QUANTICOL

A Quantitative Approach to Management and Design of Collective and Adaptive Behaviours

quanticol
http://www.quanticol.eu

TR-QC-3-2014

Stochastically timed predicate-based communication primitives for autonomic computing - Full Paper

Revision: 0.0; Feb 5, 2014

Author(s): Diego Latella (CNR), Michele Loreti (IMT-Lucca), Mieke Massink (CNR), Valerio Senni (IMT-Lucca)

Publication date: Feb 5, 2014

Funding Scheme: Small or medium scale focused research project (STREP)
Topic: ICT-2011 9.10: FET-Proactive 'Fundamentals of Collective Adaptive Systems' (FOCAS)
Project number: 600708

Coordinator: Jane Hillston (UEDIN)
e-mail: `Jane.Hillston@ed.ac.uk`
Fax: +44 131 651 1426

Part. no.	Participant organisation name	Acronym	Country
1 (Coord.)	University of Edinburgh	UEDIN	UK
2	Consiglio Nazionale delle Ricerche – Istituto di Scienza e Tecnologie della Informazione "A. Faedo"	CNR	Italy
3	Ludwig-Maximilians-Universität München	LMU	Germany
4	Ecole Polytechnique Fédérale de Lausanne	EPFL	Switzerland
5	IMT Lucca	IMT	Italy
6	University of Southampton	SOTON	UK

COOPERATION

Contents

Abstract

Predicate-based communication allows components of a system to send messages and requests to ensembles of components that are determined at execution time through the evaluation of a predicate, in a multicast fashion. Predicate-based communication can greatly simplify the programming of autonomous and adaptive systems. We present a stochastically timed extension of the Software Component Ensemble Language (SCEL) that was introduced in previous work. Such an extension raises a number of non-trivial design and formal semantics issues with different options as possible solutions at different levels of abstraction. We discuss four of these options. We provide formal semantics and an illustration of the use of the language modeling a variant of a bike sharing system, together with some preliminary analysis of the system performance.

1 Introduction

SCEL (Software Component Ensemble Language) [6], is a kernel language that is equipped with programming abstractions for the specification of system models within the framework of the *autonomic computing* paradigm, and for programming such systems. These abstractions are specifically designed for representing behaviours, knowledge, and aggregations according to specific policies, and to support programming context-awareness, self-awareness, and adaptation. SCEL is parametric with respect to some syntactic categories, namely Knowledge, Policies, Templates, and Items (Templates and Items determine the part of Knowledge to be retrieved/removed or added, respectively).

The main focus of the SCEL language is on supporting the development of autonomous, loosely-coupled, component-based software systems. For this purpose, a number of underlying assumptions are made on the kind of peculiarities of these software systems, among which adaptivity, open-endedness, ensemble-orientedness, high ability of reconfiguration, and support for heterogeneity. Two novel key aspects of SCEL, that distinguish it from other languages, are designed to support these peculiarities: predicate-based communication and the role of the component knowledge-base. Predicate-based communication allows to send messages to *ensembles* of components that are not predetermined at modeling time, but are defined at execution time, depending on how the communication predicate evaluates w.r.t. the receiver interface. The component knowledge-base allows to realize various adaptation patterns, by explicit separation of adaptation data in the spirit of [4], and to model components view on (and awareness of) the environment. In the first three years of the ASCENS project SCEL has been used to specify many scenarios related to the project Case Studies [13, 11, 16, 15]. These specifications witness how SCEL primitives simplify the programming of autonomous and adaptive systems. In these systems, the emerging behavior is realized through the coordination of the components activities.

In this paper we address the problem of enriching SCEL with information about action durations, by providing a stochastic semantics for the language. There exist various frameworks that support the systematic development of stochastic languages, such as [8]. However, the main challenge in developing a stochastic semantics for SCEL is in making appropriate modeling choices, both taking into account the specific application needs and allowing to manage model complexity and size. Our contribution in this work is the proposal of four variants of StocS, a Markovian extension of a significant fragment of SCEL, that can be used to support quantitative analysis of adaptive systems composed of ensembles of cooperating components. These variants adopt *the same language syntax* of SCEL but denote different underlying stochastic models, having a different level of granularity. The choices we make will be motivated via examples and considerations on the model complexity and size. Of course one could encode these four semantics in an expressive language like Bio-PEPA [5] or HYPE [1], thus obtaining four fragments of Bio-PEPA/HYPE, one for each semantics.

Instead, what we do in this paper is to develop a single language and define four possible semantics for it, allowing to describe models at a different level of detail (by simply modifying labels and relations used to construct the LTS). From a software engineering point of view, this is a more pragmatic and flexible solution.

In summary, StocS is essentially a *modeling language* which inherits the purpose and focus of SCEL. StocS extends SCEL by modeling the average time duration of state-permanence and by replacing non-determinism by a probability distribution over outgoing transitions, thus adopting a CTMC-based operational semantics [7]. In the current preliminary phase of the design of StocS, we deliberately omit to incorporate certain advanced features of SCEL, such as the presence and role of policies. Certain other features, such as the possibility of attaching components a unique identity, are discussed and evaluated for their impact on StocS (See Section 2). A further issue of discussion is whether StocS actions should have an atomic semantics or not (See Section 2.2) and the physical meaning of these assumptions.

Finally, an important aspect in a modeling language concerns the need of devising an appropriate syntax to express the environment model. In StocS and SCEL the only point of contact with the environment is the knowledge base, which contains both internal information and externally-sensed events. In our approach, the knowledge is the most appropriate part of the language to specify environment models.

The outline of the present report is as follows. Section 2 discusses the trade-offs between four stochastic variants of SCEL, followed by the presentation of their formal semantics in Sections 4, 5, 6, and 7, after some preliminary definitions are recalled in Section 3. Section 8 introduces a case study to illustrate various aspects of the use of the design language in the context of a smart bike sharing system. Concluding remarks and lines for possible future research are presented in Section 9.

Note: *The material presented in Section 6 has been originally developed in the context of the EU project ASCENS (nr.257414) and is reported on in [12]. We included it in the present report for completeness and self-containment reasons.*

2 StocS: a Stochastic extension of SCEL

In this section we present the main features of StocS. We start by illustrating its main syntactic ingredients. Then, we discuss possible choices for its timed semantics.

2.1 Syntax

The syntax of StocS is presented in Table 1, where the syntactic categories of the language are defined. The basic category defines Processes that are used to specify the order in which Actions can be performed. Sets of processes are used to define the behavior of Components, that in turn are used to define Systems. Actions operate on local or remote knowledge-bases and have a Target to determine which other components are involved in the action. As we mentioned in the Introduction, for the sake of simplicity, in this version of StocS we do not include Policies, whereas, like SCEL, StocS is parametric w.r.t. Knowledge, Templates and Items.

We define the following domains for variables and for defining functions signature: $\mathbb{A}$ is the set of attribute names (which include the constant id used to indicate the component identifier), $\mathbb{V}$ is the set of values, $\mathbb{K}$ is the set of possible knowledge states, $\mathbb{I}$ is the set of knowledge items, $\mathbb{T}$ is the set of knowledge templates. So, in Table 1, $\mathsf{a} \in \mathbb{A}$, $v \in \mathbb{V}$, $K \in \mathbb{K}$, $t \in \mathbb{I}$, $T \in \mathbb{T}$.

Systems:	S	$::=$	$C \mid S \parallel S$
Components:	C	$::=$	$I\,[K,\, P]$
Processes:	P	$::=$	$\mathbf{nil} \mid a.P \mid P + P \mid P \mid P \mid X \mid A(\bar{p})$
Actions:	a	$::=$	$\mathbf{get}(T)@c \mid \mathbf{qry}(T)@c \mid \mathbf{put}(t)@c$
Targets:	c	$::=$	$\mathsf{self} \mid \mathsf{p}$
Ensemble Predicates:	p	$::=$	$tt \mid e \bowtie e \mid \neg\mathsf{p} \mid \mathsf{p} \wedge \mathsf{p}$ with $\bowtie \in \{<, \leq, >, \geq\}$
Expressions:	e	$::=$	$v \mid \mathsf{x} \mid \mathsf{a} \mid \ldots$

Table 1: StocS syntax (Knowledge K, Templates T, and Items t are parameters)

Example 1 (Items and Templates as Tuples and Patterns)**.** *Consider a signature* $(\mathcal{V}, \mathcal{F})$ *where* $\mathcal{V}$ *is a set of variables and* $\mathcal{F}$ *is a set of function symbols with arity (we indicate by* f/n *a function symbol* f *with arity* n*) such that* $\langle\rangle/i \in \mathcal{F}$ *for* $i = 0, 1, 2, \ldots$*. We denote by* $Terms(\mathcal{V}, \mathcal{F})$ *the set of all possible finite terms on the given signature (i.e. the terms with variables, constructed respecting function symbols arities) and by* $Terms(\mathcal{F})$ *the set of all possible finite ground terms. A* pattern *is a term of the form* $\langle t_1, \ldots, t_n\rangle$*, with* $t_i \in Terms(\mathcal{V}, \mathcal{F})$ *for* $i = 1, \ldots, n$*. A* tuple *is a term of the form* $\langle t_1, \ldots, t_n\rangle$*, with* $t_i \in Terms(\mathcal{F})$ *for* $i = 1, \ldots, n$*. In this example we have defined the set of Templates* $\mathbb{T}$ *as the set of patterns and the set of Items* $\mathbb{I}$ *as the set of tuples.*

Systems and components

We let *Sys* denote the set of systems defined by the Syntax in Table 1 and S, S_1,..., S'...variables ranging over *Sys*. A system S consists of an aggregation of components obtained via the (parallel) *composition* operator $\_ \parallel \_$. A component $I\,[K,\, P]$ consists of:

1. An *interface*, which is a function I in the set $\mathbb{K} \to (\mathbb{A} \to \mathbb{V})$ used for publishing information about the component state in the form of attribute values. In detail, an interface defines a (partial) function from a pair knowledge-base and attribute-name to a domain of values. Among the possible attributes, id is mandatory and is bound to the name of the component. Component names are not required to be unique, so that replicated service components can be modeled. The *evaluation* of an interface I in a knowledge state K is denoted as $I(K)$. The set of possible interface evaluations is denoted as $\mathbb{E}$.

2. A *knowledge repository* K, managing both application data and awareness data (following the approach of [4]), together with the specific handling mechanism.

3. A *process* P, together with a set of process definitions. Processes may execute local computations, coordinate local and remote interaction with a knowledge repository, or perform adaptation and reconfiguration.

Processes

Processes are the active computational units. Each process is built up from the *inert* process **nil** via *action prefixing* ($a.P$), *nondeterministic choice* ($P_1 + P_2$), *parallel composition* ($P_1 | P_2$), *process variable* (X), and *parameterised process invocation* ($A(\bar{p})$). We feel free to omit trailing occurrences of **nil**, writing e.g. a instead of $a.\mathbf{nil}$, whenever there is no confusion arising.

Process variables can support *higher-order* communication, namely the capability to exchange (the code of) a process, and possibly execute it, by first adding an item containing the process to a knowledge repository and then retrieving/withdrawing this item while binding the process to a process variable. We assume that A ranges over a set of parameterised *process identifiers* that are used in recursive process definitions. We also assume that each process identifier A has a *single* definition of the form $A(\bar{f}) \triangleq P$ where all free variables in P are contained in $\bar{f}$ and all occurrences of process identifiers in P are within the scope of an action prefixing. $\bar{p}$ and $\bar{f}$ denote lists of actual and formal parameters, respectively. In the sequel we will use *Proc* to denote the set of processes, ranged over by variables P, Q,..., P_1,Q_1...., P',Q',....

Actions and targets

Processes can perform three different kinds of Actions: $\mathbf{get}(T)@c$, $\mathbf{qry}(T)@c$ and $\mathbf{put}(t)@c$, used to act over shared knowledge repositories by, respectively, withdrawing, retrieving, and adding information items from/to the knowledge repository identified by c.

These actions exploit templates T as patterns to select knowledge items t in the repositories. The precise syntax of templates and knowledge items depends on the specific instance of *knowledge repository* that is used. Indeed, in Example 1 we provided the syntax for items ($\mathbb{I}$) and templates ($\mathbb{T}$) for one possible instance of the repository. In the next section we show how StocS is in fact parametric with respect to different types of *knowledge repository.*

In the following, we identify the component executing an action as the *source*, and the component (whose knowledge state is) affected by the action as the *destination.* In order for this synchronization to occur, the *output* action (performed on the source side) has a corresponding *input* action (performed on the destination side). In terms of action labels, they are denoted by $\overline{\alpha}$ and α, respectively, while their synchronization is denoted by $\overleftrightarrow{\alpha}$.

For the sake of simplicity, in this report we restrict targets c to the distinguished variable self, that is used by processes to refer to the component hosting it, and to component *predicates* p, i.e. formulas on component attributes. A component $I\,[\,K,\,P]$ is *identified* by a predicate p if $I(K) \models \mathsf{p}$, that is, the interpretation defined by the evaluation of I in the knowledge state K is a model of the formula p. Note that here we are assuming a fixed interpretation for functions and predicate symbols that are not within the attributes ($\mathbb{A}$). E.g. $battery < 3$ is a possible predicate, where $<$ and 3 have a fixed interpretation, while the value of *battery* depends on the specific component addressed.

The informal, abstract, semantics of the actions is the following:

- $\mathbf{put}(t)@c$ is non-blocking, its execution causes knowledge item t be added to the knowledge repository of *all* the components (the interface of which is) identified by c, *if any*;
- $\mathbf{get}(T)@c$ ($\mathbf{qry}(T)@c$, respectively) is blocking, it causes a knowledge item t matching pattern T be withdrawn (retrieved, respectively) from the knowledge repository of *any of* the components (the interface of which is) identified by c, *if any.* If no such component/item is available, the process executing it is *blocked* in a waiting state. The two actions differ for the fact that **get** removes the requested item from the knowledge repository while **qry** leaves the target repository unchanged.

The set of components satisfying a given target c of a communication action can be considered as the *ensemble* with which the process performing the action intends to interact.

Knowledge behavior

Since StocS is parametric w.r.t. the specific knowledge repository used in a specification, we provide no specific syntax/semantics for knowledge repositories. We only require that a *knowledge repository type* is completely described by a tuple $(\mathbb{K}, \mathbb{I}, \mathbb{T}, \oplus, \ominus, \vdash)$ where $\mathbb{K}$ is the set of possible *knowledge states* (the variables K, K_1, ..., K', ... range over $\mathbb{K}$), $\mathbb{I}$ is the set of *knowledge items* (the variables t, t_1,…,t',… range over $\mathbb{I}$) and $\mathbb{T}$ is the set of *knowledge templates* (the variables T, T_1,…, T',… range over $\mathbb{T}$). Knowledge items have no variable, while knowledge templates have. We assume to have a partial function $\mathsf{match} : \mathbb{T} \times \mathbb{I} \to \mathsf{Subst}(\mathbb{I})$ (where $\mathsf{Subst}(X)$ is the set of substitutions with range in X) and we denote as $\mathsf{match}(T, t) = \vartheta$ the substitution obtained by matching the pattern T against the item t, if any. By a small abuse of notation, we write $\neg\mathsf{match}(T, t)$ to denote that $\mathsf{match}(T, t)$ is undefined.

The operators $\oplus$, $\ominus$, $\vdash$ are used to add, withdraw, and infer knowledge items to/from knowledge repositories in $\mathbb{K}$, respectively. These functions have the following signature, where $\mathsf{Dist}(X)$ denotes the class of probability distributions on set X with finite support:

- $\oplus : \mathbb{K} \times \mathbb{I} \to \mathsf{Dist}(\mathbb{K})$.
- $\ominus : \mathbb{K} \times \mathbb{T} \hookrightarrow \mathsf{Dist}(\mathbb{K} \times \mathbb{I})$;
- $\vdash : \mathbb{K} \times \mathbb{T} \hookrightarrow \mathsf{Dist}(\mathbb{I})$;

Function $\oplus$ is *total* and defines how a knowledge item can be inserted into a knowledge repository: $K \oplus t = \pi$ is the probability distribution over knowledge states obtained as the effect of adding t. If the item addition operation is modeled in a deterministic way, then the distribution π is a Dirac function. One advantage of allowing a probabilistic item addition operation is, for example, the ability of modeling possible failures in the item addition. We will make use of this feature in the stochastic semantics of StocS.

Function $\ominus$ is *partial* and computes the result of withdrawing a template from a knowledge state in terms of a probability distribution $K \ominus T$ over the set of pairs $(K, t) \in (\mathbb{K} \times \mathbb{I})$ such that the item t matches the template T. Intuitively, if $K \ominus T = \pi$ and $\pi(K', t) = p$ then, when one tries to remove an item matching template T from K, with probability p item t is obtained and the resulting knowledge state is K'. If a tuple matching template T is not found in K then $K \ominus T$ is undefined, which is indicated by $K \ominus T = \bot$.

Function $\vdash$ is *partial* and computes (similarly to $\ominus$) a probability distribution over the possible knowledge items matching template T that can be inferred from K. Thus, if $K \vdash T = \pi$ and $\pi(t) = p$ then the probability of inferring t when one tries to infer from K a tuple matching T is p. If no tuple matching T can be inferred from K then $K \vdash T$ is undefined, which is indicated by $K \vdash T = \bot$.

