rho_1} P_1 \parallel i ::_{\rho_2} P_2$ if ρ_1 and ρ_2 are compatible
(REN)	$\beta \vdash N \equiv \beta \circ \theta^{-1} \vdash N\theta$ for any injection $\theta : \mathcal{R} \to \mathcal{R}$ with $(\mathsf{dom}\,\theta) = (\mathsf{Rat}\,N)$ and $(\mathsf{rng}\,\theta) \cap (\mathsf{dom}\,\beta) = \varnothing$

Table 1: Structural congruence laws

$[\![P]\!]_\rho$	$\stackrel{\text{def}}{=} P\{\rho\}$	$[\![!X]\!]_\rho$	$\stackrel{\text{def}}{=} !X$
$[\![\ell]\!]_\rho$	$\stackrel{\text{def}}{=} \ell$	$[\![!u]\!]_\rho$	$\stackrel{\text{def}}{=} !u$
$[\![e]\!]_\rho$	$\stackrel{\text{def}}{=} \mathcal{E}[\![e]\!]$	$[\![!x]\!]_\rho$	$\stackrel{\text{def}}{=} !x$
$[\![(F_1, \ldots, F_n)]\!]_\rho$	$\stackrel{\text{def}}{=} ([\![F_1]\!]_\rho, \ldots, [\![F_n]\!]_\rho)$		

Table 2: Tuple evaluation

in law (CLO) the allocation environments must be taken care of; in particular, ρ_1 and ρ_2 must be *compatible* (see Def. 3.1), in which case, allocation environment $\rho_1 \bowtie \rho_2$ is defined as follows:

$$(\rho_1 \bowtie \rho_2)\, l \stackrel{\text{def}}{=} \begin{cases} \rho_1\, l, & \text{if } l \in \mathsf{dom}\,\rho_1 \\ \rho_2\, l, & \text{if } l \in \mathsf{dom}\,\rho_2 \end{cases}$$

3.2.3 Tuple evaluation

Function $[\![\cdot]\!]_\cdot$ (cf. Table 2) evaluates tuples and templates. Notice that $[\![u]\!]_\rho$ yields u. In practice, the static semantics constraints together with the semantics of the **in** and **newloc** actions as well as process instantiation guarantee that variables are properly replaced by their values whenever necessary[3]. In Table 2 function $\mathcal{E}[\![\cdot]\!]$ is used for evaluating value expressions e. The definition of $\mathcal{E}[\![\cdot]\!]$ is outside the scope of the present paper.

$P\{\rho\}$ denotes a process *closure*, i.e., a pair consisting of a process and an allocation environment. $P\{\rho\}$ behaves like process P except that any locality l in P denotes the physical address $(\rho\, l)$ if $l \in (\mathsf{dom}\,\rho)$, and is resolved with the current allocation environment otherwise. Closures are not part of the language, but are only used in the operational semantics (definition); they may occur at any place where a process is allowed.

[3]Thus, there is no need for explicitly evaluating variables by $[\![\cdot]\!]_\cdot$.

$\mathit{rename}(\mathbf{nil}, \beta)$	$\overset{\text{def}}{=}$	$(\mathbf{nil}, \beta)$
$\mathit{rename}((A, r).P, \beta)$	$\overset{\text{def}}{=}$	$((A, r').P', \beta')$ where $r' \in \mathcal{R} \setminus (\mathsf{dom}\,\beta)$ and $(P', \beta') = \mathit{rename}(P, \beta[(\beta\, r)/r'])$
$\mathit{rename}(P_1\ \mathit{op}\ P_2, \beta)$	$\overset{\text{def}}{=}$	$(P_1'\ \mathit{op}\ P_2', \beta')$, $\mathit{op} \in \{+, \vert\}$ where $(P_1', \beta'') = \mathit{rename}(P_1, \beta)$ and $(P_2', \beta') = \mathit{rename}(P_2, \beta'')$
$\mathit{rename}(X(\vec{P}, \vec{\ell}, \vec{e}), \beta)$	$\overset{\text{def}}{=}$	$(X(\vec{P}, \vec{\ell}, \vec{e}), \beta)$
$\mathit{rename}(P\{\rho\}, \beta)$	$\overset{\text{def}}{=}$	$(P'\{\rho\}, \beta')$ where $(P', \beta') = \mathit{rename}(P, \beta)$
$\mathit{rename}(\ell, \beta)$	$\overset{\text{def}}{=}$	(ℓ, β)
$\mathit{rename}(e, \beta)$	$\overset{\text{def}}{=}$	(e, β)
$\mathit{rename}((f_1, \ldots, f_n), \beta)$	$\overset{\text{def}}{=}$	$((f_1', \ldots, f_n'), \beta')$ where $(f_1', \beta_1') = \mathit{rename}(f_1, \beta)$ and $(f_j', \beta_j') = \mathit{rename}(f_j, \beta_{j-1}')$ for $1 < j \leq n$ with $\beta' = \beta_n'$

Table 3: Rate name renaming

3.2.4 Rate name renaming

Function *rename* (cf. Table 3) takes as argument a process P and a rate mapping β. It renames all rate-names occurring in P into fresh names and adapts β accordingly. *rename* will be used in defining the semantics of the actions **out** and **eval** and in process instantiation to guarantee unique rate-names. It is not difficult to establish that *rename* indeed establishes unique rate names. Notice also that, strictly speaking, Table 3 characterizes a set of functions, each specific one being defined by the particular choices performed in the second equation.

3.2.5 Substitutions

In the inference rules defined below we exploit substitutions and combinations thereof. They have the usual meaning, i.e., for $d_1, \ldots, d_n$ ranging over $\mathcal{L} \cup \mathcal{V} \cup P$, and $w_1, \ldots, w_n$ ranging over $\mathcal{L}\text{-}\mathsf{var} \cup \mathcal{V}\text{-}\mathsf{var} \cup \mathcal{P}\text{-}\mathsf{var}$, we let $[d_1/w_1 \ldots d_n/w_n]$, with $w_i \neq w_j$ for $i \neq j$, denote the substitution which replaces w_j by d_j for $0 < j \leq n$. Let $[]$ denote the empty substitution and, w.l.o.g, for substitution Θ_1:

$$[d_1/w_1, \ldots, d_n/w_n, d_1'/w_1', \ldots, d_m'/w_m']$$

and substitution Θ_2:

$$[d_1''/w_1', .., d_m''/w_m', d_{m+1}''/w_{m+1}', .., d_{m+h}''/w_{m+h}']$$

with $\{w_{m+1}', \ldots, w_{m+h}'\} \cap \{w_1, \ldots, w_n\} = \varnothing$, let $\Theta_1 \triangleleft \Theta_2$ be the substitution:

$$[d_1/w_1, \ldots, d_n/w_n, d_1''/w_1', \ldots, d_m''/w_m', d_{m+1}''/w_{m+1}', \ldots, d_{m+h}''/w_{m+h}'] \quad .$$