Example 2 (K as a tuple store)**.** *Consider the sets $\mathbb{I}$ of items and $\mathbb{T}$ of templates defined in Example 1. A knowledge state K is a multi-set over a set $M \subseteq \mathbb{I}$, that is a pair $\langle M, f_K \rangle$, where $f_K : M \to \mathbb{N}$ is the so-called* characteristic function *of K. Given two multisets A and B, the functions $A \uplus B$ (B/A) denote the addition (resp. removal) of the elements of A and (resp. from) B. The restriction $M|_T$ of a multiset M to a pattern T is the sub-multiset of M of those t that match T, that is: $M|_T = \{\{t \mid t \in M \wedge \exists\vartheta.\ \mathsf{match}(T, t) = \vartheta\}\}$. In the following we assume that $[x_1 \mapsto y_1, \ldots, x_n \mapsto y_n]$ is the function mapping x_i to y_i for $i = 1, \ldots, n$. Furthermore, we use standard conditional notation to specify y_i. Finally, given to functions f and g, we let $(f + g)x = fx + gx$. For any $t \in \mathbb{I}$, $T \in \mathbb{T}$ and knowledge state K:*

$$\begin{aligned}
K \oplus t &= [K \uplus \{t\} \mapsto 1] \\
K \oplus_p t &= [K \uplus \{t\} \mapsto p,\ K \mapsto 1{-}p] \\
K \ominus T &= \textstyle\sum_{t \in \mathbb{I}} \left[(K/\{t\}, t) \mapsto \begin{cases} 0 & \textit{if } \neg\,\mathsf{match}(T, t) \\ \frac{f_K(t)}{card(M|_T)} & \textit{otherwise} \end{cases} \right] \\
K \vdash T &= \textstyle\sum_{t \in \mathbb{I}} \left[t \mapsto \begin{cases} 0 & \textit{if } \neg\,\mathsf{match}(T, t) \\ \frac{f_K(t)}{card(M|_T)} & \textit{otherwise} \end{cases} \right]
\end{aligned}$$

Concerning $\ominus$ and $\vdash$, one can choose a different probability distribution over matching items that the one presented in this example (which is a uniform distribution over multi-set elements), depending on the specific modeling domain.

2.2 Informal Timed Semantics - Four Variants

The semantics of SCEL does not consider any time related aspect of computation. More specifically, the execution of an action of the form $\mathbf{act}(T)@c\,.\,P$ (for **put**/**get**/**qry** actions) is described by a *single* transition of the underlying SCEL LTS semantics. In the system state reached by such a transition it is guaranteed that the process which executed the action is in its local state P and that the knowledge repositories of all components involved in the action execution have been modified accordingly. In particular, SCEL abstracts away details concerning:

1. when the execution of the action starts;
2. if c is a predicate p, when the possible destination components are required to satisfy p;
3. when the process executing the action resumes execution (i.e. becomes P);

and their consequent time relationship. If we want to extend SCEL with an explicit notion of (stochastic) time, we need to take into account the time-related issues mentioned above. These issues can be addressed at different levels of abstraction, reflecting a different choice of details that are considered in modeling SCEL actions. In this section, we discuss and motivate several choices we take in the design of StocS and we also discuss alternative ones. In order to obtain an underlying CTMC semantics, we model state residence times in a Markovian way. Therefore, in the following, whenever we indicate that an action has rate λ, roughly speaking we mean that the duration of the action (or, equivalently, the state residence time before action execution) is modeled by a random variable (RV, in the sequel) with negative exponential distribution having rate λ. Strictly speaking, the actual residence-time depends also on other conflicting actions the process may be engaged in, and the resulting race-condition.

Concerning point (1) above, since actions may involve one or more components (except those executing on self), it is reasonable to distinguish the period in which the source component determines the set of target components (*observation*) from the period in which these target components are involved (*action realization*).

Point (2), in the case c is the predicate p, requires to define *when* a component satisfies p with respect to a process executing an action, when time and possibly space are taken into consideration. We assume that source components are not aware of which are the components satisfying predicate p. Therefore, we define the notion of *observation* of the component by the process, the result of which allows to establish whether the component satisfies the predicate or not. In the context of distributed systems this is very often realized by means of a message sent by the process to the component. According to this view, the check whether a component satisfies predicate p is performed *when the message reaches it.* This means that a StocS action may require broadcast communication to be executed, even if its effect involves a few (and possibly no) components. In distributed systems different components may have different response times depending on different network conditions and one can model explicitly the message delivery, taking into account the time required to reach the component.

Finally, point (3) raises the issue on when source component execution is to be resumed. In particular, it is necessary to identify how the source component is made aware that its role in the communication has been completed. Get/query actions are blocking and they terminate when the source receives a knowledge item from any component. A reasonable choice is that further responses received are ignored. We assume appropriate mechanisms that ensure no confusion arises between distinct actions and corresponding messages. Put actions are non-blocking, so it is sufficient that the source component is aware that all reachable components have been involved in the evaluation of the predicate. A possible choice is to set-up the transmission of one request of predicate evaluation for each component and then terminate the execution on the source side immediately. On the target side, it is necessary to model the reception time as well as subsequent evaluation and corresponding knowledge repository modification.

Depending on the degree of detail in modeling these aspects, we will define four different semantics, that we call: *network-oriented* (NET-OR), *action-oriented* (ACT-OR), *interaction-oriented* (INT-OR), and *activity-oriented* (ACTIV-OR). These semantics have an increasing level of abstraction, which allows us to manage the complexity of the model according to the application of interest. In the remaining part of this section we informally describe these four variants of the stochastic semantics and their motivations. Then, in Sections 6, 4, 5, and 7, we provide the corresponding operational semantics. The sections containing the operational semantics are ordered with the purpose of presenting the simpler semantics first (which is the action-oriented one), and then the others in an incremental way, as refinements or modifications of the previous ones.

2.2.1 Network-oriented Semantics

This semantics takes into account all of the assumptions and observations described previously, which entails that actions are *non-atomic*. Indeed, they are executed through several intermediate steps, each of which requires appropriate time duration modeling. In particular, **put** actions are realized in two steps: (1) an envelope preparation and shipping (one for each component in the system, other than the source), (2) envelope delivery, with its own delivery time, test of the truth value of the communication predicate, and update of the knowledge-state. The actions **get**/**qry** are realized in two steps: (1) initiation of the item retrieval by a source component by entering in a waiting state, (2) synchronization with a destination component and exchange of the retrieved item. Since actions are not executed atomically, their execution is interleaved with that of other actions executed in parallel. This kind of semantics is appropriate for models with spatial aspects, where distribution is a sensible aspect influencing the duration of communications on the basis of the location of components. A reasonable assumption is that rates of locally executed actions are rather high, if compared to rates of remote actions. If the rates of local and remote actions differ orders of magnitudes, this may lead to the so-called “stiff” Markov models. An efficient analysis of such models may require a multi-scale (in time) analysis approach.

In Section 6 we provide an operational semantics for this variant of the language.

2.2.2 Action-oriented

In this simplified semantics, an action of the form $\mathbf{act}(T)@c$ (for **put**/**get**/**qry** actions) is described by a *single* transition and has a state residence time provided by a function $\mathcal{R}$ that takes into account the *source* component, the *target* component, the *cost* of retrieved/transmitted knowledge item, and possibly other parameters. Thus, we can expect this rate to (partially) take into account different locations and thus different response times of components. Upon item retrieval (by a **get**/**qry** action), eligible components (in terms of predicate satisfaction and availability of requested item) are in a race for response, with rate assigned component-wise. The underlying stochastic semantics drives the outcome of the race, with appropriate weighting depending on the rates. Even if this semantics does not consider the realistic aspects of predicate-based communication, it is simpler and can be used in all of those scenarios where the average execution time of actions does not depend on the number of components involved in the communication.

In Section 4 we define an operational semantics based on these ideas.

2.2.3 Interaction-oriented

This is based on the action-oriented semantics and distinguishes local and remote actions, by assuming that local actions are executed instantaneously. In some scenarios, local actions happen in a time-scale which is very different (usually much smaller) from that of remote actions. In these situations it is reasonable to consider as instantaneous the execution of local actions, which is the idea we realize in the definition of the INT-OR semantics. As a useful side effect of ignoring the duration of local actions, we obtain more concise models. This approximation in the modeling can be considered as an approach to reducing multi-scale models to single-scale models. In these single-scale models, the macro-scale of inter-component communication drives the execution of macro-actions. A similar idea is explored for Bio-PEPA models in [9] and used to abstract away from fast reactions in biochemical networks. There, under the so-called Quasi-Steady-State Assumption of the system, it is also defined a form of bisimilarity between the abstract and the concrete model.

The assumption we make for defining this semantics is that each remote (timed) action is followed by a (possibly empty) sequence of local (probabilistic) actions. We ensure this assumption is satisfied by imposing syntactic restrictions on processes. Then, by realizing a form of maximal progress [10] we execute a timed action and all of its subsequent probabilistic actions in a single transition of the StocS LTS. Note that, similarly to the NET-OR semantics, also in this semantics we assume an error probability (called p_{err}) modeling failed delivery of the **put** action.

In Section 5 we define an operational semantics based on these ideas.

2.2.4 Activity-oriented

This semantics is very abstract and allows to explicitly declare as atomic an entire sequence of actions, by assigning to it an execution rate that models the duration of the entire sequence. Since the execution of the sequence of actions is atomic, it allows no interleaving of other actions. As an interesting consequence of this, we have a significant reduction in the state-space of the system.

This variant of the semantics is motivated by the fact that StocS only provides primitives for asynchronous communication. Synchronization, if needed, has to be encoded through a protocol using asynchronous communication primitives [14]. Whatever is the adopted semantics among the previous ones (for example, the Action-oriented semantics), the protocol execution for the synchronization action is interleaved with the execution of other actions. This leads to have unclear dependencies of protocol execution times from the environment. Therefore, the Activity-oriented semantics allows us to declare as atomic an entire sequence of actions and to assign a rate to it. More in general, the purpose of this semantics is to have a very high-level abstraction of the interaction mechanisms. Of course, this must be handled with care as potentially relevant system behaviors (and interleavings) may be no longer present in the model. Therefore, properties of the model are not necessarily satisfied also by the system. Also in this semantics we assume an error probability (called p_{err}) modeling failed delivery of the **put** action.

3 Preliminary Definitions for Operational Semantics

In this section we provide preliminary notions to support the presentation of the four stochastic semantics of StocS formalizing the ideas described in the previous section. The semantics definition is given in the FuTSs style [8] and, in particular, using its Rate Transition Systems (RTS) instantiation [7].

3.1 Preliminaries

In RTSs a transition is a triple of the form $(P, \alpha, \mathscr{P})$, the first and second components of which are the source state and the transition label, as usual, and the third component $\mathscr{P}$ is the *continuation function* that associates a real non-negative value with each state P'. A non-zero value represents the rate of the exponential distribution characterizing the time needed for the execution of the action represented by α, necessary to reach P' from P via the transition. Whenever $\mathscr{P}(P') = 0$, this means that P' is not reachable from P via α. RTS continuation functions are equipped with a rich set of operations that help to define these functions over sets of processes, components, and systems. Below we show the definition of those functions that we use in this paper, after having recalled some basic notation, and we define them in an abstract way, with respect to a generic set X.

Let $\mathbf{TF}(X, \mathbb{R}_{\geq 0})$ denote the set of *total* functions from X to $\mathbb{R}_{\geq 0}$, and $\mathscr{F}, \mathscr{P}, \mathscr{Q}, \mathscr{R}, \ldots$ range over it. We define $\mathbf{FTF}(X, \mathbb{R}_{\geq 0})$ as the subset of $\mathbf{TF}(X, \mathbb{R}_{\geq 0})$ containing only functions with *finite support*: function $\mathscr{F}$ is an element of $\mathbf{FTF}(X, \mathbb{R}_{\geq 0})$ if and only if there exist $\{d_1, \ldots, d_m\} \subseteq X$, the *support* of $\mathscr{F}$, such that $\mathscr{F}\, d_i \neq 0$ for $i = 1 \ldots m$ and $\mathscr{F}\, d = 0$ for all $d \in X \setminus \{d_1, \ldots, d_m\}$. We equip $\mathbf{FTF}(X, \mathbb{R}_{\geq 0})$ with the operators defined below. The resulting *algebraic structure* of the set of finite support functions will be crucial for the compositional features of our approach.

Def. 3.1. *Let X be a set, and $d, d_1, \ldots, d_m$ be distinct elements of X, $\gamma_1, \ldots, \gamma_m \in \mathbb{R}_{\geq 0}$, let furthermore $\bullet : X \times X \to X$ be an injective binary operator, $\mathscr{F}_1$ and $\mathscr{F}_2$ in $\mathbf{FTF}(X, \mathbb{R}_{\geq 0})$:*

1. *$[d_1 \mapsto \gamma_1, \ldots, d_m \mapsto \gamma_m]$ denotes the following function:*

$$[d_1 \mapsto \gamma_1, \ldots, d_m \mapsto \gamma_m]\, d \;=_{\text{def}}\; \begin{cases} \gamma_i & \textit{if } d = d_i \in \{d_1, \ldots, d_m\}, \\ 0 & \textit{otherwise.} \end{cases}$$

   the 0 constant function in $\mathbf{FTF}(X, \mathbb{R}_{\geq 0})$ is denoted by $[]$;

2. *Function $+$ is defined as $(\mathscr{F}_1 + \mathscr{F}_2)\, d \;=_{\text{def}}\; (\mathscr{F}_1\, d) + (\mathscr{F}_2\, d)$;*

3. *We lift operator $\bullet$ to $\mathbf{FTF}(X, \mathbb{R}_{\geq 0})$ as follows (with a bit of overloading):*

$$(\mathscr{F}_1 \bullet \mathscr{F}_2)s \;=_{\text{def}}\; \begin{cases} (\mathscr{F}_1\, d_1) \cdot (\mathscr{F}_2\, d_2) & \textit{if} \quad \exists d_1, d_2 \in X \;.\quad d \;=\; d_1 \bullet d_2, \\ 0 & \textit{otherwise.} \end{cases}$$

4. *The characteristic function $\mathcal{X} : X \to \mathbf{FTF}(X, \mathbb{R}_{\geq 0})$ with $\mathcal{X}\, d \;=_{\text{def}}\; [d \mapsto 1]$.*

Def. 3.2. *An A-*labelled Rate Transition System *(RTS) is a tuple $(S, A, \mathbb{R}_{\geq 0}, \rightarrowtail)$ where S and A are countable, non-empty, sets of* states *and* transition *labels, respectively, and $\rightarrowtail \subseteq S \times A \times \mathbf{FTF}(S, \mathbb{R}_{\geq 0})$ is the A-labelled* transition relation.

In order to distinguish and identify the rules of the semantics definition, we label them by unique names. Note that a rule with name r may have one or more associated blocking rules r_{B} which have the role of allowing the execution of no actions other than those explicitly allowed by existing inference rules. These B-rules will not be further commented in the following sections.

4 Action-oriented Operational Semantics

To facilitate the presentation of the StocS variants, we start with the semantics called Action-oriented (abbreviated to ACT-OR), associating a single transition to each StocS action. The more complex semantics, presented in Sections 5, 6, and 7, will be described as modifications of this simpler one.

Before discussing the semantics in detail, we want to discuss the role of interface evaluations. We have explained, in Sec. 2, that interface evaluations are used to establish the truth value of predicates. Now we want to discuss how they are also used to compute appropriate action rates. Indeed, interface evaluations are included within the transition labels to exchange information about source/destination components in a synchronization action, as we now explain.

Interface evaluations (i.e. the values of the component attributes at a given time) are used within a *rate function* $\mathcal{R} : \mathbb{E} \times Act \times \mathbb{E} \to \mathbb{R}_{\geq 0}$, which takes the interface evaluation of the *source*, an action in the set of labels

$$Act = \{\mathbf{put}(t)@c,\ \mathbf{get}(T : t)@c,\ \mathbf{qry}(T : t)@c \ \mid\ t \in \mathbb{I} \text{ and } T \in \mathbb{T} \text{ and } c \in \textsc{Target}\}$$

and the interface evaluation of the *destination*, and returns a value in $\mathbb{R}_{\geq 0}$, which is the rate of execution of the given action with counterparts having those interface evaluations. Note that **get**/**qry** actions argument has been set to $T : t$ (rather than T as in Table 1) because the *labels* of **get**/**qry** transition will contain also the matching/retrieved term t. We do not explicitly give the definition of this function, as it is a parameter of the language[1]. Considering interface evaluations in the rate functions, together with the executed action, allow us to keep into account, in the computation of the action rate, various aspects depending on the component state such as the position/distance, as well as other time-dependent (i.e. knowledge state dependent) parameters such as the load of the component.

We also assume to have an *error function* $f_{\mathsf{err}} : \mathbb{E} \times Act \times \mathbb{E} \to [0, 1]$ that, similarly to the function $\mathcal{R}$, computes the probability of an error in the delivery of messages. Also for this function we do not explicitly give the definition, as it is a parameter of the language, but we expect it to be defined through an appropriate syntax.

Interface evaluations, the rate function $\mathcal{R}$, and the error function f_{err} have the roles described above also in the other semantics, presented in Sections 5, 6, and 7, so we avoid to mention and explain them again in those sections.

A further clarification we want to make in this premise concerns synchronization of input and output actions. As mentioned in Sec. 2, an input action α and an output action $\overline{\alpha}$ synchronize into a $\overleftrightarrow{\alpha}$ action. These synchronizations can happen within a component, for *local* actions (@self), between a source and a destination component, in *remote* **get**/**qry** actions (@p), and between a source component and all the remaining components, in *remote* **put** actions (@p). In the case of **get**/**qry** actions, the interface transmitted in the synchronization label will be that of the destination, which allows to have a different rate for each possible destination. In the case of **put** action, the interface transmitted in the synchronization label will be that of the source, since there are many possible destinations which are all involved (at once) in the computation. Therefore, for the **put** action, we abstract from the actual destinations and we compute the rate only on the basis of the source.

Finally, to simplify the synchronization of input and output actions, we force input actions to be *probabilistic*, and output actions to be *stochastic*. Therefore their composition can be performed through a simple multiplication.

[1]We plan, however, to extend the language with an appropriate syntax for defining the rate function in terms of sender/receiver component attributes and sent messages.

Inactive process:

$$\frac{}{\mathbf{nil} \overset{\alpha}{\rightharpoondown} []} \ (\textsc{nil})$$

Actions (where, $\mathbf{gq} \in \{\mathbf{get}, \mathbf{qry}\}$ and c is a Target):

$$\frac{\lambda = \mathcal{R}(\sigma, \mathbf{put}(t)@c, \_)}{\mathbf{put}(t)@c\,.\,P \overset{\overline{\mathbf{put}(t)@c}}{\rightharpoondown}_{\sigma} [P \mapsto \lambda]} \ (\textsc{put}) \qquad \frac{\alpha \neq \overline{\mathbf{put}(t)@c}}{\mathbf{put}(t)@c.P \overset{\alpha}{\rightharpoondown} []} \ (\textsc{put}_{\mathrm{B}})$$

$$\frac{\mathsf{match}(T,t) = \vartheta \quad \lambda = \mathcal{R}(\sigma, \mathbf{gq}(T:t)@c, \delta)}{\mathbf{gq}(T)@c\,.\,P \overset{\overline{\delta:\mathbf{gq}(T:t)@c}}{\rightharpoondown}_{\sigma} [P\vartheta \mapsto \lambda]} \ (\textsc{gq})$$

$$\frac{\neg\mathsf{match}(T,t)}{\mathbf{gq}(T)@c\,.\,P \overset{\overline{\_:\mathbf{gq}(T:t)@c}}{\rightharpoondown} []} \ (\textsc{gq}_{\mathrm{B1}}) \qquad \frac{\alpha \neq \overline{\_:\mathbf{gq}(T:t)@c}}{\mathbf{gq}(T)@c\,.\,P \overset{\alpha}{\rightharpoondown} []} \ (\textsc{gq}_{\mathrm{B2}})$$

Choice, definition, and parallel composition:

$$\frac{P \overset{\alpha}{\rightharpoondown}_e \mathscr{P} \quad Q \overset{\alpha}{\rightharpoondown}_e \mathscr{Q}}{P + Q \overset{\alpha}{\rightharpoondown}_e \mathscr{P} + \mathscr{Q}} \ (\textsc{cho}) \qquad \frac{A(\overrightarrow{x}) \overset{def}{=} P \quad P[\overrightarrow{v}/\overrightarrow{x}] \overset{\alpha}{\rightharpoondown}_e \mathscr{P}}{A(\overrightarrow{v}) \overset{\alpha}{\rightharpoondown}_e \mathscr{P}} \ (\textsc{def})$$

$$\frac{P \overset{\alpha}{\rightharpoondown}_e \mathscr{P} \quad Q \overset{\alpha}{\rightharpoondown}_e \mathscr{Q}}{P \mid Q \overset{\alpha}{\rightharpoondown}_e \mathscr{P} \mid (\mathcal{X}\, Q) + (\mathcal{X}\, P) \mid \mathscr{Q}} \ (\textsc{par})$$

Table 2: Operational semantics of StocS processes (act-or).

4.1 Operational semantics of processes

The act-or semantics of StocS *processes* is the RTS $(\mathit{Proc}, \mathit{Act}_{\mathit{Proc}}, \mathbb{R}_{\geq 0}, \rightharpoondown_e)$ where *Proc* is the set of process terms defined according to the syntax of StocS given in Table 1. The set $\mathit{Act}_{\mathit{Proc}}$ of labels is defined according to the grammar below (where e' is the evaluation of an interface, $t \in \mathbb{I}$, $T \in \mathbb{T}$, and c is a Target) and it is ranged over by $\alpha, \alpha', \ldots$:

$$\mathit{Act}_{\mathit{Proc}} \ ::= \ \overline{\mathbf{put}(t)@c} \ \mid \ \overline{e' : \mathbf{get}(T:t)@c} \ \mid \ \overline{e' : \mathbf{qry}(T:t)@c}$$

and $\rightharpoondown_e \subseteq \mathit{Proc} \times \mathit{Act}_{\mathit{Proc}} \times \mathbf{FTF}(\mathit{Proc}, \mathbb{R}_{\geq 0})$ is the least relation satisfying the rules of Table 2, which we describe in the rest of this section ($\rightharpoondown_e$ is parameterized by an interface evaluation e: we feel free to omit the parameter if unnecessary).

(nil) **nil** is the terminated process, since no process is reachable from it via any action.

(put) allows a process to execute an action **put** over the knowledge state of a target c with rate λ, which is computed by the rate function $\mathcal{R}$. The execution of a $\mathbf{put}(t)@c$ action depends on the source component and *all* the other componens in the system, which are involved as potential destinations. Consequently, the execution rate λ can be seen as a function of the action and of the source component only (whose interface evaluation is obtained as a parameter of the $\rightharpoondown$ relation); in particular, the action rate *does not* depend on (the interface evaluation of) a specific (destination) component (since the destination is not unique because **put** is a broadcast action); this is represented by using the symbol $\_$ in the destination argument of $\mathcal{R}$.

(gq) allows a process to execute a **get**/**qry** action over the knowledge state of a target c with rate λ, computed by the function $\mathcal{R}$ considering (i) the interface evaluation σ of the source component (obtained as a parameter of the $\rightharpoondown$ relation), (ii) the interface evaluation δ of the destination (received in the synchronization label), and (iii) the sent template T and the retrieved item t. The transition is labeled $\overline{\delta : \mathbf{gq}(T : t)@c}$, which indicates a request for a knowledge item t matching template T from a component identified by c. Note that, in the case of $\mathbf{gq}(T)@c$ the execution rate depends also on the destination interface evaluation because only one destination will be involved in the completion of the execution of the action.

(cho) accumulates the relevant rates by the application of the *sum* operator $+$ on the continuation $\mathscr{P}$ of P and $\mathscr{Q}$ of Q, according to the race condition principle of CTMCs;

(def) allows the instantiation of a process definition;

(par) realizes process parallel composition $P \mid Q$ and uses Def. 3.1, item (3) applied to the process parallel composition syntactic constructor $\mid$ (which is obviously injective).

Concerning the rule (par), given two functions $\mathscr{R}_1$ and $\mathscr{R}_2$, the function $\mathscr{R}_1 \mid \mathscr{R}_2$ applied to process term R returns the product $(\mathscr{R}_1\, R_1) \cdot (\mathscr{R}_2\, R_2)$, whenever R is of the form $R_1 \mid R_2$, for some terms R_1 and R_2, and 0 otherwise. In the rule, also the characteristic function $\mathcal{X}$ is used. Function $\mathscr{P} \mid (\mathcal{X} Q)$ applied to R returns $\mathscr{P}\, R'$ if $R = R' \mid Q$ for some R' and 0 otherwise; i.e. the function behaves as the continuation $\mathscr{P}$ of P for terms where Q does not progress (for one step). In conclusion, $\mathscr{P} \mid (\mathcal{X}\, Q) + (\mathcal{X}\, P) \mid \mathscr{Q}$ correctly represents process interleaving, keeping track of the relevant rates.

4.2 Operational semantics of components and systems

The act-or semantics of StocS *systems* is the RTS $(Sys, Act_{Sys}, \mathbb{R}_{\geq 0}, \rightarrow)$ where Sys is the set of system terms defined according to the syntax of StocS given in Table 1. The set Act_{Sys} of labels is defined according to the grammar below (where $\mathbf{gq} \in \{\mathbf{get}, \mathbf{qry}\}$, e' is the evaluation of an interface, $t \in \mathbb{I}$, $T \in \mathbb{T}$, and p is a Predicate):

$$\begin{array}{rclclcl} Act_{Sys} & ::= & e' : \mathbf{put}(t)@\mathsf{p} & \mid & e' : \mathbf{gq}(T : t)@\mathsf{p} & \mid & \text{(input actions)} \\ & & \overline{e' : \mathbf{put}(t)@\mathsf{p}} & \mid & \overline{e' : \mathbf{gq}(T : t)@\mathsf{p}} & \mid & \text{(output actions)} \\ & & \overleftrightarrow{e' : \mathbf{put}(t)@\mathsf{self}} & \mid & \overleftrightarrow{e' : \mathbf{gq}(T : t)@c} & & \text{(synchronizations)} \end{array}$$

and $\rightarrow \subseteq Sys \times Act_{Sys} \times \mathbf{FTF}(Sys, \mathbb{R}_{\geq 0})$ is the least relation satisfying the rules of Tables 3 and 4, where the process relation $\rightharpoondown_e$ defined in Table 2 is also used.

In Table 3, rules for components are grouped by action type:

(c-putl) allows a local process P to execute a local **put** action. Let $I\,[K,\,P]$ be a component; this rule states that P executes action $\mathbf{put}(t)@\mathsf{self}$ with local interface evaluation $\sigma = I(K)$ and evolves to $\mathscr{P}$, then a local execution of the action can occur and the entire component evolves with label $\overleftrightarrow{\sigma : \mathbf{put}(t)@\mathsf{self}}$ to $I[\pi, \mathscr{P}]$, where $\pi = K \oplus t$ is a probability distribution over the possible knowledge states obtained from K by adding the knowledge item t, while $I[\pi, \mathscr{P}]$ is the function which maps any term of the form $I\,[K,\,P]$ to $(\pi K) \cdot (\mathscr{P} P)$ and any other term to 0.

(c-puto) is used when the target of a **put** is not self but a predicate p. This rule simply lifts an output **put** action from the process level to the component level and transmits to its counterpart its current interface evaluation σ by including it in the transition label (which will be used in rule (c-puti) to compute the communication error probability).

put actions:

$$\frac{\sigma = I(K) \quad P \overset{\overline{\mathbf{put}(t)@\mathsf{self}}}{\longrightharpoonup}_{\sigma} \mathscr{P} \quad K \oplus t = \pi}{I\,[\,K,\,P\,] \xrightarrow{\overleftrightarrow{\sigma:\mathbf{put}(t)@\mathsf{self}}} I[\pi, \mathscr{P}]} \;(\textsc{c-putl})$$

$$\frac{\sigma = I(K) \quad P \overset{\overline{\mathbf{put}(t)@\mathsf{p}}}{\longrightharpoonup}_{\sigma} \mathscr{P}}{I\,[\,K,\,P\,] \xrightarrow{\overline{\sigma:\mathbf{put}(t)@\mathsf{p}}} I[(\mathcal{X}K), \mathscr{P}]} \;(\textsc{c-puto})$$

$$\frac{\delta = I(K) \quad \delta \models \mathsf{p} \quad K \oplus t = \pi \quad p_{\mathsf{err}} = f_{\mathsf{err}}(\sigma, \mathbf{put}(t)@\mathsf{p}, \delta)}{I\,[\,K,\,P\,] \xrightarrow{\sigma:\mathbf{put}(t)@\mathsf{p}} [\,I\,[\,K,\,P\,] \mapsto p_{\mathsf{err}}\,] + I[\pi, (\mathcal{X}P)] \cdot (1 - p_{\mathsf{err}})} \;(\textsc{c-puti})$$

$$\frac{I(K) \not\models \mathsf{p}}{I\,[\,K,\,P\,] \xrightarrow{\sigma:\mathbf{put}(t)@\mathsf{p}} [\,I\,[\,K,\,P\,] \mapsto 1\,]} \;(\textsc{c-putir})$$

get/**qry** actions (where, $\mathbf{gq} \in \{\mathbf{get}, \mathbf{qry}\}$):

$$\frac{\sigma = I(K) \quad P \overset{\overline{\sigma:\mathbf{get}(T{:}t)@\mathsf{self}}}{\longrightharpoonup}_{\sigma} \mathscr{P} \quad K \ominus T = \pi}{I\,[\,K,\,P\,] \xrightarrow{\overleftrightarrow{\sigma:\mathbf{get}(T{:}t)@\mathsf{self}}} I[\pi(t), \mathscr{P}]} \;(\textsc{c-getl}) \qquad \frac{K \ominus T = \bot}{I\,[\,K,\,P\,] \xrightarrow{\overleftrightarrow{\sigma:\mathbf{get}(T{:}t)@\mathsf{self}}} []} \;(\textsc{c-getl}_{\mathrm{B}})$$

$$\frac{\sigma = I(K) \quad P \overset{\overline{\sigma:\mathbf{qry}(T{:}t)@\mathsf{self}}}{\longrightharpoonup}_{\sigma} \mathscr{P} \quad K \vdash T = \pi}{I\,[\,K,\,P\,] \xrightarrow{\overleftrightarrow{\sigma:\mathbf{qry}(T{:}t)@\mathsf{self}}} I[(\mathcal{X}K) \cdot \pi(t), \mathscr{P}]} \;(\textsc{c-qryl}) \qquad \frac{K \vdash T = \bot}{I\,[\,K,\,P\,] \xrightarrow{\overleftrightarrow{\sigma:\mathbf{qry}(T{:}t)@\mathsf{self}}} []} \;(\textsc{c-qryl}_{\mathrm{B}})$$

$$\frac{\sigma = I(K) \quad P \overset{\overline{\delta:\mathbf{gq}(T{:}t)@\mathsf{p}}}{\longrightharpoonup}_{\sigma} \mathscr{P}}{I\,[\,K,\,P\,] \xrightarrow{\overline{\delta:\mathbf{gq}(T{:}t)@\mathsf{p}}} I[K, \mathscr{P}]} \;(\textsc{c-gqo})$$

$$\frac{\delta = I(K) \quad \delta \models \mathsf{p} \quad K \ominus T = \pi}{I\,[\,K,\,P\,] \xrightarrow{\delta:\mathbf{get}(T{:}t)@\mathsf{p}} I[\pi(t), (\mathcal{X}P)]} \;(\textsc{c-geti}) \qquad \frac{\delta \neq I(K) \;\vee\; I(K) \not\models \mathsf{p} \;\vee\; K \ominus T = \bot}{I\,[\,K,\,P\,] \xrightarrow{\delta:\mathbf{get}(T{:}t)@\mathsf{p}} []} \;(\textsc{c-geti}_{\mathrm{B}})$$

$$\frac{\delta = I(K) \quad \delta \models \mathsf{p} \quad K \vdash T = \pi}{I\,[\,K,\,P\,] \xrightarrow{\delta:\mathbf{qry}(T{:}t)@\mathsf{p}} [\,I\,[\,K,\,P\,] \mapsto \pi(t)]} \;(\textsc{c-qryi}) \qquad \frac{\delta \neq I(K) \;\vee\; I(K) \not\models \mathsf{p} \;\vee\; K \vdash T = \bot}{I\,[\,K,\,P\,] \xrightarrow{\delta:\mathbf{qry}(T{:}t)@\mathsf{p}} []} \;(\textsc{c-qryi}_{\mathrm{B}})$$

Table 3: Operational semantics of StocS components (act-or).

(c-puti) models a component *accepting* a **put**. The rule is applied when the component satisfies the predicate p. When predicate p is satisfied by $I(K)$ and $K \oplus t = \pi$, component $I\,[\,K,\,P\,]$ evolves to $[\,I\,[\,K,\,P\,] \mapsto p_{\mathsf{err}}\,] + I[\pi, (\mathcal{X}P)] \cdot (1 - p_{\mathsf{err}})$. The first term, that is selected with probability p_{err}, models a failure in the action execution, for instance due to a communication error. Value p_{err} is computed by the function f_{err} taking into account the source, the action performed, and the destination. The second term identifies the different configurations the component can reach when knowledge item t is added to the knowledge repository K.

(c-putir) accepts an input **put** action producing no effect, under the assumption that the predicate p is not satisfied (the component allows synchronization as required by the broadcast scheme).