3.2.6 Matching

Pattern-matching of templates with (stored) tuples is used to define the semantics of input and read-actions. In essence, this goes along similar lines as for Klaim (and is rather standard) except that tuple fields can be processes and may include rate names. A matching is successful when—in addition to the usual matching criteria—rate names indicate the same rate. Function *match* (cf. Table 4) yields a substitution if a matching is successful. Here, it is assumed that β is the rate mapping for which the matching is considered. (Strictly speaking, β is a parameter of *match*, but as it is unchanged in all cases, this is left implicit for the sake of readability.) In the definition of *match*, processes are considered the same as closures with an empty allocation environment.

$$match(l,l) \stackrel{\text{def}}{=} [] \qquad match(v,v) \stackrel{\text{def}}{=} [] \qquad match(\mathbf{nil},\mathbf{nil}) \stackrel{\text{def}}{=} []$$

$$match(!X, P\{\rho\}) \stackrel{\text{def}}{=} [P\{\rho\}/X] \quad match(!u,l) \stackrel{\text{def}}{=} [l/u] \quad match(!x,v) \stackrel{\text{def}}{=} [v/x]$$

$$\frac{match(P,P') = [] \qquad (\beta\, r) = (\beta\, r')}{match((A,r).P,(A,r').P') \stackrel{\text{def}}{=} []} \qquad \frac{match(P_1,P_1') = [] \qquad match(P_2,P_2') = []}{match(P_1+P_2,P_1'+P_2') \stackrel{\text{def}}{=} []}$$

$$\frac{match(\vec{P},\vec{P'}) = [] \qquad match(\vec{l},\vec{l'}) = [] \qquad match(\vec{v},\vec{v'}) = []}{match(X(\vec{P},\vec{l},\vec{v}),X(\vec{P'},\vec{l'},\vec{v'})) \stackrel{\text{def}}{=} []} \qquad \frac{match(P,P') = []}{match(P\{\rho\},P'\{\rho\}) \stackrel{\text{def}}{=} []}$$

$$\frac{match(F_1,f_1') = \Theta_1 \qquad \ldots \qquad match(F_n,f_n') = \Theta_n}{match((F_1,\ldots,F_n),(f_1',\ldots,f_n')) \stackrel{\text{def}}{=} \Theta_1 \triangleleft \ldots \triangleleft \Theta_n}$$

Table 4: Pattern-matching of tuples against templates

3.2.7 Labeled transition system semantics

Given the auxiliary definitions so far, we are now in a position to define the operational semantics of a StoKlaim specification $(\beta, N, \vec{D})$ that consists of network N, rate mapping β and process definitions in $\vec{D}$. The semantics is defined in terms of a labeled transition system where states are in fact configurations modulo structural equivalence. Let *Conf*, ranged over by $c, c', c_1, \ldots$, be the set of all representatives of the equivalence classes induced by the structural congruence $\equiv$. We abstract here from the way in which such representatives are chosen, except that we require that at most one process node per physical address can occur in the network component of any such a representative[4].

The *derivatives* (*Der* c) of configuration $c \in Conf$ is the smallest set such that it includes c, and is closed under the inference rules defined in Tables 5 and 1. The labeled transition system of StoKlaim specification $(\beta, N, \vec{D})$ is the quadruple $(C, \Lambda, \longrightarrow, c_0)$ with:

- $c_0 \in Conf$, the initial state, is (the standard representative of) $(\mathsf{Adr}\, N), (\mathsf{Loc}\, N), \beta \vdash N$;
- $C \stackrel{\text{def}}{=} (Der\, c_0)$ is the set of states;
- $\Lambda \subseteq (\mathcal{I} \times \mathcal{A}) \times \mathcal{R}$ is the label-set; and
- $\longrightarrow$, the transition relation, is the smallest relation on $C \times \Lambda \times C$ induced by the rules of Table 5 and the laws of Table 1.

Here, $\mathcal{A}$ is the set of *ground* actions constructed according to the grammar:

$$\mathbf{o}(\vec{\mathcal{F}},\mathcal{I}) \mid \mathbf{i}(\vec{\mathcal{F}},\mathcal{I}) \mid \mathbf{r}(\vec{\mathcal{F}},\mathcal{I}) \mid \mathbf{e}(P,\mathcal{I}) \mid \mathbf{n}(\mathcal{I})$$

[4]This can be easily achieved by means of law (CLO) and has the consequence that all processes running at a given site are in fact represented as (parallel components of) a single process node.

for an output, input, read, eval, and newloc action, respectively. The tuple parameters $\mathcal{F}$ are defined as follows:

$$\mathcal{F} ::= P \mid l \mid v$$

We let $\gamma, \gamma', \gamma_1, \ldots$ range over action-labels and denote $(c, (\gamma, r), c') \in \longrightarrow$ by $c \xrightarrow{\gamma, r} c'$. Finally, we let N_c (β_c, resp.) denote the network (rate maping, resp.) component of c.

It remains to explain the rules in Table 5 where for simplicity we have omitted the components I, L and β from a configuration whenever they are unchanged. We discuss the rules one by one.

3.2.8 Process and data distribution

Rule (OUT) models the dispatching of a tuple at an existing (possibly remote) site. The basic format of the rule is as follows:

$$\frac{\rho_1\, l = i_2}{i_1 :: (\mathbf{out}(\vec{f})@l, r).P \parallel i_2 :: E \longrightarrow i_1 :: P \parallel i_2 :: E \parallel i_2 :: \langle \vec{f}' \rangle}$$

where ρ_j, i.e. the allocation environment for i_j ($j = 1, 2$) is not shown in the conclusion of rule above, for simplicity. For action $\mathbf{out}(\vec{f})@l$, the tuple $\vec{f}$ first needs to be evaluated, yielding $\vec{f}'$. To do so, we may either adopt a static or a dynamic scoping rule. According to the former, the allocation environment ρ_1 of the source site is used in the evaluation of $\vec{f}$; for the latter, ρ_2 of the destination site is used instead. In Section 4, the latter option will be adopted for the **out** action.[5] Prior to evaluation, the tuple needs to undergo (rate) renaming— as it may contain processes, and thus rate names—in order to guarantee uniqueness of rate names. Rule (OUT) in Table 5 contains the full details for the static approach.

3.2.9 Code on demand

Rule (IN) defines the operation of retrieving a tuple from an existing (possibly remote) site. In its most rudimentary form, the inference rule roughly reads as follows:

$$\frac{\rho_1\, l = i_2}{i_1 :: (\mathbf{in}(\vec{F})@l, r).P \parallel i_2 :: \langle \vec{f} \rangle \longrightarrow i_1 :: P\Theta \parallel i_2 :: \mathbf{nil}}$$

where—as before—the allocation environment ρ_j for i_j ($j = 1, 2$) is not indicated explicitly and Θ is the substitution resulting from the matching of $\vec{f}$ with the template $\vec{F}$. All free variables occurring in P which occur as binders in $\vec{F}$ are replaced by proper values according to Θ. Action $\mathbf{in}(\vec{F})@l$ is blocking in the sense that it can only be performed if a tuple $\vec{f}$ is present at site i_2 that matches template $[\![\vec{F}]\!]_{\rho_1}$. The inference rule (RD) for reading a tuple is similar except that the matched tuple is not removed from the repository at l.

3.2.10 Remote evaluation

Rule (EVL) defines the spawning of process P' to an existing (possibly remote) site. In its most rudimentary form, the inference rule roughly reads:

$$\frac{\rho_1\, l = i_2}{i_1 :: (\mathbf{eval}(P')@l, r).P \parallel i_2 :: E \longrightarrow i_1 :: P \parallel i_2 :: E \parallel i_2 :: P'}$$

where for the allocation environments we adopt the convention as before. In order to establish uniqueness of rate names, process P' has to be renamed prior to the spawning at l. The rate mapping of the resulting net has to be adapted accordingly. The resulting refined inference rule is listed in Table 5. Note that a dynamic scoping rule has been adopted (as in [12]); a static rule is obtained if P' is closed using ρ_1 before spawning.

[5]Note that the implementation of the dynamic scoping rule by means of **eval** as suggested e.g., in [12] (page 321) cannot be used here since sequentialization of several actions has a non-trivial impact on the timing aspects of the modeled behaviour.

(OUT) $$\frac{\rho_1\, l = i_2}{\beta \vdash i_1 ::_{\rho_1} (\mathbf{out}(\vec{f})@l, r).P \parallel i_2 ::_{\rho_2} E \xrightarrow{(i_1, \mathbf{o}(\vec{f''}, i_2)), r} \beta' \vdash i_1 ::_{\rho_1} P \parallel i_2 ::_{\rho_2} E \parallel i_2 ::_{\rho_2} \langle \vec{f''} \rangle}$$

where $(\vec{f'}, \beta') = \mathit{rename}(\vec{f}, \beta)$ and $\vec{f''} = [\![\vec{f'}]\!]_{\rho_1}$

(IN) $$\frac{\rho_1\, l = i_2}{i_1 ::_{\rho_1} (\mathbf{in}(\vec{F})@l, r).P \parallel i_2 ::_{\rho_2} \langle \vec{f} \rangle \xrightarrow{(i_1, \mathbf{i}(\vec{f}, i_2)), r} i_1 ::_{\rho_1} P\Theta \parallel i_2 ::_{\rho_2} \mathbf{nil}}$$

where $\mathit{match}([\![\vec{F}]\!]_{\rho_1}, \vec{f}) = \Theta$

(RD) $$\frac{\rho_1\, l = i_2}{i_1 ::_{\rho_1} (\mathbf{read}(\vec{F})@l, r).P \parallel i_2 ::_{\rho_2} \langle \vec{f} \rangle \xrightarrow{(i_1, \mathbf{r}(\vec{f}, i_2)), r} i_1 ::_{\rho_1} P\Theta \parallel i_2 ::_{\rho_2} \langle \vec{f} \rangle}$$

where $\mathit{match}([\![\vec{F}]\!]_{\rho_1}, \vec{f}) = \Theta$

(EVL) $$\frac{\rho_1\, l = i_2}{\beta \vdash i_1 ::_{\rho_1} (\mathbf{eval}(P')@l, r).P \parallel i_2 ::_{\rho_2} E \xrightarrow{(i_1, \mathbf{e}(P'', i_2)), r} \beta' \vdash i_1 ::_{\rho_1} P \parallel i_2 ::_{\rho_2} E \parallel i_2 ::_{\rho_2} P''}$$

where $(P'', \beta') = \mathit{rename}(P', \beta)$

(NLC) $$\frac{i_2 \in \mathcal{I} \setminus I \qquad l_2 \in \mathcal{L} \setminus L}{I, L \vdash i_1 ::_{\rho_1} (\mathbf{newloc}(!u), r).P \xrightarrow{(i_1, \mathbf{n}(i_2)), r} I', L' \vdash i_1 ::_{\rho'_1} P[l_2/u] \parallel i_2 ::_{\rho_2} \mathbf{nil}}$$

where $I' = I \cup \{i_2\}$, $L' = L \cup \{l_2\}$, $\rho'_1 = \rho_1 \bullet [l_2 \mapsto i_2]$ and $\rho_2 = \rho'_1 \bullet [\mathsf{self} \mapsto i_2]$

(CLS) $$\frac{I, L, \beta \vdash i_1 ::_{\rho_1 \bullet \rho_2} P \parallel N \xrightarrow{\gamma, r} I', L', \beta' \vdash i_1 ::_{\rho'_1 \bullet \rho_2} P' \parallel N'}{I, L, \beta \vdash i_1 ::_{\rho_1} P\{\rho_2\} \parallel N \xrightarrow{\gamma, r} I', L', \beta' \vdash i_1 ::_{\rho'_1} P'\{\rho_2\} \parallel N'}$$

(PIN) $$\frac{X(!\vec{X'}, !\vec{u}, !\vec{x}) \stackrel{\Delta}{=} P \qquad I, L, \beta \vdash i ::_\rho P' \parallel N \xrightarrow{\gamma, r} I', L', \beta' \vdash N'}{I, L, \beta \vdash i ::_\rho X(\vec{P''}, \vec{l}, \vec{v}) \parallel N \xrightarrow{\gamma, r} I', L', \beta' \vdash N'}$$

where $(P', \beta') = \mathit{rename}(P[\vec{P''}/\vec{X'}, \vec{l}/\vec{u}, \vec{v}/\vec{x}], \beta)$

(CHO) $$\frac{I, L, \beta \vdash i ::_\rho P \parallel N \xrightarrow{\gamma, r} I', L', \beta' \vdash N'}{I, L, \beta \vdash i ::_\rho P \,+\, P' \parallel N \xrightarrow{\gamma, r} I', L', \beta' \vdash N'}$$