(c-getl)/(c-qryl) realize the local **get** (**qry**) action retrieving an item $t \in \mathbb{I}$ matching the pattern T, if possible, in the execution of process P (in the label of the process action we include the item t and we include the interface evaluation σ of the component for computing the action rate) and, since the **get** (**qry**) action may result in several distinct knowledge bases, these need to be summed together considering all possibilities: π is a distribution over pairs (knowledge base and knowledge item) and the possible components in the continuation are weighted by using π;

(c-gqo) lifts a **get**/**qry** output action to the level of the component, similarly to (c-puto), and produces a continuation computed by the $\rightharpoondown$ relation (in rule gq) using the source interface evaluation (σ) and the destination interface evaluation (δ, taken from the synchronization label);

(c-geti)/(c-qryi) under the condition that a component satisfies the predicate p, allow the component to accept an input **get** (**qry**) action (respectively), by computing the continuation on the basis of the distribution π over possible pairs of knowledge state and retrieved item (retrieved item, respectively), computed by using the $\ominus$ ($\vdash$, respectively) operator, but they do not compute any rate as this is computed on the source side by rules (gq/c-gqo); notice that the component communicates its interface evaluation δ (destination) in the synchronization label.

It is worth noting that rules (c-puti) and (c-putir) deal with probability only. In fact, the actual rate of the action is the one which will result from system synchronization (Rules (s-po) and (s-pi) in Table 4) on the basis of the rates settled by the rule (put) of Table 2.

Concerning (c-putl), (c-puti), (c-getl), and (c-geti), **put**/**get** actions may result in several distinct knowledge states and the rate functions need to be summed together considering all possible knowledge states. This is addressed by using Def. 3.1, item (3) applied to the component syntactic constructor $I[\,\cdot\,,\cdot\,]$, which is obviously injective. Thus, for $\pi \in \mathsf{Dist}(\mathbb{K})$ and $\mathscr{P} \in \mathbf{FTF}(\mathit{Proc}, \mathbb{R}_{\geq 0})$, function $I[\pi, \mathscr{P}] \in \mathbf{FTF}(\mathit{Sys}, \mathbb{R}_{\geq 0})$ returns $(\mathscr{P}\, P)\cdot\pi(K)$ on systems $I\,[\,K,\, P]$, for any P, and 0 on any other system term. On the contrary, when we consider rules (c-qryl) and (c-qryi), even though **qry** may return several retrieved knowledge items, the actual transition fixes a precise item t, whose retrieval probability $\pi(t)$ is combined with the component continuation.

Let us now consider systems of components: similarly to $|$ in the case of components, also the definition of the semantics of system parallel composition $S_1 \parallel S_2$ uses Def. 3.1, item (3) applied to the system parallel composition syntactic constructor $\parallel$, which is obviously injective. As usual, interleaving is modeled as a combination of lifted $\parallel$, $+$ on functions and the characteristic function. In Table 4 we give the rules for systems, grouped by action type:

(s-po)/(s-pi) realize the broadcast communication of **put**: (s-po) ensures that if any subsystem executes an output **put** action (i.e. it executes a σ : $\mathbf{put}(t)@\mathsf{p}$ labeled transition), then the remaining subsystem must execute the corresponding input **put** action (i.e. it should execute a σ : $\mathbf{put}(t)@\mathsf{p}$ labeled transition). the composed system does not exhibit a synchronization label, but it rather propagates the output σ : $\mathbf{put}(t)@\mathsf{p}$ to allow further synchronization with all the other components in parallel; in the computation of the final rate it is necessary to consider output on the left sub-system and input on the right as well as the symmetric case. Rule (s-pi) ensures that all subcomponents of a system synchronize, all together, on a (specific) input **put** action, completing the broadcast communication. Note that each component is constantly enabled on the input label for any **put** action (rules (c-puti) and (c-putir).

(s-gqs) realizes one-to-one synchronization of **get**/**qry** actions (which are not broadcast), denoted by a $\overleftrightarrow{e : \mathbf{gq}(T : t)@\mathsf{p}}$ label, and performs aggregation of: (1) the synchronization rate of the left (right) sub-system, with the right (resp. left) subsystem that must

put synchronization:

$$\frac{S_1 \xrightarrow{\overline{\sigma : \mathbf{put}(t)@\mathsf{p}}} \mathscr{S}_1^o \quad S_1 \xrightarrow{\sigma : \mathbf{put}(t)@\mathsf{p}} \mathscr{S}_1^i \quad S_2 \xrightarrow{\overline{\sigma : \mathbf{put}(t)@\mathsf{p}}} \mathscr{S}_2^o \quad S_2 \xrightarrow{\sigma : \mathbf{put}(t)@\mathsf{p}} \mathscr{S}_2^i}{S_1 \parallel S_2 \xrightarrow{\overline{\sigma : \mathbf{put}(t)@\mathsf{p}}} \mathscr{S}_1^o \parallel \mathscr{S}_2^i + \mathscr{S}_1^i \parallel \mathscr{S}_2^o} \quad (\textsc{s-po})$$

$$\frac{S_1 \xrightarrow{\sigma : \mathbf{put}(t)@\mathsf{p}} \mathscr{S}_1 \quad S_2 \xrightarrow{\sigma : \mathbf{put}(t)@\mathsf{p}} \mathscr{S}_2}{S_1 \parallel S_2 \xrightarrow{\sigma : \mathbf{put}(t)@\mathsf{p}} \mathscr{S}_1 \parallel \mathscr{S}_2} \quad (\textsc{s-pi})$$

get/**qry** synchronization ($\mathbf{gq} \in \{\mathbf{get}, \mathbf{qry}\}$):

$$\frac{\begin{array}{c} S_1 \xrightarrow{\overleftrightarrow{\delta : \mathbf{gq}(T{:}t)@\mathsf{p}}} \mathscr{S}_1^s \quad S_1 \xrightarrow{\overline{\delta : \mathbf{gq}(T{:}t)@\mathsf{p}}} \mathscr{S}_1^o \quad S_1 \xrightarrow{\delta : \mathbf{gq}(T{:}t)@\mathsf{p}} \mathscr{S}_1^i \\ S_2 \xrightarrow{\overleftrightarrow{\delta : \mathbf{gq}(T{:}t)@\mathsf{p}}} \mathscr{S}_2^s \quad S_2 \xrightarrow{\overline{\delta : \mathbf{gq}(T{:}t)@\mathsf{p}}} \mathscr{S}_2^o \quad S_2 \xrightarrow{\delta : \mathbf{gq}(T{:}t)@\mathsf{p}} \mathscr{S}_2^i \end{array}}{S_1 \parallel S_2 \xrightarrow{\overleftrightarrow{\delta : \mathbf{gq}(T{:}t)@\mathsf{p}}} \mathscr{S}_1^s \parallel (\mathcal{X}\, S_2) + \mathscr{S}_1^o \parallel \mathscr{S}_2^i + \mathscr{S}_1^i \parallel \mathscr{S}_2^o + (\mathcal{X}\, S_1) \parallel \mathscr{S}_2^s} \quad (\textsc{s-gqs})$$

$$\frac{S_1 \xrightarrow{\delta : \mathbf{gq}(T{:}t)@\mathsf{p}} \mathscr{S}_1 \quad S_2 \xrightarrow{\delta : \mathbf{gq}(T{:}t)@\mathsf{p}} \mathscr{S}_2}{S_1 \parallel S_2 \xrightarrow{\delta : \mathbf{gq}(T{:}t)@\mathsf{p}} \mathscr{S}_1 \parallel (\mathcal{X}\, S_2) + (\mathcal{X}\, S_1) \parallel \mathscr{S}_2} \quad (\textsc{s-gqi})$$

Internal actions ($\mathbf{gq} \in \{\mathbf{get}, \mathbf{qry}\}$):

$$\frac{S_1 \xrightarrow{\overleftrightarrow{e : \mathbf{put}(t)@\mathsf{self}}} \mathscr{S}_1 \quad S_2 \xrightarrow{\overleftrightarrow{e : \mathbf{put}(t)@\mathsf{self}}} \mathscr{S}_2}{S_1 \parallel S_2 \xrightarrow{\overleftrightarrow{e : \mathbf{put}(t)@\mathsf{self}}} \mathscr{S}_1 \parallel (\mathcal{X}\, S_2) + (\mathcal{X}\, S_1) \parallel \mathscr{S}_2} \quad (\textsc{s-spl})$$

$$\frac{S_1 \xrightarrow{\overleftrightarrow{e : \mathbf{gq}(T{:}t)@\mathsf{self}}} \mathscr{S}_1 \quad S_2 \xrightarrow{\overleftrightarrow{e : \mathbf{gq}(T{:}t)@\mathsf{self}}} \mathscr{S}_2}{S_1 \parallel S_2 \xrightarrow{\overleftrightarrow{e : \mathbf{gq}(T{:}t)@\mathsf{self}}} \mathscr{S}_1 \parallel (\mathcal{X}\, S_2) + (\mathcal{X}\, S_1) \parallel \mathscr{S}_2} \quad (\textsc{s-sgql})$$

Table 4: Operational semantics of StocS systems (act-or).

not progress (this is realized using the $\mathcal{X}$ characteristic function), and (2) output rates of the left sub-system and input rates of the right subsystem (as well as the symmetric case), combined with the $\parallel$ operator.

(s-gqi) realizes the input **get** (resp. **qry**) action for systems in which *one* component, among those satisfying the target predicate and having a matching knowledge item, can answer.

(s-spl/s-sgql) allow a system to execute a local **put**/**get**/**qry** action and exposes the label denoting the type of action to allow appropriate aggregation of the observed rates.

As a final observation, we want to underline that the interleaving semantics for concurrency is realized, for **put** actions, by rule (s-po) and, for **get**/**qry** actions, by rule (s-gqs).

5 Interaction-oriented Operational Semantics

In this section we illustrate the Interaction-oriented semantics (abbreviated to int-or), which is based on the idea of having different semantics for local and remote actions (as defined in

Section 4). In particular, remote actions maintain their stochastic semantics defined in ACT-OR, while local actions are instantaneous and their non-determinism is resolved by relying on probability distributions computed by applying the knowledge base operators $\oplus$, $\ominus$, and $\vdash$. Since we are interested in obtaining a CTMC semantics also for this variant of StocS, we require that the execution of a remote action α_R by a component C includes the execution of *all* the subsequent local actions $\alpha_{L_1} \cdots \alpha_{L_n}$ in the behavior of C, until a new remote action or a **nil** is found, by applying a sort of maximal execution [10]. As a consequence, we consider as *atomic* the execution of $\alpha_R \alpha_{L_1} \cdots \alpha_{L_n}$ by the component C. To this end, we impose restrictions on process syntax that avoid the possibility of realizing *infinite* sequences of local actions. That is, sequences of local actions are always finite and either terminate in a **nil** or in a remote action. The stochastic semantics of the (atomic) sequence $\alpha_R \alpha_{L_1} \cdots \alpha_{L_n}$ of actions is obtained by appropriate composition of the ACT-OR stochastic semantics of α_R with the *probabilistic* semantics of $\alpha_{L_1} \cdots \alpha_{L_n}$, which we shall define in the rest of this section. For this purpose, we will use the notion of *convolution*.

Convolution. Given a relation $\circ \subseteq X \times \mathbf{FTF}(X, \mathbb{R}_{\geq 0})$, we define the convolution $\circ^{cv} \subseteq \mathbf{FTF}(X, \mathbb{R}_{\geq 0}) \times \mathbf{FTF}(X, \mathbb{R}_{\geq 0})$ as follows:

$$\delta_1 \circ^{cv} \delta_2 \quad \text{iff} \quad \forall x \in X \; \exists \delta_x \in \mathsf{SubDist}(X) \; (\; x \circ \delta_x \; \wedge \; \delta_2 = \sum_{y \in X} \delta_x \cdot \delta_1(y) \;)$$

Example 3. *Consider a sub-probability distribution* $\delta_1 = [a \mapsto 0.2, b \mapsto 0.3, c \mapsto 0.5]$ *and a relation* $\circ = \{(a, [a \mapsto 0.4, b \mapsto 0.6]), (b, [b \mapsto 0.5, c \mapsto 0.5]), (b, [a \mapsto 0.3, b \mapsto 0.7]), (c, [a \mapsto 0.8, c \mapsto 0.2])\}$ *on a set* $X = \{a, b, c\}$. *Then,* $\delta_1 \circ^{cv} \delta_2$ *for* $\delta_2 = [a \mapsto 0.48, b \mapsto 0.27, c \mapsto 0.25]$ *and also* $\delta_1 \circ^{cv} \delta_3$ *for* $\delta_3 = [a \mapsto 0.57, b \mapsto 0.33, c \mapsto 0.1]$.

Before describing the details of the semantics, let us now introduce informally a bit of notation concerning transition relations. We distinguish between (i) *remote-enabled* process states, that have *only* remote (@p) actions activated, *if any*, and (ii) *local-enabled* process states that consist of a non-empty sequence of local actions (@self) followed by a remote-enabled process state. At the process level, we change the ACT-OR transition relation $\rightharpoondown_e$ as follows. First, we limit the applicability of the process relation $\rightharpoondown_e$ to labels pertaining to *remote* actions, so that it applies only to remote-enabled process states. Then, we define an additional transition relation $\underset{\mathrm{L}}{\rightharpoondown}$ for executing local actions (@self). At the component level, the $\rightarrow$ relation is modified as follows: (1) $\rightarrow$ is restricted to remote actions, (2) input remote actions remain as in ACT-OR while output remote actions semantics is changed, and (3) local actions are *not* allowed by $\rightarrow$. In order to execute local actions, we introduce a transition relation $\underset{\mathrm{L}}{\rightarrow}$, which essentially lifts the relation $\underset{\mathrm{L}}{\rightharpoondown}$ to the level of components. For realizing the maximal execution of the $\alpha_{L_1} \cdots \alpha_{L_n}$ action sequence, we introduce the relation $\underset{\mathrm{SL}}{\rightarrow}$ which concatenates local transitions, executed by using $\underset{\mathrm{L}}{\rightarrow}$, until no local action remains to be executed. In this concatenation, the convolution operator $\cdot^{cv}$ is used to lift $\underset{\mathrm{SL}}{\rightarrow}$ from a component-to-rate-function relation to a rate-function-to-rate-function relation $\underset{\mathrm{SL}}{\rightarrow}^{cv}$. We discuss this aspect in more details in the rest of this section. The final step to define the new semantics of the $\rightarrow$ relation for components is that each sequence $\alpha_R \alpha_{L_1} \cdots \alpha_{L_n}$ is executed by appropriate composition of the stochastic transition relation $\rightharpoondown_e$, for the execution of the remote action α_R, and the probabilistic transition relation $\underset{\mathrm{SL}}{\rightarrow}$, for the execution of the sequence $\alpha_{L_1} \cdots \alpha_{L_n}$ of local actions. Note that if any of the α_{L_i} cannot be executed, then the entire sequence $\alpha_R \alpha_{L_1} \cdots \alpha_{L_n}$ is not executed.

To complete this informal introduction to the ideas underlying the INT-OR semantics we mention that the new $\rightarrow$ relation for components is designed to be applied only to components with a remote-enabled process state. The effect of its application is to obtain a new component with a remote-enabled process state (so it is remote-enabled-preserving). A sequence $\alpha_R \alpha_{L_1} \cdots \alpha_{L_n}$ is, therefore, executed entirely in an atomic way, and the label of $\rightarrow$ will be just the remote action α_R, thus hiding the sequence of local actions performed.

In the rest of this section we formalize these ideas. For the sake of conciseness, we show only the modifications to the Action-oriented semantics that are necessary to implement the Interaction-oriented semantics. In particular, in Table 6 we omit the rules that are unchanged w.r.t Table 3 and we just mention their labels. Tables 2 and 4 are unchanged so we omit them.

5.1 Operational semantics of processes

We refine the syntactic category of processes provided in Table 1 by imposing constraints on process syntax. In particular, the syntactic category R below defines the remote-enabled process states, that have only remote ($@\mathsf{p}$) actions activated, if any, and include **nil**. The syntactic category L below defines the local-enabled process states, that are made of a non-empty sequence of local ($@\mathsf{self}$) actions followed by a remote-enabled process state. We also consider the category of *mixed* processes (M), constructed by parallel composition of local- and remote-enabled processes. This is a syntactic category that will appear in computations as an intermediate product of the semantics. We do not allow to directly specify components behavior using mixed processes. In particular we require that, for the INT-OR semantics, components have their processes specified according to the *remote* syntactic category only.

In this section we put special attention on the use of names for variables ranging on processes. In particular, R,R_1,…,R',… are variables ranging over remote-enabled processes, L,L_1,…,L',… are variables ranging over local-enabled processes, and M,M_1,…,M',… are variables ranging over mixed processes.

$$\begin{array}{lcl} \mathrm{R} & ::= & \mathbf{nil} \;\big|\; \mathbf{act}(T)@\mathsf{p}.L \;\big|\; \mathbf{act}(T)@\mathsf{p}.R \;\big|\; R + R \;\big|\; R \mid R \;\big|\; X \;\big|\; A(\bar{p}) \\ \mathrm{L} & ::= & \mathbf{act}(T)@\mathsf{self}.L \;\big|\; \mathbf{act}(T)@\mathsf{self}.R \\ \mathrm{M} & ::= & L \;\big|\; R \;\big|\; M \mid M \end{array}$$

Furthermore, process definitions must use remote-enabled processes on their right-hand sides.