(PAR) $$\frac{I, L, \beta \vdash N_1 \xrightarrow{\gamma, r} I', L', \beta' \vdash N'}{I, L, \beta \vdash N_1 \parallel N_2 \xrightarrow{\gamma, r} I', L', \beta' \vdash N' \,\parallel\, N_2}$$

(STC) $$\frac{\beta \vdash N \equiv \beta \circ \theta^{-1} \vdash N_1 \qquad I, L, \beta \circ \theta^{-1} \vdash N_1 \xrightarrow{\gamma, (\theta\, r)} I', L', \beta_2 \vdash N_2 \qquad \beta_2 \vdash N_2 \equiv \beta' \vdash N'}{I, L, \beta \vdash N \xrightarrow{\gamma, r} I', L', \beta' \vdash N'}$$

Table 5: Reduction rules for StoKlaim

3.2.11 Site creation

Rule (NLC) models the creation of a new node, with fresh address i_2 and fresh locality l_2. In its most rudimentary form, the inference rule roughly reads:

$$\frac{i_2 \in \mathcal{I} \setminus I \qquad l_2 \in \mathcal{L} \setminus L}{i_1 ::_{\rho_1} (\mathbf{newloc}(!u), r).P \rightarrow i_1 ::_{\rho'_1} P[l_2/u] \parallel i_2 ::_{\rho_2} \mathbf{nil}}$$

As expected, all free occurrences of u in P are replaced by l_2. Accordingly, the allocation environment in the nodes of the resulting net need to be adjusted. The new allocation environment ρ'_1 extends the current environment ρ_1 with mapping l_2 to i_2. Formally: $\rho'_1 = \rho_1 \bullet [l_2 \mapsto i_2]$. Environment ρ_2 equals ρ'_1 except that self is bounded to i_2 instead of i_1, i.e., $\rho_2 = \rho'_1 \bullet [\mathsf{self} \mapsto i_2]$. The operators $[\cdot \mapsto \cdot]$ and $\bullet$ are formally defined as follows. For $d_1 \in D_1$ and $d_2 \in D_2$, $[d_1 \mapsto d_2] : D_1 \rightarrow D_2$ is defined as:

$$[d_1 \mapsto d_2]\, d \stackrel{\text{def}}{=} \begin{cases} d_2, & \text{if } d = d_1 \\ \text{undefined} & \text{otherwise} \end{cases}$$

and $\bullet : (D_1 \rightarrow D_2) \times (D_1 \rightarrow D_2) \rightarrow (D_1 \rightarrow D_2)$:

$$(g_1 \bullet g_2)\, d \stackrel{\text{def}}{=} \begin{cases} g_2\, d, & \text{if } d \in (\mathsf{dom}\, g_2) \\ g_1\, d & \text{otherwise} \end{cases} .$$

The complete inference rule is listed as rule (NLC) in Table 5.

3.2.12 Process closure

Rule (CLS) asserts that closure $P\{\rho_2\}$, under allocation environment ρ_1 behaves the same as P under allocation environment $\rho_1 \bullet \rho_2$. In practice this means that for any locality occurring in P, first an attempt is made to resolve the locality using ρ_2 and, if this fails, the locality is resolved using ρ_1.

3.2.13 Process instantiation, choice and concurrency

The inference rules for these operators are completely standard, except that for process instantiation a renaming needs to be performed.

3.2.14 Structural congruence

The final reduction rule concerns the replacement of nets by congruent ones. This is formalized by *rule (STC)*. It states that if $N \equiv N_1$ and N_1 may evolve into N_2, with $N_2 \equiv N'$, then N may evolve to N' performing "the same" transition. In practice, since structural congruence may imply rate renaming, the transitions $N_1 \rightarrow N_2$ and $N \rightarrow N'$ may differ in their rate-name, via the (inverse of) the same rate-name substitution involved in the structural congruence (see law (REN) in Table 1).

3.3 Differences with Klaim semantics

A major difference between the version of KLAIM used here and, e.g., that in [12], is that here processes are completely denied *direct* access to physical addresses. They can refer to them only by means of logical addresses (and allocation environments). In [12], localities are kept only as long as they are not evaluated, which occurs when they are either communicated—by means of storing them into a tuple space—or used for pattern-matching. In fact, the semantics of the **out** (**in**, respectively) action requires its argument to be evaluated before being stored in the specified locality (using it in pattern-matching, respectively). On the other hand, the evaluation function applied to a locality yields the physical address bound to the locality by the current allocation environment. As a result, processes *cannot* receive localities but only physical addresses and, consequently, they are forced to use *only* physical addresses, unless localities are constants. In practice, this means that there is no way for a process to send a locality to another process (e.g., the *logical* name of a service) and, in general, benefits of relative addressing are partially lost. The same considerations apply to **newloc**$(!u).P$: in [12], the physical address of the newly created node is used for directly replacing the occurrences of u in P.

3.4 Obtaining a CTMC

As a final step in the semantics, we present the details of obtaining a CTMC, in fact an action-labeled CTMC, from a StoKlaim specification. Given the mapping presented so far onto labeled transition systems, where transition labels are pairs of (ground) actions and rate names, this last step is rather straightforward: basically rate names need to be turned into rates. This entails that whenever $c \xrightarrow{\gamma,r} c'$ and $c \xrightarrow{\gamma,r'} c'$, a single γ-labeled transition from configuration c to c' should be obtained with rate $(\beta\, r) + (\beta r')$.

An *action-labelled* CTMC (AMC) $\mathcal{M}$ is a triple $(S, ACT, \longmapsto)$ where S is a set of states, ACT is a set of actions, and $\longmapsto\ \subseteq S \times (ACT \times \mathbb{R}_{>0}) \times S$ is the transition relation. Transition $s \overset{\gamma,\lambda}{\longmapsto} s'$ means that the AMC may evolve from state s to s' while performing action γ with an execution time determined by an exponential distribution with rate λ. The total rate for going from s to s' by performing action γ is defined by:

$$\mathbf{R}_\gamma(s, s') \overset{\text{def}}{=} \sum_{s \overset{\gamma,\lambda}{\longmapsto} s'} \lambda \quad .$$

Assume that the labeled transition system associated to the StoKlaim specification finite, i.e., it is finitely branching and has a finite number of states[6].

For net specification $(\beta, N, \vec{D})$ with finite transition system $(C, \Lambda, \longrightarrow, c_0)$, let $\mathsf{AMC}(\beta, N, \vec{D}) = (S, ACT, \longmapsto)$ with:

- $S = C$
- $ACT \subseteq \mathcal{I} \times \mathcal{A}$, and
- $c \overset{\gamma,\lambda}{\longmapsto} c'$ if and only if $0 < \lambda = \sum_{c \xrightarrow{\gamma,r} c'} (\beta_c\, r) \quad .$

4 Modeling and analysis of the spreading of a virus

In this section we show how StoKlaim can be used for modeling the *spreading* of a virus in a network. The example we present has been inspired by a similar one in [16].

We model a network as a set of sites and the virus running on a site can move arbitrarily from the current site to others, infecting them. At each site, an instance of the operating system runs, which, upon receiving the virus, can either run it or suppress it. In this paper, for the sake of simplicity, we consider simple networks which are in fact grids of $n \times m$ sites. Each site is connected with its four neighbors ($north, south, east, west$), except for border sites, which lack some connections in the obvious way (e.g. the sites on the east border have no east connection). Moreover, we assume that the virus can move only to *one* adjacent site. Finally, we refrain from modeling aspects of the virus other than the way it replicates in the network. In particular we do not consider the local effects of the virus and we make the virus die as soon as it has infected one of the neighbors of its site. Similarly, we abstract from all details of the operating system, except those directly dealing with the virus.

The process definitions for the virus V and the operating system running at each node are given in Fig. 1. For the sake of notational simplicity, a matrix-like notation is used for the (definitions of the) operating system (processes) O_{jk}, with $(1 \leq j \leq n, 1 \leq k \leq m)$. Once activated, at site

[6]There are several ways for assuring finiteness of transition systems obtained from process algebras; see, e.g., [17]. We will not dwell further upon this issue here.

$$
\begin{array}{lll}
V & \stackrel{\Delta}{=} & ((\mathbf{out}(V)@north, rn).\mathbf{nil}) + \\
 & & ((\mathbf{out}(V)@south, rs).\mathbf{nil}) + \\
 & & ((\mathbf{out}(V)@east, re).\mathbf{nil}) + \\
 & & ((\mathbf{out}(V)@west, rw).\mathbf{nil}) \\
 & & \\
O_{jk} & \stackrel{\Delta}{=} & ((\mathbf{in}(!X)@\mathsf{self}, u_{jk}).(\mathbf{eval}(X)@\mathsf{self}, r_{jk}).O_{jk}) + \\
 & & ((\mathbf{in}(!X)@\mathsf{self}, d_{jk}).O_{jk})
\end{array}
$$

Figure 1: Specification of an infected network

(with address) i_{jk}, the virus sends a copy of itself to either of the neighbors of its site, the choice being nondeterministic. The allocation environment ρ_{jk} of site i_{jk} is the expected one: for a grid of $n \times m$ sites we have the following definition for ρ_{jk}:

$$
\rho_{jk} l = \begin{cases} i_{j-1,k} & \text{if } l = north \text{ and } j > 1 \\ i_{j+1,k} & \text{if } l = south \text{ and } j < n \\ i_{j,k+1} & \text{if } l = east \text{ and } k < m \\ i_{j,k-1} & \text{if } l = west \text{ and } k > 1 \end{cases}
$$

Notice that for sites 'at the boarder' (like i_{12}) the allocation environment is defined only for those directions which don't point 'outside the grid' itself (for instance ρ_{12} is not defined on *north*). Consequently, when the virus is running on such a site, those alternatives which would make it try to send its own code 'outside the grid' will simply be discarded, as no reduction rule is applicable. The output rate toward *north* ($south, east, west$, resp.) is established by rn (rs, re, rw, resp.). Notice that the *dynamic* scoping rule for the output action (see remarks on page 11) is used for this example: when the virus running at site i_{jk} uploads a copy of its code to a different site $i_{j'k'}$, the localities of such a copy, when in execution, will be resolved using the allocation environment of the *remote* site $i_{j'k'}$. The definition of O_{jk} postulates that O_{jk} may receive the virus and execute it, with probability given by $\frac{u_{jk}}{u_{jk}+d_{jk}}$, or detect and discard it after receiving it, of course with probability $\frac{d_{jk}}{u_{jk}+d_{jk}}$. The rate of launching the execution of the virus is given by r_{jk}. For a system with nine sites, i.e. for $n = m = 3$, the network component N_0 of the initial configuration is the following:

$$
i_{11} ::_{\rho_{11}} \langle V \rangle \parallel (\big\|_{\substack{1 \leq j \leq 3 \\ 1 \leq k \leq 3}} i_{jk} ::_{\rho_{jk}} O_{jk})
$$

A graphical representation of N_0 is given in Fig 2. Site i_{jk} is represented by a square box annotated with i_{jk}; the data (running processes, resp.) of the site are displayed in the inner upper (inner lower) part of the box while each locality l on which ρ_{jk} is defined is used as the label of an arrow pointing to the site (with physical address) $\rho_{jk}\, l$. The locality (address, resp.) set ($\mathsf{Loc}\, N_0$) (($\mathsf{Adr}\, N_0$), resp.) corresponding to N_0 is $\{north, south, east, west\}$ ($\{i_{j,k} \mid 1 \leq j \leq 3, 1 \leq k \leq 3\}$, resp.). Let us assume that the operating systems of the sites on the diagonal from bottom-left to top-right—O_{31}, O_{22}, and O_{13}—have a relatively high rate of detection and can be considered as a firewall to protect the sites i_{32}, i_{33}, and i_{23}. For instance, let β_0 be defined as follows: $\beta_0\, rn = \beta_0\, rs = \beta_0\, re = \beta_0\, rw = 2$, $\beta_0\, r_{jk} = 2$ for $1 \leq j,k \leq 3$, $\beta_0\, d_{31} = \beta_0\, d_{22} = \beta_0\, d_{13} = 10$, and $\beta_0\, d_{ij} = 1$ otherwise, $\beta_0\, u_{31} = \beta_0\, u_{22} = \beta_0\, u_{13} = 1$, and $\beta_0\, u_{jk} = 10$ otherwise (notice as $\beta_0\, d_{31} >> \beta_0\, u_{31}$ and similarly for site i_{22} and i_{13}). The LTS associated to the above StoKlaim specification is shown in Fig. 3. For readability reasons we use the following conventions: (i) transitions are labeled with the associated rate-names only, since the action involved can easily be deduced from the context; (ii) details such as allocation environments or rate mappings are not shown, for the same reason; (iii) each state of the network is represented as a square box