The INT-OR semantics of StocS processes is the RTS $(\mathit{Proc}, \mathit{Act}_{\mathit{Proc}}, \mathbb{R}_{\geq 0}, \rightharpoonup_e \cup \underset{\mathrm{L}}{\rightharpoonup})$, where Proc is the set of process terms defined according to the syntax of StocS given in Table 1 with the above syntactic restrictions. The set $\mathit{Act}_{\mathit{Proc}}$ of process actions labels is partitioned into the two sets $\mathit{Act}^R_{\mathit{Proc}}$ and $\mathit{Act}^L_{\mathit{Proc}}$ of remote and local process action labels, respectively. Those two sets are defined according to the grammar below (where $t \in \mathbb{I}$, $T \in \mathbb{T}$, c is a TARGET, p is a PREDICATE, and e is the evaluation of an interface):

$$\begin{array}{lcl} \mathit{Act}^R_{\mathit{Proc}} & ::= & \overline{e : \mathbf{put}(t)@\mathsf{p}} \;\big|\; \overline{e : \mathbf{get}(T : t)@\mathsf{p}} \;\big|\; \overline{e : \mathbf{qry}(T : t)@\mathsf{p}} \\ \mathit{Act}^L_{\mathit{Proc}} & ::= & \mathbf{put}(t)@\mathsf{self} \;\big|\; \mathbf{get}(T)@\mathsf{self} \;\big|\; \mathbf{qry}(T)@\mathsf{self} \;\big|\; \surd \end{array}$$

The transition relation $\rightharpoonup_e \subseteq \mathit{Proc} \times \mathit{Act}^R_{\mathit{Proc}} \times \mathbf{FTF}(\mathit{Proc}, \mathbb{R}_{\geq 0})$ is the least relation satisfying the rules of Table 2 (note that here *we restrict the action labels* to disallow actions $@\mathsf{self}$ on $\rightharpoonup_e$). It uses the auxiliary relation $\underset{\mathrm{L}}{\rightharpoonup} \subseteq \mathit{Proc} \times \mathit{Act}^L_{\mathit{Proc}} \times \mathbf{FTF}(\mathit{Proc}, \mathbb{R}_{\geq 0})$, which is the the least relation satisfying the rules of Table 5 discussed in the following:

(TPUT/TGQ/TCHO/TPARL) execute the $\surd$ action that has the role of allowing local-enabled processes to progress in parallel with remote-enabled and mixed ones, within the (TPAR) rule;

(PUTL/GQL) allow local-enabled processes to execute local actions with rate 1;

Tick ($\surd$ is the tick label, $\mathbf{gq}\in\{\mathbf{get},\mathbf{qry}\}$, and p is a Predicate):

$$\frac{}{\mathbf{put}(t)@\mathsf{p}\,.\,P \xrightarrow[\mathrm{L}]{\surd} [\mathbf{put}(t)@\mathsf{p}\,.\,P \mapsto 1]}\ (\textsc{tput}) \qquad \frac{\alpha \neq \surd}{\mathbf{put}(t)@\mathsf{p}\,.\,P \xrightarrow[\mathrm{L}]{\alpha} []}\ (\textsc{tput}_{\mathrm{B}})$$

$$\frac{}{\mathbf{gq}(T)@\mathsf{p}\,.\,P \xrightarrow[\mathrm{L}]{\surd} [\mathbf{gq}(T)@\mathsf{p}\,.\,P \mapsto 1]}\ (\textsc{tgq}) \qquad \frac{\alpha \neq \surd}{\mathbf{gq}(T)@\mathsf{p}\,.\,P \xrightarrow[\mathrm{L}]{\alpha} []}\ (\textsc{tgq}_{\mathrm{B}})$$

$$\frac{}{R_1 + R_2 \xrightarrow[\mathrm{L}]{\surd} [R_1 + R_2 \mapsto 1]}\ (\textsc{tcho}) \qquad \frac{\alpha \neq \surd}{R_1 + R_2 \xrightarrow[\mathrm{L}]{\alpha} []}\ (\textsc{tcho}_{\mathrm{B}})$$

$$\frac{R_1 \xrightarrow[\mathrm{L}]{\surd} \mathscr{R}_1 \quad R_2 \xrightarrow[\mathrm{L}]{\surd} \mathscr{R}_2}{R_1 \mid R_2 \xrightarrow[\mathrm{L}]{\surd} \mathscr{R}_1 \mid \mathscr{R}_2}\ (\textsc{tparl})$$

Actions (where, $\mathbf{gq}\in\{\mathbf{get},\mathbf{qry}\}$, c is a Target, and p is a Predicate):

$$\frac{}{\mathbf{put}(t)@\mathsf{self}\,.\,P \xrightarrow[\mathrm{L}]{\mathbf{put}(t)@\mathsf{self}} [P \mapsto 1]}\ (\textsc{putl}) \qquad \frac{\alpha \neq \mathbf{put}(t)@\mathsf{self}}{\mathbf{put}(t)@\mathsf{self}\,.\,P \xrightarrow[\mathrm{L}]{\alpha} []}\ (\textsc{putl}_{\mathrm{B}})$$

$$\frac{}{\mathbf{gq}(T)@\mathsf{self}\,.\,P \xrightarrow[\mathrm{L}]{\mathbf{gq}(T:t)@\mathsf{self}} [P \mapsto 1]}\ (\textsc{gql}) \qquad \frac{\alpha \neq \mathbf{gq}(T:t)@\mathsf{self}}{\mathbf{gq}(T)@\mathsf{self}\,.\,P \xrightarrow[\mathrm{L}]{\alpha} []}\ (\textsc{gql}_{\mathrm{B}})$$

Choice, definition, and parallel composition:

$$\frac{M_1 \xrightarrow[\mathrm{L}]{\alpha} \mathscr{M}_1 \quad M_1 \xrightarrow[\mathrm{L}]{\surd} \mathscr{M}_1^{\surd} \quad M_2 \xrightarrow[\mathrm{L}]{\alpha} \mathscr{M}_2 \quad M_2 \xrightarrow[\mathrm{L}]{\surd} \mathscr{M}_2^{\surd} \quad \alpha \neq \surd}{M_1 \mid M_2 \xrightarrow[\mathrm{L}]{\alpha} \mathscr{M}_1 \mid \mathscr{M}_2^{\surd} + \mathscr{M}_1^{\surd} \mid \mathscr{M}_2}\ (\textsc{tpar})$$

Table 5: Operational semantics of StocS processes (int-or), $\xrightarrow[\mathrm{L}]{}$ relation only.

(tpar) applies to mixed processes and is used to postpone the execution of remote-enabled processes by allowing only the execution of local non-$\surd$ actions, indeed $\surd$ actions are used to keep non-local processes unchanged while local ones progress (recall that $\xrightarrow[\mathrm{L}]{}$ is defined on Act^{L}_{Proc} actions).

5.2 Operational semantics of components and systems

The int-or semantics of StocS systems is the RTS $(Sys, Act_{Sys}, \mathbb{R}_{\geq 0}, \rightarrow)$ where Sys is the set of system terms defined according to the syntax of StocS given in Table 1. The set Act_{Sys} of labels is partitioned into the two sets Act^{L}_{Sys} and Act^{R}_{Sys}, defined according to the grammar below (where $\mathbf{gq}\in\{\mathbf{get},\mathbf{qry}\}$, e' is the evaluation of an interface, $t \in \mathbb{I}$, $T \in \mathbb{T}$, and p is a Predicate) and it is ranged over by $\alpha, \alpha', \ldots$:

$$\begin{array}{rlll} Act^{R}_{Sys} & ::= & e' : \mathbf{put}(t)@\mathsf{p} \ \mid \ e' : \mathbf{gq}(T:t)@\mathsf{p} \ \mid & \text{(input actions)} \\ & & \overline{e' : \mathbf{put}(t)@\mathsf{p}} \ \mid \ \overline{e' : \mathbf{gq}(T:t)@\mathsf{p}} \ \mid & \text{(output actions)} \\ & & \overleftrightarrow{e' : \mathbf{gq}(T:t)@\mathsf{p}} & \text{(synchronizations)} \end{array}$$

put actions (R is a remote-enabled process):

$$\frac{\sigma = I(K) \quad R \xrightarrow{\overline{\mathbf{put}(t)@\mathsf{p}}}_{\sigma} \mathscr{R} \quad I[K,\mathscr{R}] \xrightarrow[\text{SL}]{} \mathscr{C}}{I[K,R] \xrightarrow{\overline{\sigma : \mathbf{put}(t)@\mathsf{p}}} \mathscr{C}} \ (\textsc{c-puto})$$

($\textsc{c-puti}$, Table 3) ($\textsc{c-putir}$, Table 3)

get/qry actions (R is a remote-enabled process):

$$\frac{\sigma = I(K) \quad R \xrightarrow{\overline{\delta : \mathbf{gq}(T{:}t)@\mathsf{p}}}_{\sigma} \mathscr{R} \quad I[K,\mathscr{R}] \xrightarrow[\text{SL}]{} \mathscr{C}}{I[K,R] \xrightarrow{\overline{\delta : \mathbf{gq}(T{:}t)@\mathsf{p}}} \mathscr{C}} \ (\textsc{c-gqo})$$

($\textsc{c-geti}$, Table 3) ($\textsc{c-geti}_\text{B}$, Table 3) ($\textsc{c-qryi}$, Table 3) ($\textsc{c-qryi}_\text{B}$, Table 3)

Table 6: Operational semantics of StocS components (INT-OR).

put local actions (M is a mixed process):

$$\frac{M \xrightarrow[\text{L}]{\mathbf{put}(t)@\mathsf{self}} \mathscr{M} \quad K \oplus t = \pi}{I[K,M] \xrightarrow[\text{L}]{\mathbf{put}(t)@\mathsf{self}} I[\pi, \mathscr{M}]} \ (\textsc{c-putl})$$

get/qry local actions (M is a mixed process):

$$\frac{M \xrightarrow[\text{L}]{\mathbf{get}(T)@\mathsf{self}} \mathscr{M} \quad K \ominus T = \pi}{I[K,M] \xrightarrow[\text{L}]{\mathbf{get}(T)@\mathsf{self}} \sum_{t \in \{t \,|\, match(T,t)=\vartheta\}} I[\pi(t), \mathscr{M}\vartheta]} \ (\textsc{c-getl})$$

$$\frac{M \xrightarrow[\text{L}]{\mathbf{qry}(T)@\mathsf{self}} \mathscr{M} \quad K \vdash T = \pi}{I[K,M] \xrightarrow[\text{L}]{\mathbf{qry}(T)@\mathsf{self}} \sum_{t \in \{t \,|\, match(T,t)=\vartheta\}} I[(\mathcal{X}K) \cdot \pi(t), \mathscr{M}\vartheta]} \ (\textsc{c-qryl})$$

$$\frac{K \ominus T = \bot}{I[K,M] \xrightarrow[\text{L}]{\mathbf{get}(T)@\mathsf{self}} []} \ (\textsc{c-getl}_\text{B}) \qquad \frac{K \vdash T = \bot}{I[K,M] \xrightarrow[\text{L}]{\mathbf{qry}(T)@\mathsf{self}} []} \ (\textsc{c-qryl}_\text{B})$$

Maximal execution of local actions:

$$\frac{}{I[K,R] \xrightarrow[\text{SL}]{} [I[K,R] \mapsto 1]} \ (\textsc{c-rmt}) \quad R \text{ is a remote-enabled process}$$

$$\frac{I[K,M] \xrightarrow[\text{L}]{\alpha} \mathscr{C} \quad \mathscr{C} \xrightarrow[\text{SL}]{}^{cv} \mathscr{D}}{I[K,M] \xrightarrow[\text{SL}]{} \mathscr{D}} \ (\textsc{c-mxt})$$

M is a mixed process, $\alpha \in Act^L_{Sys}$, and $\xrightarrow[\text{SL}]{}^{cv}$ is the convolution of $\xrightarrow[\text{SL}]{}$

Table 7: Operational semantics of StocS components (INT-OR), $\xrightarrow[\text{L}]{}$ and $\xrightarrow[\text{SL}]{}$ transition relations.

$$Act^L_{Sys} \ ::= \ \mathbf{put}(t)@\mathsf{self} \ \big| \ \mathbf{get}(T)@\mathsf{self} \ \big| \ \mathbf{qry}(T)@\mathsf{self}$$

The transition relation $\rightarrow \subseteq Sys \times Act^R_{Sys} \times \mathbf{FTF}(Sys, \mathbb{R}_{\geq 0})$ for components is the least relation satisfying the rules of Tables 6 and 4. Note that it does not allow to execute local actions. Indeed, these are executed during the execution of a remote action in an atomic way, according to the idea discussed at the beginning of this section. The process transition relation $\rightharpoondown_e \subseteq Proc \times Act^R_{Proc} \times \mathbf{FTF}(Proc, \mathbb{R}_{\geq 0})$ is defined in Table 2 and, again, only allows the execution of remote actions. The transition relation $\underset{\text{SL}}{\rightarrow} \subseteq Comp \times \mathsf{Dist}(Comp)$ is defined only on the components syntactic category $Comp$ and on the set of local actions Act^L_{Sys} by rules (c-rmt/c-mxt) in Table 7 and it allows the maximal execution of local actions within a component with mixed process state. Let us now discuss the rules in Table 6.

(c-puto) applies to a remote-enabled process (R) and realizes maximal execution of a remote **put** operation by: (i) applying a stochastic **put** transition by using the $\rightharpoondown$ relation, (ii) applying local transitions as much as possible by using the $\underset{\text{SL}}{\rightarrow}$ relation, and (iii) sending the interface evaluation σ of the source component (itself) in the synchronization label, all this within the atomic component transition $\xrightarrow{\overline{\sigma : \mathbf{put}(t)@\mathsf{p}}}$, note that for maximal execution of local actions we consider the entire component, rather than only the behavior process, because we need to take into account also the knowledge state;

(c-gqo) applies to a remote-enabled process (R) and realizes maximal execution of a remote **get**/**qry** action by: (i) applying a stochastic **get**/**qry** transition by using the $\rightharpoondown$ relation, (ii) applying local transitions as much as possible by using the $\underset{\text{SL}}{\rightarrow}$ relation, and (iii) receiving the interface evaluation δ of the destination component in the synchronization label, all this within an atomic component transition $\xrightarrow{\overline{\delta : \mathbf{gq}(T{:}t)@\mathsf{p}}}$, note that similarly to the previous rule for maximal execution of local actions we consider the entire component, for the same reasons.

Now we discuss the transition relation $\underset{\text{SL}}{\rightarrow} \subseteq Comp \times \mathsf{Dist}(Comp)$, defined in Table 7 by rules (c-rmt/c-mxt) in a recursive way. This relation realizes maximal executions of the local actions by iterating the application of the component transition relation $\underset{\text{L}}{\rightarrow} \subseteq Comp \times Act^L_{Sys} \times \mathsf{Dist}(Comp)$ for local actions. This relation is defined only on the components syntactic category $Comp$ and on the set of local actions Act^L_{Sys}. The relation $\underset{\text{L}}{\rightarrow}$ is defined in Table 7 by rules (c-putl/c-getl/c-qryl). Let us now discuss the rules.

(c-rmt) is the base case of the definition of $\underset{\text{SL}}{\rightarrow}$, it applies to a *remote-enabled* process (R), and, since there is no *local* action to perform, it returns the same component state with rate 1;

(c-mxt) is the inductive case of the definition, it applies to a mixed process (M) and it executes local actions as much as possible by: (1) applying one local action ($\underset{\text{L}}{\rightarrow}$), thus obtaining a continuation $\mathscr{C}$ which is a probability distribution over possible components, and (2) applying the convolution $\underset{\text{SL}}{\rightarrow}^{cv} \subseteq \mathsf{Dist}(Comp) \times \mathsf{Dist}(Comp)$ of the relation $\underset{\text{SL}}{\rightarrow} \subseteq Comp \times \mathsf{Dist}(Comp)$ to the continuation $\mathscr{C}$;

(c-putl) applies to mixed processes (M) and allows to execute a local **put** action, by using the $\oplus$ operator to add the item t to the local knowledge state and obtaining a probability distribution π on possible new knowledge states which is combined with the continuation $\mathscr{M}$ of the process M, which is simply a Dirac distribution (see Table 5);

(c-getl)/(c-qryl) apply to mixed processes (M) and realize the local **get**/**qry** actions, by using the $\vdash$ operator to remove/deduce an item t matching the template T from the local knowledge state and combining the resulting probability distribution with the continuation $\mathscr{M}$ of the process M, which is again a Dirac distribution.

The rules for systems are the same as those of the ACT-OR semantics and can be found in Table 4.

6 Network-oriented Operational Semantics

For the sake of conciseness, in this section we illustrate only the modifications to the Action-oriented semantics necessary to model the Network-oriented semantics. In particular, in Tables 8, 9, and 10 we omit the rules that are unchanged w.r.t Tables 2, 3, and 4 and we just leave their label, as usual.

In order to realize this semantics we extend the set of labels of actions performed by processes and systems as described in the following.

6.1 Explanatory example

In the following we illustrate this variant of the semantics through a couple of examples and we point out issues that lead us to necessary simplifications and further design choices.