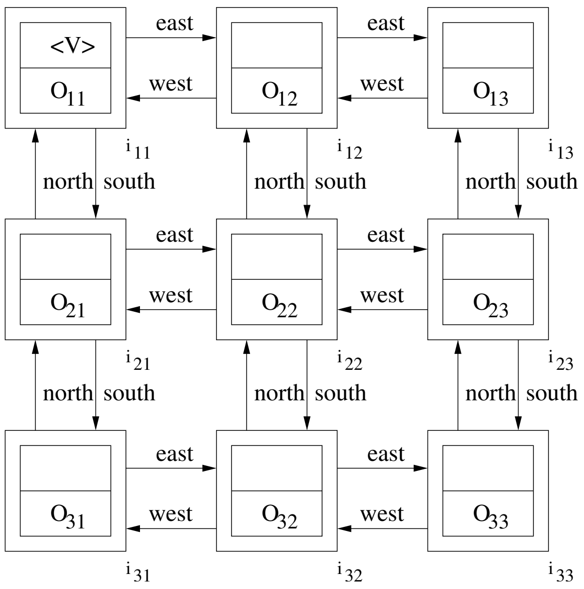

Figure 2: Initial configuration

with nine sectors: a blank sector jk corresponds to $i_{jk} ::_{\rho_{jk}} O_{jk}$, i.e. a normally running node; a black triangle in sector jk corresponds to the expression $i_{jk} ::_{\rho_{jk}} O_{jk} \parallel i_{jk} ::_{\rho_{jk}} \langle V \rangle$, i.e. a node which has been infected; a thick $\times$ in sector jk corresponds to $i_{jk} ::_{\rho_{jk}} (\mathbf{eval}(V)@\mathsf{self}, r_{jk}).O_{jk}$, i.e. a node which is going to run the virus; and $i_{jk} ::_{\rho_{jk}} O_{jk} \parallel V$—i.e. the virus is running at the node—is represented by a black sector jk. By means of standard transient analysis of the CTMC associated to the LTS of Fig. 3, we can compute the probability that the virus is executing on site i_{33} by time t, i.e. the probability of state 27 of the CTMC (associated to the LTS) of the picture[7]. Such a probability is less then 10^{-5} for $t = 1$ and grows up to 0.01147 after 10 time units passed, remaining essentially the same (0.01152) for $t = 100$, showing an acceptable detection power of the firewall.

As we already mentioned in Sect. 1, in [13] we present a logic which allows the integrated formal characterization of functional temporal properties and non-functional ones, as those typical of CTMC stationary and transient analysis. Moreover, proper model-checking algorithms for the logic allow for their the automatic verification against CTMC generated from StoKlaim specifications.

5 Related Work

In our proposal for StoKlaim, at a conceptual level, we follow essentially the same approach as Priami in [27] where he extends the π-Calculus with stochastic features. There is however a key difference between our work and the above mentioned one. In fact, the basic model of interaction of π-Calculus processes is *synchronous*, while that of Klaim processes is *asynchronous*. Synchronization of actions with exponentially distributed durations poses non-trivial problems to the compositional definition of the operational semantics when the intuition on action execution times is to be preserved by composition operators. For an interesting discussion on the subject we refer the reader to [7]. The choice of using an asynchronous model of interaction as the underlying model for stochastic behaviour allows for a rather simple definition of the operational semantics. Moreover it preserves a direct relation between the rates assigned to actions in specifications and those assigned to them in the automata models associated to such specifications. Such a relation is

[7]The reader should remember that the states of the LTS associated to a StoKlaim specification coincide with those of the related CTMC.

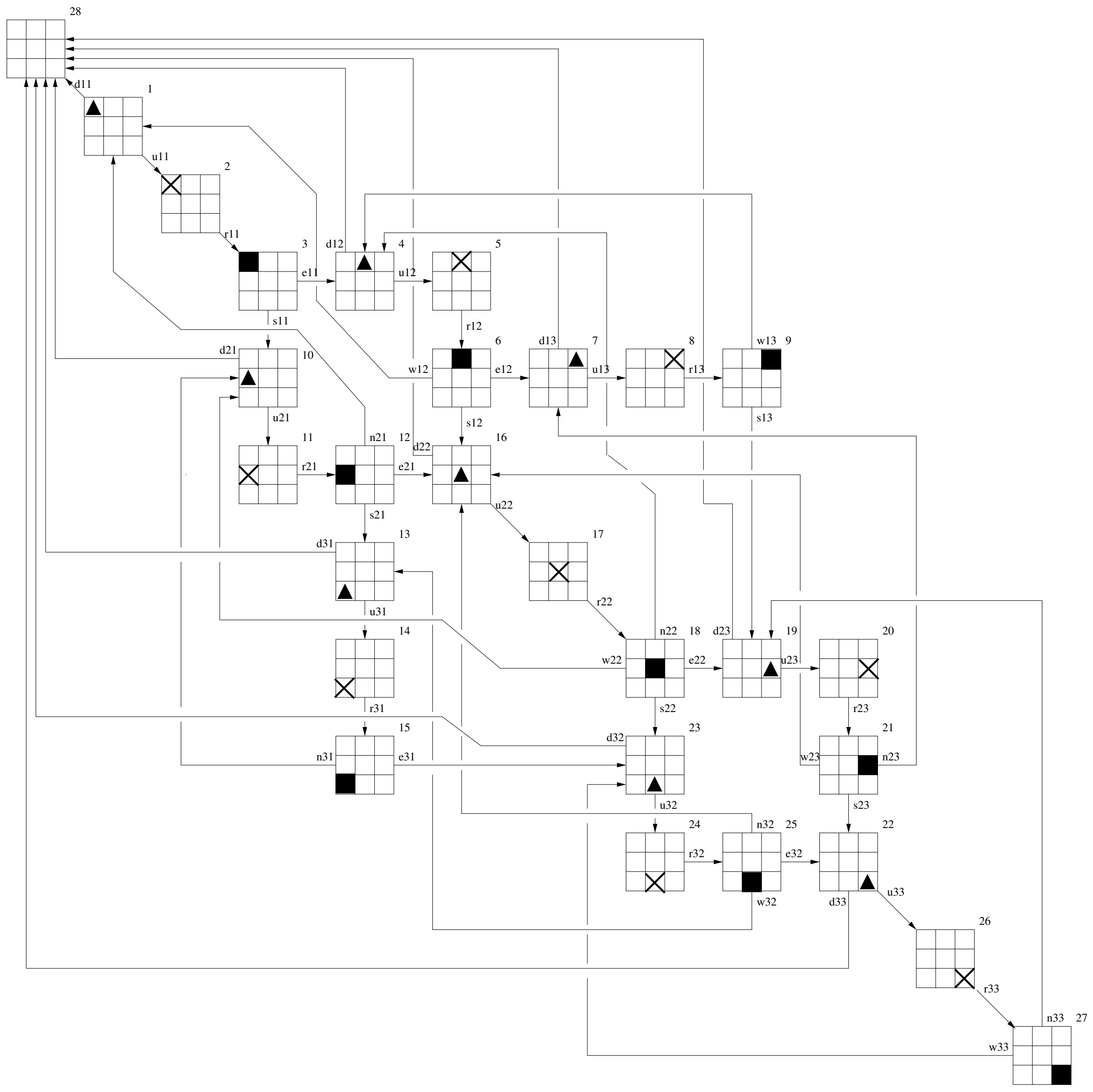

Figure 3: LTS for the virus example

more involving in approaches based on synchronous models of interaction due to rate/probability normalization procedures required by such models. Of course, the above advantages come at the price of dropping component synchronization as a primitive interaction mechanism. However, experience has shown that many fundamental behavioural aspects of mobile, cooperating agents in distributed networks can be satisfactorily described and analyzed by relying on asynchronous models of interaction [12, 5].

On a more technical level, another peculiarity of our approach is the fact that the definition of the operational semantics of the language is based on a structural congruence which *includes*, among others, *commutativity* and *associativity* of (network and) process parallel composition, non-deterministic choice, and *absorption*. The use of such "coarser" structural congruences greatly simplifies the definition of the operational semantics of locality-based, Klaim-like languages. By using approaches which rule out commutativity and associativity of parallel and choice operators, one cannot easily exploit the locality-based pattern matching style which is typical of the operational semantics definition of Klaim-like languages.

In [16] a probabilistic discrete- (resp. continuous-) time extension of full Klaim has been proposed. In that proposal, basically, all sources of non-determinism in the notation have been enriched with probabilistic information. In particular, (process) choice and parallel composition operators have been replaced by their probabilistic counterparts and, in the discrete-time case, probabilities have been added also to the network nodes used in network composition. Intuitively, the probability attached to nodes is related to the scheduling criteria at the global network level and extends the scheduling probability defined by the process parallel composition operator at the node level. In the continuous time case, rates of exponential distributions are associated to nodes, which are related to the execution time of any action in the node. Finally, the mappings of logical to physical addresses (i.e. Klaim allocation environments) have been replaced by mappings from logical addresses to probability distributions on physical ones. Our proposal is orthogonal to this approach in the sense that non-deterministic and parallel operators are left unchanged while *specific* rates are associated to *each* action, so that the former are features of the specific actions rather than of the site where the actions are executed. This gives rise to a clean semantic model which directly reflects the modeling choices expressed at the specification level, whereas the probabilities of different alternatives of choice, parallel, or network compositions are *derived* on the basis of the *race condition* principle [27]. In the proposal of [16] the specifier has several different conceptual tools and related linguistic constructs for expressing probabilistic information. On the other hand there is a certain interference among such concepts which results in several normalization steps. As a result, the relationship between the specific probability/rate values used in a specification and those resulting in the associated semantical structure can be quite complicated.

In [21] a probabilistic extension of the *asynchronous* π-Calculus is proposed, which does not address time and continuous distributions.

In [20] PEPA nets are proposed, where mobile code is modeled by expressions of the stochastic process algebra PEPA which play the role of tokens in (stochastic) Petri nets. The Petri net of a PEPA net models the architecture of the net, which to our understanding, is a static one. A PEPA expression can move from one place to another if there is a transition from the first place to the second. A proper synchronization mechanism between PEPA expressions and Petri nets is provided in order to fire transitions (i.e. to allow for code mobility).

In [8], a proposal for quantitative extensions of a tuple-based, Linda-like language is presented which includes the enrichment of the tuple space with *probabilistic* information attached to the tuples stored in the space. Such probabilities are represented by means of weights associated to tuples. The underlying semantic model is that of probabilistic transition systems. Interesting results are presented concerning the expressive power of the resulting language. The issue of action durations and underlying CTMCs is not addressed.

Finally, in [15] a preliminary version of StoKlaim has been proposed which extends a strict subset of Klaim. In particular logical addresses as well as allocation environments are not dealt with in [15], data are restricted to localities and templates are restricted to only one variable.

We are not aware of other proposals for stochastic/probabilistic calculi for mobile systems.

6 Conclusions and Future Work

In this paper we presented StoKlaim, a stochastic extension of Klaim, that makes it possible to integrate the modeling of quantitative and qualitative aspects of mobile systems.

The starting point of our proposal is to use continuous random variables with exponential distributions for modeling action durations in StoKlaim processes.

We presented the formal operational semantics for StoKlaim that associates an LTS to each StoKlaim specification and we showed how the LTS can be transformed into a Continuous Time Markov Chain (CTMC).

For illustrating the technique we worked out a small example modeling the spreading of a virus through a network. We gave an example of quantitative measure, namely the velocity of spreading of the virus that would be of interest for automatic verification using stochastic model checking techniques.

The results in this paper show that the choice of an asynchronous model of computation for the base-language greatly simplifies the definition of the operational semantics of the stochastic extension. Such definition is further simplified by the fact that the auxiliary structural congruence includes, among others, associativity and commutativity of parallel and non-deterministic operators. These advantages are not restricted to Klaim, but can be exploited for other languages based on an asynchronous model of interaction, s.a. Linda-based languages.

The ideas proposed in this paper give rise to a whole range of related interesting research questions. First of all, proper tools for supporting system modeling and verification based on StoKlaim and a proper temporal logic should be developed. The logic is addressed in [13] where a translation of a relavant fragment into aCSL—an action based version of the stochastic temporal logic CSL [1, 3] proposed in [23]—is defined as well. The logic is an extension of the one proposed in [14] for the simplified language described in [15]

Second, the modeling language should be extended in order to cover the recent extensions of Klaim, e.g. dynamic connections.

Further extensions concern the inclusion of rate-variables which would make it possible to dynamically modify the rates at which actions take place. Moreover, the inclusion of probabilistic operators, like probabilistic choice and probabilistic parallel composition, would certainly increase the expressive power of the language.

The explicit representation of non-determinism, by means of eliminating the race-condition assumption, would open the way to more realistic models, although in that case CTMCs can no longer be used as semantic model and other, richer, models like Markov Decision Processes [28], would be needed.

Similarly, the possibility of using distributions different from exponential ones would greatly increase the class of behaviours which can be modeled, requiring different analysis techniques such as discrete event simulation and the possibility of integrating the specification and use of Phase-Type distributions [26] in StoKlaim specifications.

References

[1] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Model checking Continuous Time Markov Chains. *ACM Transactions on Computational Logic*, 1(1):162–170, 2000.

[2] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model-Checking Algorithms for Continuous-Time Markov Chains. *IEEE Transactions on Software Engineering. IEEE CS*, 29(6):524–541, 2003.