Let us consider a process P, of the form $\mathbf{put}(v)@\mathsf{p}\,.\,Q$, and the execution of action $\mathbf{put}(v)@\mathsf{p}$, as illustrated in Figure 1. The execution begins with the creation of one special message *for each* component of the system. These messages play the role of *observers*: each of them travels in the system and reaches the component it is associated with. The special message creation and message-component association phase has rate λ: this value is computed as a function of several factors, among which (the size of) t. After message creation, P can proceed—since **put** actions are non-blocking—behaving like Q. Each special message has to reach its destination component, which checks if its interface satisfies p, and if so, it delivers t in the knowledge repository of the component. Observer delivery is performed with rate μ, which may depend on t and other parameters like the distance between the component where P resides and the target component. Therefore, a *distinct* rate μ is associated to each target

In practice, one can be interested in modeling also the event of failed delivery of the observers. This is interesting both for producing more realistic models (with unreliable network communication), and for allowing the application of advanced analysis techniques based on fluid approximation [3], such as fluid model-checking [2]. Therefore, we add an error probability to the observers delivery, which we indicate as p_{err} (or simply err, in the figure). This more detailed semantics of the $\mathbf{put}(v)@\mathsf{p}$ action is illustrated in Fig. 3 and described below in more detail.

We consider three components: $C_1 = I_1\,[\,K_1,\,P_1\,]$, $C_2 = I_2\,[\,K_2,\,P_2\,]$, and $C_3 = I_3\,[\,K_3,\,P_3\,]$ and we assume process P_1 is defined as $\mathbf{put}(v)@\mathsf{p}\,.\,Q$ as described above. Note that different components may be in different locations. The interaction we illustrate starts with process $\underline{P_1}$ executing the first phase of $\mathbf{put}(v)@\mathsf{p}$, i.e. creating two[2] copies of the special message $\overline{\{v@\mathsf{p}\}}$, one for component C_2 and one for component C_3, and sending these messages. The time required for this phase (denoted in blue in the figure) is modeled by a RV with rate λ. The time for the two messages to arrive at components C_2 and C_3 is modeled by RVs with rates μ_2 and μ_3 (in the figure this is illustrated by two arrows). The delivery of the two messages fails with probability err_2 and err_3, respectively, and succeeds with their complement. The execution of component C_1 restarts as soon as the copies of the messages are sent, without waiting for their arrival at the destination components (the red stripe in the figure illustrates the resumed execution of C_1). The evaluation of predicate p is performed when the message arrives at the corresponding component so, for example, it may happen that C_2 satisfies p

[2]For the sake of notational simplicity, in the present report we assume that predicate p in process actions implicitly refers only to the *other* components, excluding the one where the process is in execution.

at the time the message arrives (so K_2 is updated accordingly), while C_3 does not satisfy p (thus leaving K_3 unchanged).

Let us now consider the execution of a process P of the form $\mathbf{get}(T)@\mathsf{p}.Q$ (or, equivalently, $\mathbf{qry}(T)@\mathsf{p}.Q$), as illustrated in Fig. 2. Similarly to the execution of $\mathbf{put}(v)@\mathsf{p}$, the first operation performed consists in the creation of a special request message, which is delivered to every component. Since the **get** (resp. **qry**) action is blocking, the process moves into a waiting state after creation of this message. Each copy of the message is delivered to the corresponding component. Upon message arrival, every component satisfying predicate p and having in its knowledge an item t matching the template T, is eligible to answer the request with item t. Message preparation, message delivery, as well as response delivery are events that take time (e.g. depending on the size of the item t or the distance of the involved component) and have rates λ, β_j, and μ_j, respectively. Concerning response delivery, there is a race condition and *only one* component succeeds in providing the item t. Once the item has been delivered, process P can restart its execution from $Q\vartheta$, with a suitable variable binding $\mathsf{match}(T, t) = \vartheta$.

In order to simplify the semantics of the **get**/**qry** actions and to make it more similar to the two-steps semantics of the **put** action, we decided to model the two phases of delivery and response as a single one. So, on message creation, the sender is blocked on waiting for some receiver to synchronize with it on the exchange of the retrieved item t matching the template T, as illustrated in Fig. 4. During this synchronization, the predicate p is also checked, on the side of the receiver, and the knowledge is changed accordingly. In terms of the underlying stochastic model, we are replacing an Erlang RV by an Exponential RV. This choice is also convenient for simplifying the definition of the formal semantics, since it avoids the need of giving a unique id to observer messages, to be used in the subsequent response phase.

In Fig. 4, we illustrate an example of execution of a **get**/**qry** action. Here we consider four components: $C_1 = I_1\,[\,K_1,\,P_1\,]$, $C_2 = I_2\,[\,K_2,\,P_2\,]$, $C_3 = I_3\,[\,K_3,\,P_3\,]$, and $C_4 = I_4\,[\,K_4,\,P_4\,]$ and we assume process P_1 is defined as $\mathbf{get}(T)@\mathsf{p}\,.\,Q$. The interaction we illustrate starts with process P_1 creating the observer messages with rate λ (denoted in blue in the figure). The waiting state of C_1, after the creation of the messages, is denoted by a dashed line. The messages are delivered to C_2, C_3, and C_4. Upon arrival, components check for satisfaction of predicate p and availability of an item matching T. Let us assume that C_2 and C_3 satisfy p and have matching knowledge items t_2 and t_3, respectively, whereas C_4 either does not satisfy p or has no matching knowledge item. Therefore, C_2 and C_3 are the only components that can synchronize with C_1: they perform this synchronization with rates μ_2 and μ_3, respectively (depending, for example, on the size of t_2 and t_3, and on their locations). In the figure, this is represented by arrows from C_2 and C_3 to C_1. In the Markovian semantics, this is realized by a race condition in which just one of the two synchronizations is actually executed. In the figure, it is assumed that component C_2 wins the race, and synchronizes on an item t_2. Then, the execution of C_1 restarts (the red stripe in the figure illustrates the resumed execution of C_1). The synchronization attempt of C_3 is simply lost. Here we need to be careful in how to handle these message deliveries. If we are considering a **get** action, the retrieving of a knowledge item *from* a component requires the item to be *removed* from the knowledge. In the case illustrated in the figure, then, the knowledge K_2 needs to be updated removing t_2, while the knowledge K_3 must remain unchanged.

6.2 Operational semantics of processes

The NET-OR semantics of StocS processes is the RTS $(\mathit{Proc}, \mathit{Act}_{\mathit{Proc}}, \mathbb{R}_{\geq 0}, \rightarrowtail_e)$, where Proc is the set of process terms defined according to the syntax of StocS given in Table 1 and the set $\mathit{Act}_{\mathit{Proc}}$ of labels is defined according to the grammar below (where $t \in \mathbb{I}$, $T \in \mathbb{T}$, $\mathbf{gq} \in \{\mathbf{get}, \mathbf{qry}\}$, c is a Target, and e is the evaluation of an interface) and it is ranged over

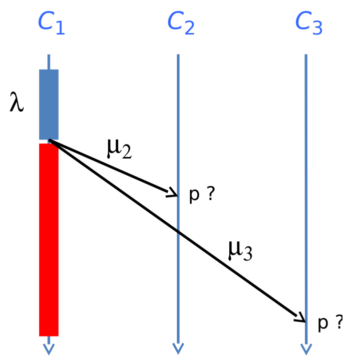

Figure 1: Dynamics of the **put** action.

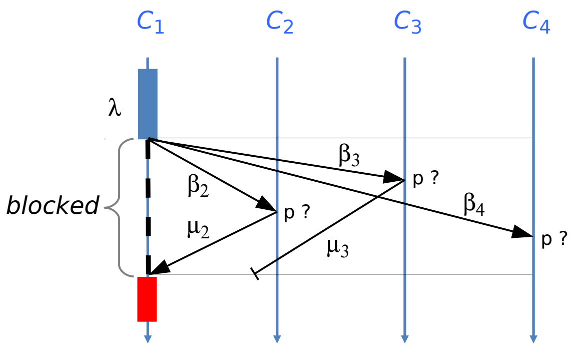

Figure 2: Dynamics of the **get**/**qry** action.

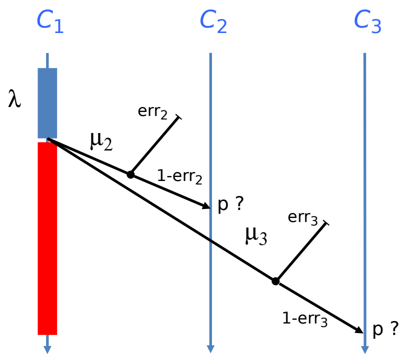

Figure 3: Actual model of **put**.

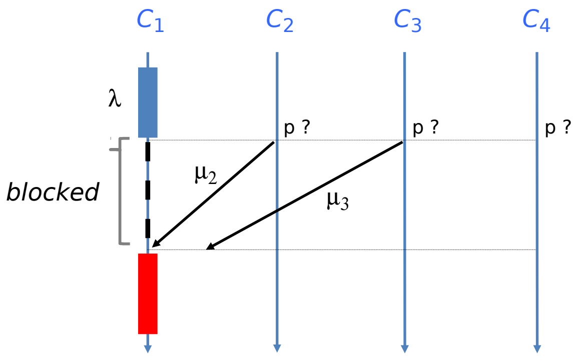

Figure 4: Actual model of **get**/**qry**.

by $\alpha, \alpha', \ldots$:

$$Act_{Proc} \quad ::= \quad \tau \quad | \quad \overline{\{t@\mathsf{p}\}} \quad | \quad \overline{e : \mathbf{put}(t)@c} \quad | \quad \overline{e : \mathbf{gq}(T : t)@c}$$

and $\rightharpoondown \subseteq Proc \times Act_{Proc} \times \mathbf{FTF}(Proc, \mathbb{R}_{\geq 0})$ is the least relation satisfying the rules of Table 8 (like in the ACT-OR semantics, $\rightharpoondown_e$ is parametrized by e, which is the interface evaluation of the component in which the process resides: we feel free to omit the parameter, if not used in the rule).

We now briefly illustrate the rules of Table 8. We assume to have additional syntactical terms (not available at the user syntax level) which we call *envelopes*. They are of the form $\{t@\mathsf{p}\}_\mu$, can be put in parallel with processes, and denote messages that are currently traveling towards targets. A second syntactical construct we introduce is $\{\mathbf{get}(T)@\mathsf{p}\}$ ($\{\mathbf{qry}(T)@\mathsf{p}\}$, respectively) which denotes a *waiting* state of the process and it is treated as an action.

(ENV) allows to complete envelope delivery within a duration parametrized by μ;

(PUT) allows a process to issue a **put** action at a target c with rate 1;

(GQL) allows a process to issue a **get** (**qry**, respectively) action over the local knowledge repository (i.e. with target self). The rule models the execution of action $\mathbf{get}(T)@\mathsf{self}$ ($\mathbf{qry}(T)@\mathsf{self}$, respectively) by process $\mathbf{get}(T)@\mathsf{self}.P$ ($\mathbf{qry}(T)@\mathsf{self}.P$, respectively). The duration of this action is described by a rate λ computed using the function $\mathcal{R}$

Inactive process and envelopes:

(NIL, Table 2) $$\frac{}{\{t@\mathsf{p}\}_\mu \xrightarrow{\overline{\{t@\mathsf{p}\}}} [\mathbf{nil} \mapsto \mu]}\ (\text{ENV}) \qquad \frac{\alpha \neq \overline{\{t@\mathsf{p}\}}}{\{t@\mathsf{p}\}_\mu \xrightarrow{\alpha} []}\ (\text{ENV}_\text{B})$$

Actions (where, $\mathbf{gq} \in \{\mathbf{get}, \mathbf{qry}\}$, c is a Target, and p is a Predicate):

(PUT, Table 2) (PUT$_\text{B}$, Table 2)

$$\frac{\mathsf{match}(T,t) = \vartheta \quad \lambda = \mathcal{R}(\sigma, \mathbf{gq}(T:t)@\mathsf{self}, \_)}{\mathbf{gq}(T)@\mathsf{self}.P \xrightarrow{\overline{\sigma:\mathbf{gq}(T:t)@\mathsf{self}}}_\sigma [P\vartheta \mapsto \lambda]}\ (\text{GQL})$$

$$\frac{\neg\mathsf{match}(T,t)}{\mathbf{gq}(T)@\mathsf{self}.P \xrightarrow{\overline{\_:\mathbf{gq}(T:t)@\mathsf{self}}} []}\ (\text{GQL}_\text{B1}) \qquad \frac{\alpha \neq \overline{\_:\mathbf{gq}(T:t)@\mathsf{self}}}{\mathbf{gq}(T)@\mathsf{self}.P \xrightarrow{\alpha} []}\ (\text{GQL}_\text{B2})$$

$$\frac{\lambda = \mathcal{R}(\sigma, \mathbf{gq}(T:\_)@\mathsf{p}, \_)}{\mathbf{gq}(T)@\mathsf{p}.P \xrightarrow{\tau}_\sigma [\{\mathbf{gq}(T)@\mathsf{p}\}.P \mapsto \lambda]}\ (\text{GQW}) \qquad \frac{\alpha \neq \tau}{\mathbf{gq}(T)@\mathsf{p}.P \xrightarrow{\alpha} []}\ (\text{GQW}_\text{B})$$

$$\frac{\mathsf{match}(T,t) = \vartheta \quad \beta = \mathcal{R}(\sigma, \{\mathbf{gq}(T:t)@\mathsf{p}\}, \delta)}{\{\mathbf{gq}(T)@\mathsf{p}\}.P \xrightarrow{\delta:\overline{\{\mathbf{gq}(T:t)@\mathsf{p}\}}}_\sigma [P\vartheta \mapsto \beta]}\ (\text{GQD})$$

$$\frac{\neg\mathsf{match}(T,t)}{\{\mathbf{gq}(T)@\mathsf{p}\}.P \xrightarrow{\overline{\_:\{\mathbf{gq}(T:t)@\mathsf{p}\}}} []}\ (\text{GQD}_\text{B1}) \qquad \frac{\alpha \neq \overline{\_:\{\mathbf{gq}(T:t)@\mathsf{p}\}}}{\{\mathbf{gq}(T)@\mathsf{p}\}.P \xrightarrow{\alpha} []}\ (\text{GQD}_\text{B2})$$

Choice, definition, and parallel composition:

(CHO, Table 2) (DEF, Table 2) (PAR, Table 2)

Table 8: Operational semantics of StocS processes (NET-OR).

depending on the interface evaluation of the source σ (i.e. the container component) and on the action;

(GQW) realizes the first step of a **get** (**qry**, respectively) action over a remote knowledge in a component satisfying a predicate p, which consists in preparing an envelope $\{\mathbf{get}(T)@\mathsf{p}\}$ ($\{\mathbf{qry}(T)@\mathsf{p}\}$, respectively), which takes a time interval exponentially distributed with rate λ, and brings process P to a *wait* state $\{\mathbf{get}(T)@\mathsf{p}\}.P$ ($\{\mathbf{qry}(T)@\mathsf{p}\}.P$, respectively). Recall that **get**/**qry** actions are *blocking* and the execution of P is resumed only when a counterpart satisfying p has a knowledge item t matching T available and the delivery of t is completed. The duration of this first step is described by a rate λ computed using the function $\mathcal{R}$ depending only on the interface evaluation of the source σ (i.e. the container component) and the sent template T;

(GQD) realizes the second step of a **get** (**qry**, respectively) action, which consists in the *delivery* of the knowledge item t matching T and has a duration described by a rate β computed by the function $\mathcal{R}$. Note that in this case the function $\mathcal{R}$ is computed considering interface evaluation of the source σ and the destination δ, as well as the sent item t, which means that this rate can be made dependent (for example) on the distance of the two parties.

put actions:

(C-PUTL, Table 3) (C-PUTO, Table 3)

$$\frac{\delta = I(K) \quad \mu = \mathcal{R}(\sigma, \{t@\mathsf{p}\}, \delta) \quad p_{\mathsf{err}} = f_{\mathsf{err}}(\sigma, \{t@\mathsf{p}\}, \delta)}{I\,[\,K,\,P\,] \xrightarrow{\sigma\,:\,\mathbf{put}(t)@\mathsf{p}} [\,I\,[\,K,\,P\,] \mapsto p_{\mathsf{err}},\; I[K, P|\{t@\mathsf{p}\}_{\mu}] \mapsto (1 - p_{\mathsf{err}})\,]} \quad (\text{C-PUTI})$$

$$\frac{P \xrightarrow{\overline{\{t@\mathsf{p}\}}} \mathscr{P} \quad I(K) \models \mathsf{p} \quad K \oplus t = \pi}{I\,[\,K,\,P\,] \xrightarrow{\overline{\{t@\mathsf{p}\}}} I[\pi, \mathscr{P}]} \quad (\text{C-ENVA}) \qquad \frac{P \xrightarrow{\overline{\{t@\mathsf{p}\}}} \mathscr{P} \quad I(K) \not\models \mathsf{p}}{I\,[\,K,\,P\,] \xrightarrow{\overline{\{t@\mathsf{p}\}}} I[(\mathcal{X}K), \mathscr{P}]} \quad (\text{C-ENVR})$$

get/**qry** actions (where, $\mathbf{gq} \in \{\mathbf{get}, \mathbf{qry}\}$):

(C-GETL, Table 3) (C-QRYL, Table 3) (C-GETL$_{\mathrm{B}}$, Table 3) (C-GETL$_{\mathrm{B}}$, Table 3)

$$\frac{\sigma = I(K) \quad P \xrightarrow{\overline{\delta\,:\,\{\mathbf{gq}(T:t)@\mathsf{p}\}}}_{\sigma} \mathscr{P}}{I\,[\,K,\,P\,] \xrightarrow{\overline{\delta\,:\,\{\mathbf{gq}(T:t)@\mathsf{p}\}}} I[(\mathcal{X}K), \mathscr{P}]} \quad (\text{C-GQO})$$

$$\frac{\delta = I(K) \quad \delta \models \mathsf{p} \quad K \ominus T = \pi}{I\,[\,K,\,P\,] \xrightarrow{\delta\,:\,\{\mathbf{get}(T:t)@\mathsf{p}\}} I[\pi(t), (\mathcal{X}P)]} \quad (\text{C-GETI})$$

$$\frac{\delta \neq I(K) \quad \vee \quad I(K) \not\models \mathsf{p} \quad \vee \quad K \ominus T = \bot}{I\,[\,K,\,P\,] \xrightarrow{-\,:\,\{\mathbf{get}(T:t)@\mathsf{p}\}} []} \quad (\text{C-GETI}_{\mathrm{B}})$$

$$\frac{\delta = I(K) \quad \delta \models \mathsf{p} \quad K \vdash T = \pi}{I\,[\,K,\,P\,] \xrightarrow{\delta\,:\,\{\mathbf{qry}(T:t)@\mathsf{p}\}} [I\,[\,K,\,P\,] \mapsto \pi(t)]} \quad (\text{C-QRYI})$$

$$\frac{\delta \neq I(K) \quad \vee \quad I(K) \not\models \mathsf{p} \quad \vee \quad K \vdash T = \bot}{I\,[\,K,\,P\,] \xrightarrow{-\,:\,\{\mathbf{qry}(T:t)@\mathsf{p}\}} []} \quad (\text{C-QRYI}_{\mathrm{B}})$$

τ actions:

$$\frac{\rho = I(K) \quad P \xrightarrow{\tau}_{\rho} \mathscr{P}}{I\,[\,K,\,P\,] \xrightarrow{\tau} I[(\mathcal{X}K), \mathscr{P}]} \quad (\text{C-TAU})$$

Table 9: Operational semantics of StocS components (NET-OR).

6.3 Operational semantics of components and systems

The NET-OR semantics of StocS systems is the RTS $(Sys, Act_{Sys}, \mathbb{R}_{\geq 0}, \rightarrow)$, where Sys is the set of system terms defined according to the syntax of StocS given in Table 1 and the set Act_{Sys} of labels is defined according to the grammar below (where $\mathbf{gq} \in \{\mathbf{get}, \mathbf{qry}\}$, $t \in \mathbb{I}$, $T \in \mathbb{T}$, p is a Predicate, and e is the evaluation of an interface):

$$\begin{array}{lllll}
Act_{Sys} & ::= & e : \mathbf{put}(t)@\mathsf{p} \;\mid\; e : \{\mathbf{gq}(T : t)@\mathsf{p}\} \;\mid & & \text{(input actions)} \\
 & & \overline{e : \mathbf{put}(t)@\mathsf{p}} \;\mid\; \overline{e : \{\mathbf{gq}(T : t)@\mathsf{p}\}} \;\mid & & \text{(output actions)} \\
 & & \tau \;\mid\; \overleftrightarrow{e : \{\mathbf{gq}(T : t)@\mathsf{p}\}} \;\mid & & \text{(synchronizations)} \\
 & & \overline{\{t@\mathsf{p}\}} & & \text{(envelopes)}
\end{array}$$

and $\rightarrow \subseteq Sys \times Act_{Sys} \times \mathbf{FTF}(Sys, \mathbb{R}_{\geq 0})$ is the least relation satisfying the rules of Tables 9 and 10, where the process relation defined in Table 8 is also used.

put synchronization:

(s-po, Table 4) (s-pi, Table 4)

get/**qry** synchronization ($\mathbf{gq} \in \{\mathbf{get}, \mathbf{qry}\}$):

$$\frac{\begin{array}{ccc} S_1 \xrightarrow{\overleftrightarrow{\delta{:}\{\mathbf{gq}(T{:}t)@\mathsf{p}\}}} \mathscr{S}_1^s & S_1 \xrightarrow{\overline{\delta{:}\{\mathbf{gq}(T{:}t)@\mathsf{p}\}}} \mathscr{S}_1^o & S_1 \xrightarrow{\delta{:}\{\mathbf{gq}(T{:}t)@\mathsf{p}\}} \mathscr{S}_1^i \\ S_2 \xrightarrow{\overleftrightarrow{\delta{:}\{\mathbf{gq}(T{:}t)@\mathsf{p}\}}} \mathscr{S}_2^s & S_2 \xrightarrow{\overline{\delta{:}\{\mathbf{gq}(T{:}t)@\mathsf{p}\}}} \mathscr{S}_2^o & S_2 \xrightarrow{\delta{:}\{\mathbf{gq}(T{:}t)@\mathsf{p}\}} \mathscr{S}_2^i \end{array}}{S_1 \parallel S_2 \xrightarrow{\overleftrightarrow{\delta{:}\{\mathbf{gq}(T{:}t)@\mathsf{p}\}}} \mathscr{S}_1^s \parallel (\mathcal{X}\, S_2) + \mathscr{S}_1^o \parallel \mathscr{S}_2^i + \mathscr{S}_1^i \parallel \mathscr{S}_2^o + (\mathcal{X}\, S_1) \parallel \mathscr{S}_2^s} \ (\textsc{s-gqs})$$

$$\frac{S_1 \xrightarrow{\delta{:}\{\mathbf{gq}(T{:}t)@\mathsf{p}\}} \mathscr{S}_1 \quad S_2 \xrightarrow{\delta{:}\{\mathbf{gq}(T{:}t)@\mathsf{p}\}} \mathscr{S}_2}{S_1 \parallel S_2 \xrightarrow{\delta{:}\{\mathbf{gq}(T{:}t)@\mathsf{p}\}} \mathscr{S}_1 \parallel (\mathcal{X}\, S_2) + (\mathcal{X}\, S_1) \parallel \mathscr{S}_2} \ (\textsc{s-gqi})$$

Internal actions, for $\alpha \in \{\, \tau, \overleftrightarrow{e : \mathbf{put}(t)@\mathsf{self}}, \overleftrightarrow{e : \mathbf{gq}(T : t)@\mathsf{self}}, \overline{\{t@\mathsf{p}\}}\,\}$:

$$\frac{S_1 \xrightarrow{\alpha} \mathscr{S}_1 \quad S_2 \xrightarrow{\alpha} \mathscr{S}_2}{S_1 \parallel S_2 \xrightarrow{\alpha} \mathscr{S}_1 \parallel (\mathcal{X}\, S_2) + (\mathcal{X}\, S_1) \parallel \mathscr{S}_2} \ (\textsc{s-spl})$$

Table 10: Operational semantics of StocS systems (net-or).

The definition of the semantics of system parallel composition $S_1 \parallel S_2$ uses Def. 3.1, item (3) applied to the system parallel composition constructor $\parallel$, which is injective. As usual, interleaving is modelled as a combination of lifted $\parallel$, $+$ on functions and the characteristic function. In the rules, we also use Def. 3.1, item (3) applied to the component syntactic constructors $I[\,\cdot,\cdot\,]$, which is injective.

In Table 9, rules are grouped to illustrate how the various action types are realized.

(c-puti) models the *initiation* of the execution of action $\mathbf{put}(t)@c$, which requires several steps to complete, it allows the reception of a **put** action, and it is responsible for the creation of the envelope (carrying the incoming message) in parallel to the local process of a component, thus modeling its travel towards that component in terms of the time necessary to reach it, parametrized by rate μ (the fact that the envelope is in parallel with the process of the potential receiver component by no means should be interpreted as the representation of the fact that the message reached the component; simply, the association between the message and the component is represented by means of a parallel composition term; in other words, the fact that a specific message is 'addressed' to a component is represented syntactically by such a parallel composition); this action is executed with rate λ, computed using the function $\mathcal{R}$ depending on the interface evaluation of the source σ (i.e. the container component) and the sent item t; this is postulated by the rule (put) and realized at system level by the broadcast rules of Table 4.

(c-enva)/(c-envr) realize envelope delivery by specifying the conditions under which a component *accepts* or *refuses*, respectively, an arriving envelope;

(c-gqo) realizes an output **get**/**qry** action as in the act-or semantics, but with a different label ($\{\ldots\}$) which denotes the synchronization on a *waiting state*;

(c-geti)/(c-qryi) realize an input **get**/**qry** action, again as in the act-or semantics, but with a different label ($\{\ldots\}$);

(c-tau) allows a component to make a τ whenever its process makes such an action.

7 Activity-oriented Operational Semantics

In this section we illustrate the Activity-oriented semantics (abbreviated to ACTIV-OR), which is essentially based on the ACT-OR syntax and semantics. The main novelty w.r.t. that semantics is the *replacement* of StocS actions by a new syntactic constructor $\langle\, seq \,\rangle_\lambda$ that allows a user to specify a sequence *seq* of StocS actions to be executed *atomically* and with overall exponential rate λ. The execution rate λ is not influenced by the actions performed, but specified by the user (possibly through a function of the state of the system). In the case of the empty sequence of actions ($\langle\,\rangle_\lambda$) this constructor can be also used to specify delays with rate λ having no effect on local/remote knowledge states.

The motivation for this semantics, as already discussed in Section 2.2, is to have the possibility of specifying, at a high-level of abstraction, interaction mechanisms which we call *activities*, that are modeled atomically. Such an abstraction provides a significant reduction of the state-space of the system, since the execution of actions within sequences is not interleaved with that of other processes in parallel.

Recall that in ACT-OR local and output actions are stochastic, while input actions are probabilistic. They are composed together into a timed synchronization. In ACTIV-OR, in order to execute action sequences atomically and with a given rate λ, we give a *time-abstract* semantics for local and output actions of *components* (defining the transition relation $\xrightarrow[\text{TA}]{}$), a *stochastic* semantics for sequences of actions of *components* (defining the transition relation $\xrightarrow[\text{T}]{}$), and we maintain the *probabilistic* semantics for input actions of *components* (implemented by the transition relation $\xrightarrow[\text{PR}]{}$). To obtain a time-abstract semantics it is enough to redefine output and local actions in a time-abstract way (i.e. probabilistically). As a simple consequence, the synchronization of output actions and input actions will be itself time-abstract. Then, when the execution of the sequence of actions is terminated, we perform a stochastic delay λ and compose it with the probability distribution obtained by time-abstract execution. In particular, an action sequence $\langle\, \alpha_1.\cdots.\alpha_n \,\rangle_\lambda$ is rewritten, by a sequence initiation action $\xrightarrow[\text{TA}]{\mathsf{init\_s}}$ into a new sequence $\alpha_1.\cdots.\alpha_n.\mathsf{end\_s}.\mathsf{delay}(\lambda)$. Then, using the transition relation $\xrightarrow[\text{TA}]{}$, the sequence $\alpha_1.\cdots.\alpha_n$ is executed in a time-abstract fashion (taking only the probabilistic aspect of α_i and combining them by using the convolution operator $\cdot^{cv}$), and terminated by a sequence termination action $\xrightarrow[\text{TA}]{\mathsf{end\_s}}$. The remaining $\mathsf{delay}(\lambda)$ is executed using the stochastic transition $\xrightarrow{\mathsf{delay}}$. At the *system* level, then, the stochastic and probabilistic semantics ($\xrightarrow[\text{T}]{}$ and $\xrightarrow[\text{PR}]{}$, respectively) are composed together by the transition relation $\rightarrow$.

7.1 Operational semantics of processes

The *stochastic* ACTIV-OR semantics of StocS *processes* is the RTS $(\mathit{ExtProc}, \mathit{Act}_{\mathit{Proc}}, \mathbb{R}_{\geq 0}, \rightarrow)$ where *ExtProc* is the set of process terms defined by the following syntax, starting from the symbol EP:

$$\begin{array}{rcl} EP & ::= & \mathit{Seq}.\mathsf{end\_s}.\mathsf{delay}(\lambda).P \;\big|\; P \\ P & ::= & \mathbf{nil} \;\big|\; \langle\, \mathit{Seq} \,\rangle_\lambda.P \;\big|\; P + P \;\big|\; P \,|\, P \;\big|\; X \;\big|\; A(\bar{p}) \\ \mathit{Seq} & ::= & \varepsilon \;\big|\; a.\mathit{Seq} \end{array}$$

The set *Proc* modifies the syntax of StocS processes given in Table 1 by *replacing* actions with sequences $\langle \dots \rangle_\lambda$ of actions (where ε denotes the empty sequence of actions and $\langle\, \varepsilon \,\rangle \equiv \langle\,\rangle$). The set *ExtProc* is introduced only for internal purposes of the time-abstract semantics and it extends the set *Proc*, defined by the above grammar starting from the symbol P, by allowing the prefix $\mathit{Seq}.\mathsf{end\_s}.\mathsf{delay}(\lambda)$ of Processes, made of an (un-parenthesized) time-abstract action

Time-Abstract actions (where, $\mathbf{gq}\in\{\mathbf{get},\mathbf{qry}\}$ and c is a Target):

$$\frac{}{\langle\,\rangle_\lambda.\,P \xrightharpoonup{\langle\,\rangle} [\,P\mapsto\lambda\,]}\ (\textsc{eseq}) \qquad \frac{}{\mathsf{delay}(\lambda).\,P \xrightharpoonup{\mathsf{delay}} [\,P\mapsto\lambda\,]}\ (\textsc{delay})$$

$$\frac{\alpha\in \mathit{OneAct}_{\mathit{Proc}}\cup\{\mathsf{init\_s},\mathsf{end\_s}\}}{P \xrightharpoonup{\alpha} []}\ (\textsc{time-abstract}_{\mathrm{B}})$$

Choice, definition, and parallel composition:

$$\frac{P \xrightharpoonup{\alpha} \mathscr{P} \quad Q \xrightharpoonup{\alpha} \mathscr{Q}}{P+Q \xrightharpoonup{\alpha} \mathscr{P}+\mathscr{Q}}\ (\textsc{cho}) \qquad \frac{A(\overrightarrow{x}) \overset{def}{=} P \quad P[\overrightarrow{v}/\overrightarrow{x}] \xrightharpoonup{\alpha} \mathscr{P}}{A(\overrightarrow{v}) \xrightharpoonup{\alpha} \mathscr{P}}\ (\textsc{def})$$

$$\frac{P \xrightharpoonup{\alpha} \mathscr{P} \quad Q \xrightharpoonup{\alpha} \mathscr{Q}}{P \mid Q \xrightharpoonup{\alpha} \mathscr{P} \mid (\mathcal{X}\,Q) + (\mathcal{X}\,P) \mid \mathscr{Q}}\ (\textsc{par})$$

Table 11: Operational sem. of StocS processes (activ-or), $\rightharpoonup$ transition relation.