[3] C. Baier, J.-P. Katoen, and H. Hermanns. Approximate Symbolic Model Checking of Continuous-Time Markov Chains. In J. Baeten and S. Mauw, editors, *Concur '99*, volume 1664 of *LNCS*, pages 146–162. Springer-Verlag, 1999.

[4] L. Bettini, V. Bono, R. De Nicola, G. Ferrari, D. Gorla, M. Loreti, E. Moggi, R. Pugliese, E. Tuosto, and B. Venneri. The Klaim Project: Theory and Practice. In C. Priami, editor, *Global Computing: Programming Environments, Languages, Security and Analysis of Systems*, volume 2874 of *LNCS*, pages 88–150. Springer-Verlag, 2003.

[5] L. Bettini, R. De Nicola, and M. Loreti. Formulae meet Programs over the Net: a Framework for Reliable Network Aware Programming, 2003. (submitted for publication. Available at: `http://music.dsi.unifi.it`).

[6] G. Boudol. ULM: a core programming model for global computing: (extended abstract). In D.A. Schmidt, editor, *Programming Languages and Systems, 13th European Symposium on Programming (ESOP)*, volume 2986 of *LNCS*, pages 234–248. Springer-Verlag, 2004.

[7] J. Bradley and N. Davies. Reliable Performance Modeling with Approximate Synchronisations. In J. Hillston and M. Silva, editors, *Proceedings of the 7th workshop on process algebras and performance modeling*, pages 99–118. Prensas Universitarias de Zaragoza, September 1999.

[8] M. Bravetti, R. Gorrieri, R. Lucchi, and G. Zavattaro. Quantitative Information in the Tuple Space Coordination Model. *Theoretical Computer Science. Elsevier.*, 346(1):28–57, 2005.

[9] L. Cardelli. A Language with Distributed Scope. In *22nd Annual ACM Symposium on Principles of Programming Languages*, pages 286–297. ACM, 1995.

[10] L. Cardelli. Abstractions for Mobile Computations. In J. Vitek and C. Jensen, editors, *Secure Internet Programming*, volume 1603 of *LNCS*, pages 51–94. Springer-Verlag, 1999.

[11] G. Castagna and J. Vitek. Seal: A framework for Secure Mobile Computations. In H. Bal, B. Belkhouche, and L. Cardelli, editors, *Internet Programming Languages*, volume 1686 of *LNCS*, pages 47–77. Springer-Verlag, 1999.

[12] R. De Nicola, G. Ferrari, and R. Pugliese. KLAIM: A Kernel Language for Agents Interaction and Mobility. *IEEE Transactions on Software Engineering. IEEE CS*, 24(5):315–329, 1998.

[13] R. De Nicola, J.-P. Katoen, D. Latella, and M. Massink. MoSL: A Stochastic Logic for StoKlaim. Technical report, Consiglio Nazionale delle Ricerche, Istituto di Scienza e Tecnologie dell'Informazione 'A. Faedo', 2005. (To appear).

[14] R. De Nicola, J.-P. Katoen, D. Latella, and M. Massink. Towards a logic for performance and mobility. In A. Cerone and H. Wiklicky, editors, *Quantitative Aspects of Programming Languages (QAPL).*, pages 132–146. LFCS, Univ. of Edimburg, 2005. (to appear in the Electronic Notes in Theoretical Computer Science. Elsevier).

[15] R. De Nicola, D. Latella, and M. Massink. Formal modeling and quantitative analysis of KLAIM-based mobile systems. In H. Haddad, L. Liebrock, A. Omicini, R. Wainwright, M. Palakal, M. Wilds, and H. Clausen, editors, *APPLIED COMPUTING 2005. Proceedings of the 20th Annual ACM Symposium on Applied Computing*, pages 428–435. Association for Computing Machinery - ACM, 2005. ISBN 1-58113-964-0.

[16] A. Di Pierro, C. Hankin, and H. Wiklicky. Probabilistic KLAIM. In R. De Nicola, G. Ferrari, and G. Meredith, editors, *Coordination Models and Languages*, volume 2949 of *LNCS*. Springer-Verlag, 2004.

[17] A. Fantechi, S. Gnesi, and G. Mazzarini. How Much Expressive Are LOTOS Expressions? In J. Quemada, J. Manas, and M. Thomas, editors, *Formal Description Techniques — III*. North-Holland Publishing Company, 1991.

[18] D. Gelernter. Generative Communication in Linda. *Communications of the ACM. ACM Press*, 7(1):80–112, 1985.

[19] A. Giacalone, C.C. Jou, and S.A. Smolka. Algebraic reasoning for probabilistic concurrent systems. In M. Broy and C.B. Jones, editors, *Working Conference on Programming Concepts and Methods*. IFIP TC 2, North-Holland Publishing Company, 1990.

[20] S. Gilmore, J. Hillston, L. Kloul, and M. Ribaudo. PEPA Nets: a Structured Performance Modelling Formalism. *Performance Evaluation - An International Journal. Elsevier*, 54:79–104, 2003.

[21] O. Herescu and C. Palamidessi. Probabilistic Asynchronous π-Calculus. In J. Tiuryn, editor, *FoSSaCS 2000*, volume 1784 of *LNCS*, pages 146–160. Springer-Verlag, 2000.

[22] H. Hermanns, U. Herzog, and J.-P. Katoen. Process algebra for performance evaluation. *Theoretical Computer Science. Elsevier.*, 274(1-2):43–87, 2002.

[23] H. Hermanns, J.-P Katoen, J. Meyer-Kayser, and M. Siegle. Towards Model Checking Stochastic Process Algebra. In W. Grieskamp, T. Santen, and B. Stoddart, editors, *Integrated Formal Methods - IFM 2000*, volume 1945 of *LNCS*, pages 420–439. Springer-Verlag, 2000.

[24] J. Hillston. Process algebras for quantitative analysis. In *IEEE Symposium on Logic in Computer Science*, pages 239–248. IEEE, Computer Society Press, 2005.

[25] V. Kulkarni. *Modeling and Analysis of Stochastic Systems*. Chapman & Hall, 1995.

[26] M.F. Neuts. *Matrix-geometric Solutions in Stochastic Models—An Algorithmic Approach*. The Johns Hopkins University Press, Baltimore, 1981.

[27] C. Priami. Stochastic π-Calculus. *The Computer Journal. Oxford University Press.*, 38(7):578–589, 1995.

[28] M.L. Puterman. *Markiv Decision Processes*. The Weizmann Institute of Science, 1994.

[29] E. Stark and S.A. Smolka. A complete axiom system for finite-state probabilistic processes. In G. Plotkin, C. Stirling, and M. Tofte, editors, *Proof, Language and Interaction: Essays in Honour of Robin Milner*. MIT Press, 2000.