Time-Abstract actions (where, $\mathbf{gq}\in\{\mathbf{get},\mathbf{qry}\}$ and c is a Target):

$$\frac{}{\mathbf{put}(t)@c\,.\,P \xrightharpoonup[\text{TA}]{\mathbf{put}(t)@c} [\,P\mapsto 1\,]}\ (\textsc{put-ta}) \qquad \frac{\alpha\neq \overline{\mathbf{put}(t)@c}}{\mathbf{put}(t)@c\,.\,P \xrightharpoonup[\text{TA}]{\alpha} []}\ (\textsc{put-ta}_{\mathrm{B}})$$

$$\frac{}{\mathbf{gq}(T)@c\,.\,P \xrightharpoonup[\text{TA}]{\mathbf{gq}(T)@c} [\,P\mapsto 1\,]}\ (\textsc{gq-ta}) \qquad \frac{\alpha\neq \mathbf{gq}(T)@c}{\mathbf{gq}(T)@c\,.\,P \xrightharpoonup[\text{TA}]{\alpha} []}\ (\textsc{gq-ta}_{\mathrm{B}})$$

$$\frac{\mathit{seq}=a_1.\cdots.a_n \quad \text{for } n\geq 1}{\langle\,\mathit{seq}\,\rangle_\lambda.\,P \xrightharpoonup[\text{TA}]{\mathsf{init\_s}} [\,\mathit{seq}.\mathsf{end\_s}.\mathsf{delay}(\lambda).P\mapsto 1\,]}\ (\textsc{init-ta}) \qquad \frac{}{\mathsf{end\_s}.\,P \xrightharpoonup[\text{TA}]{\mathsf{end\_s}} [\,P\mapsto 1\,]}\ (\textsc{end-ta})$$

$$\frac{\alpha\in\{\langle\,\rangle,\mathsf{delay}\}}{\alpha.\,P \xrightharpoonup[\text{TA}]{\alpha} []}\ (\textsc{timed-ta}_{\mathrm{B}})$$

Choice, definition, and parallel composition:

$$\frac{P \xrightharpoonup[\text{TA}]{\alpha} \mathscr{P} \quad Q \xrightharpoonup[\text{TA}]{\alpha} \mathscr{Q}}{P+Q \xrightharpoonup[\text{TA}]{\alpha} \mathscr{P}+\mathscr{Q}}\ (\textsc{cho-ta}) \qquad \frac{A(\overrightarrow{x}) \overset{def}{=} P \quad P[\overrightarrow{v}/\overrightarrow{x}] \xrightharpoonup[\text{TA}]{\alpha} \mathscr{P}}{A(\overrightarrow{v}) \xrightharpoonup[\text{TA}]{\alpha} \mathscr{P}}\ (\textsc{def-ta})$$

$$\frac{P \xrightharpoonup[\text{TA}]{\alpha} \mathscr{P} \quad Q \xrightharpoonup[\text{TA}]{\alpha} \mathscr{Q}}{P \mid Q \xrightharpoonup[\text{TA}]{\alpha} \mathscr{P} \mid (\mathcal{X}\,Q) + (\mathcal{X}\,P) \mid \mathscr{Q}}\ (\textsc{par-ta})$$

Table 12: Operational sem. of StocS processes (activ-or), $\xrightharpoonup[\text{TA}]{}$ transition relation.

sequence, a time-abstract sequence terminator $\mathsf{end\_s}$, and a stochastic action $\mathsf{delay}(\lambda)$. The set $\mathit{Act}_{\mathit{Proc}}$ of labels is defined according to the grammar below (where $t\in\mathbb{I}$, $T\in\mathbb{T}$, and c is a Target) and it is ranged over by $\alpha,\alpha',\ldots$:

$$\begin{array}{ll} \mathit{Act}_{\mathit{Proc}} & ::= \ \mathsf{init\_s} \ \big|\ \mathsf{end\_s} \ \big|\ \mathsf{delay}(\lambda) \ \big|\ \langle\,\rangle \ \big|\ \mathit{OneAct}_{\mathit{Proc}} \\ \mathit{OneAct}_{\mathit{Proc}} & ::= \ \mathbf{put}(t)@c \ \big|\ \mathbf{get}(T:t)@c \ \big|\ \mathbf{qry}(T:t)@c \end{array}$$

The transition relation $\rightharpoonup\ \subseteq \mathit{ExtProc}\times \mathit{Act}_{\mathit{Proc}}\times \mathbf{FTF}(\mathit{Proc},\mathbb{R}_{\geq 0})$ is the least relation satis-

Sequence synchronization
(where $C_1,\ldots,C_n$ are components and *seq* a non-empty sequence of actions):

$$\frac{C_i \xrightarrow[\mathrm{T}]{\overline{\langle\,\rangle}} \mathscr{C}_i \quad C_1 \| \ldots \| C_{i-1} \| C_{i+1} \| \ldots \| C_n \xrightarrow[\mathrm{PR}]{\langle\,\rangle} \mathscr{S}_i \quad \text{for } i = 1,\ldots,n}{C_1 \parallel \ldots \parallel C_n \xrightarrow{\overleftrightarrow{\langle\,\rangle}} \sum_{i=1}^{n} \mathscr{C}_i \parallel \mathscr{S}_i} \;(\textsc{s-delay})$$

$$\frac{C_i \xrightarrow[\mathrm{T}]{\overline{\langle\, seq\,\rangle}} \mathscr{C}_i \quad C_1 \| \ldots \| C_{i-1} \| C_{i+1} \| \ldots \| C_n \xrightarrow[\mathrm{PR}]{\langle\, seq\,\rangle} \mathscr{S}_i \quad \text{for } i = 1,\ldots,n}{C_1 \parallel \ldots \parallel C_n \xrightarrow{\overleftrightarrow{\langle\, seq\,\rangle}} \sum_{i=1}^{n} \mathscr{C}_i \parallel \mathscr{S}_i} \;(\textsc{s-ssync})$$

Table 13: Operational semantics of StocS systems (activ-or), $\rightarrow$ relation.

fying the rules of Table 11. Note that $\rightharpoondown$ always produces processes in the set *Proc*.

The *time-abstract* activ-or semantics of StocS *processes* is the RTS ($\mathit{ExtProc}, \mathit{Act}_{\mathit{Proc}}$, $\mathbb{R}_{\geq 0}, \xrightarrow[\mathrm{TA}]{}$) where the transition relation $\xrightarrow[\mathrm{TA}]{} \subseteq \mathit{ExtProc} \times \mathit{Act}_{\mathit{Proc}} \times \mathsf{Dist}(\mathit{ExtProc})$, is the least relation satisfying the rules of Table 12

7.2 Operational semantics of components and systems

The activ-or semantics of StocS *systems* is the RTS ($\mathit{Sys}, \mathit{Act}_{\mathit{Sys}}, \mathbb{R}_{\geq 0}, \rightarrow$) where *Sys* is the set of system terms defined according to the syntax of StocS given in Table 1, with the exception of processes, that are taken from the set *Proc* defined in the previous sub-section.

The set $\mathit{Act}_{\mathit{Sys}}$ of labels is defined according to the grammar below (where $\mathbf{gq} \in \{\mathbf{get}, \mathbf{qry}\}$, e is the evaluation of an interface, $t \in \mathbb{I}$, $T \in \mathbb{T}$, and p is a Predicate):

$$\mathit{Act}_{\mathit{Sys}} \quad ::= \quad \overleftrightarrow{\langle\, \mathit{Seq}_{\mathit{Sys}}\,\rangle}$$

$$\mathit{Seq}_{\mathit{Sys}} \quad ::= \quad \varepsilon \quad \Big| \quad \mathit{OneAct}_{\mathit{Proc}} \,.\, \mathit{Seq}_{\mathit{Sys}}$$

and $\rightarrow \subseteq \mathit{Sys} \times \mathit{Act}_{\mathit{Sys}} \times \mathbf{FTF}(\mathit{Sys}, \mathbb{R}_{\geq 0})$ is the least relation satisfying the rules of Table 13. The $\rightarrow$ transition relation combines the stochastic execution of $\overline{\langle\, \mathit{Seq}_{\mathit{Sys}}\,\rangle}$ output/local action sequences, performed by using the $\xrightarrow[\mathrm{T}]{}$, and the probabilistic execution of $\langle\, \mathit{Seq}_{\mathit{Sys}}\,\rangle$ input action sequences.

In particular, the $\xrightarrow[\mathrm{T}]{} \subseteq \mathit{Comp} \times \mathit{Act}^{\mathrm{T}}_{\mathit{Sys}} \times \mathbf{FTF}(\mathit{Comp}, \mathbb{R}_{\geq 0})$ transition relation is defined by the rules of Table 14, where the set $\mathit{Act}^{\mathrm{T}}_{\mathit{Sys}}$ is defined as follows:

$$\mathit{Act}^{\mathrm{T}}_{\mathit{Sys}} \quad ::= \quad \overline{\langle\, \mathit{Seq}_{\mathit{Sys}}\,\rangle}$$

The $\xrightarrow[\mathrm{T}]{}$ transition relation concatenates the probabilistic sequence initiation action $\xrightarrow[\mathrm{TA}]{\mathsf{init\_s}}$, the probabilistic maximal execution of the action sequence performed by transition relation $\xrightarrow[\mathrm{TA}]{}$ (terminated by the probabilistic sequence termination action $\xrightarrow[\mathrm{TA}]{\mathsf{end\_s}}$), and the final delay action $\xrightarrow{\mathsf{delay}}{}^{cv}$ (performed using the lifting $\rightharpoondown^{cv} \subseteq \mathsf{Dist}(\mathit{ExtProc}) \times \mathit{Act}_{\mathit{Proc}} \times \mathbf{FTF}(\mathit{Proc}, \mathbb{R}_{\geq 0})$ of $\rightharpoondown$ to probability distributions over extended processes). The transition relation $\xrightarrow[\mathrm{TA}]{} \subseteq$ $\mathit{ExtComp} \times \mathit{Act}^{\mathrm{TA}}_{\mathit{Sys}} \times \mathsf{Dist}(\mathit{ExtComp})$ as well as its variant $\xrightarrow[\mathrm{TA}]{}{}^{cv} \subseteq \mathsf{Dist}(\mathit{ExtComp}) \times \mathit{Act}^{\mathrm{TA}}_{\mathit{Sys}} \times$ $\mathsf{Dist}(\mathit{ExtComp})$, lifted to probability distributions over components, are defined in Table 14. The set $\mathit{Act}^{\mathrm{TA}}_{\mathit{Sys}}$ of actions is defined as follows:

$$\mathit{Act}^{\mathrm{TA}}_{\mathit{Sys}} \quad ::= \quad \mathit{OneAct}_{\mathit{Proc}} \quad \Big| \quad \overline{\langle\, \mathit{Seq}_{\mathit{Sys}}\,\rangle}$$

The counterpart of $\xrightarrow[\mathrm{T}]{}$ is the probabilistic transition relation $\xrightarrow[\mathrm{PR}]{} \subseteq \mathit{Comp} \times \mathit{Act}^{\mathrm{PR}}_{\mathit{Sys}} \times$

put actions:

$$\frac{P \xrightharpoonup[\mathrm{TA}]{\mathbf{put}(t)@\mathsf{self}} \mathscr{P} \quad K \oplus t = \pi}{I[K,P] \xrightarrow[\mathrm{TA}]{\mathbf{put}(t)@\mathsf{self}} I[\pi, \mathscr{P}]} \ (\textsc{c-putl-ta}) \qquad \frac{P \xrightharpoonup[\mathrm{TA}]{\mathbf{put}(t)@\mathsf{p}} \mathscr{P}}{I[K,P] \xrightarrow[\mathrm{TA}]{\mathbf{put}(t)@\mathsf{p}} I[(\mathcal{X}K), \mathscr{P}]} \ (\textsc{c-puto-ta})$$

get/**qry** actions (where, $\mathbf{gq} \in \{\mathbf{get}, \mathbf{qry}\}$):

$$\frac{P \xrightharpoonup[\mathrm{TA}]{\mathbf{get}(T)@\mathsf{self}} \mathscr{P} \quad K \ominus T = \pi}{I[K,P] \xrightarrow[\mathrm{TA}]{\mathbf{get}(T)@\mathsf{self}} \sum_{t \,\in\, \{t \in \mathbb{I} \,|\, \mathsf{match}(T,t) = \vartheta\}} I[\pi(t), \mathscr{P}\vartheta]} \ (\textsc{c-getl-ta})$$

$$\frac{P \xrightharpoonup[\mathrm{TA}]{\mathbf{qry}(T)@\mathsf{self}} \mathscr{P} \quad K \vdash T = \pi}{I[K,P] \xrightarrow[\mathrm{TA}]{\mathbf{qry}(T)@\mathsf{self}} \sum_{t \,\in\, \{t \in \mathbb{I} \,|\, \mathsf{match}(T,t) = \vartheta\}} I[(\mathcal{X}K) \cdot \pi(t), \mathscr{P}\vartheta]} \ (\textsc{c-qryl-ta})$$

$$\frac{K \ominus T = \bot}{I\,[\,K,\,P\,] \xrightarrow[\mathrm{TA}]{\mathbf{get}(T)@\mathsf{self}} [\,]} \ (\textsc{c-getl-ta}_{\mathrm{B}}) \qquad \frac{K \vdash T = \bot}{I\,[\,K,\,P\,] \xrightarrow[\mathrm{TA}]{\mathbf{qry}(T)@\mathsf{self}} [\,]} \ (\textsc{c-qryl-ta}_{\mathrm{B}})$$

$$\frac{P \xrightharpoonup[\mathrm{TA}]{\mathbf{gq}(T)@\mathsf{p}} \mathscr{P} \quad \mathsf{match}(T,t) = \vartheta}{I[K,P] \xrightarrow[\mathrm{TA}]{\mathbf{gq}(T:t)@\mathsf{p}} I[(\mathcal{X}K), \mathscr{P}\vartheta]} \ (\textsc{c-gqo-ta}) \qquad \frac{P \xrightharpoonup[\mathrm{TA}]{\mathbf{gq}(T)@\mathsf{p}} \mathscr{P} \quad \neg\, \mathsf{match}(T,t)}{I[K,P] \xrightarrow[\mathrm{TA}]{\mathbf{gq}(T:t)@\mathsf{p}} [\,]} \ (\textsc{c-gqo-ta}_{\mathrm{B}})$$

Time-abstract sequences of actions:

$$\frac{P \xrightharpoonup[\mathrm{TA}]{\mathsf{end\_s}} \mathscr{P}}{I\,[\,K,\,P\,] \xrightarrow[\mathrm{TA}]{\overline{\langle\,\rangle}} I[K, \mathscr{P}]} \ (\textsc{c-base-ta}) \qquad \frac{\alpha \in \mathit{OneAct}_{\mathit{Proc}} \quad I\,[\,K,\,P\,] \xrightarrow[\mathrm{TA}]{\alpha} \mathscr{C} \quad \mathscr{C} \xrightarrow[\mathrm{TA}]{\overline{\langle\,\mathit{seq}\,\rangle}}{}^{cv}\, \mathscr{D}}{I\,[\,K,\,P\,] \xrightarrow[\mathrm{TA}]{\overline{\langle\,\alpha.\mathit{seq}\,\rangle}} \mathscr{D}} \ (\textsc{c-rec-ta})$$

Timed sequences of actions:

$$\frac{P \xrightharpoonup{\langle\,\rangle} \mathscr{Q}}{I\,[\,K,\,P\,] \xrightarrow[\mathrm{T}]{\overline{\langle\,\rangle}} I[(\mathcal{X}K), \mathscr{Q}]} \ (\textsc{c-eseq-t})$$

$$\frac{P \xrightharpoonup[\mathrm{TA}]{\mathsf{init\_s}} [\; Q \mapsto 1 \;] \quad I[K,Q] \xrightarrow[\mathrm{TA}]{\overline{\langle\,\mathit{seq}\,\rangle}} I[\mathscr{K}, \mathscr{Q}] \quad \mathscr{Q} \xrightharpoonup{\mathsf{delay}}{}^{cv}\, \mathscr{R}}{I\,[\,K,\,P\,] \xrightarrow[\mathrm{T}]{\overline{\langle\,\mathit{seq}\,\rangle}} I[\mathscr{K}, \mathscr{R}]} \ (\textsc{c-seq-t})$$

Table 14: Operational sem. of StocS components (activ-or), $\xrightarrow[\mathrm{TA}]{}$ and $\xrightarrow[\mathrm{T}]{}$ transition relations.

$\mathsf{Dist}(\mathit{Comp})$ which is defined by the rules in Table 15 and uses its lifting $\xrightarrow[\mathrm{PR}]{}{}^{cv} \subseteq \mathsf{Dist}(\mathit{Comp}) \times \mathit{Act}^{\mathrm{PR}}_{\mathit{Sys}} \times \mathsf{Dist}(\mathit{Comp})$ to probability distributions over components. The set $\mathit{Act}^{\mathrm{PR}}_{\mathit{Sys}}$ of actions is defined as follows:

$$\mathit{Act}^{\mathrm{PR}}_{\mathit{Sys}} \quad ::= \quad \mathit{OneAct}_{\mathit{Proc}} \quad | \quad \langle\, \mathit{Seq}_{\mathit{Sys}} \,\rangle$$

$$\frac{I(K) \models \mathsf{p} \quad K \oplus t = \pi}{I\,[K,\,P] \xrightarrow[\text{PR}]{\mathbf{put}(t)@\mathsf{p}} I[\pi, (\mathcal{X} P)]}\ (\text{C-PUTI-PR}) \qquad \frac{I(K) \not\models \mathsf{p}}{I\,[K,\,P] \xrightarrow[\text{PR}]{\mathbf{put}(t)@\mathsf{p}} [\,I\,[K,\,P] \mapsto 1\,]}\ (\text{C-PUTIR-PR})$$

$$\frac{I(K) \models \mathsf{p} \quad K \ominus T = \pi}{I\,[K,\,P] \xrightarrow[\text{PR}]{\mathbf{get}(T:t)@\mathsf{p}} I[\pi(t), (\mathcal{X} P)]}\ (\text{C-GETI-PR}) \qquad \frac{I(K) \not\models \mathsf{p} \ \vee\ K \ominus T = \bot}{I\,[K,\,P] \xrightarrow[\text{PR}]{\mathbf{get}(T:t)@\mathsf{p}} []}\ (\text{C-GETI-PR}_{\text{B}})$$

$$\frac{I(K) \models \mathsf{p} \quad K \vdash T = \pi}{I\,[K,\,P] \xrightarrow[\text{PR}]{\mathbf{qry}(T:t)@\mathsf{p}} [\,I\,[K,\,P] \mapsto \pi(t)]}\ (\text{C-QRYI-PR}) \qquad \frac{I(K) \not\models \mathsf{p} \ \vee\ K \vdash T = \bot}{I\,[K,\,P] \xrightarrow[\text{PR}]{\mathbf{qry}(T:t)@\mathsf{p}} []}\ (\text{C-QRYI-PR}_{\text{B}})$$

$$\frac{S_1 \xrightarrow[\text{PR}]{\mathbf{put}(t)@\mathsf{p}} \mathscr{S}_1 \quad S_2 \xrightarrow[\text{PR}]{\mathbf{put}(t)@\mathsf{p}} \mathscr{S}_2}{S_1 \parallel S_2 \xrightarrow[\text{PR}]{\mathbf{put}(t)@\mathsf{p}} \mathscr{S}_1 \parallel \mathscr{S}_2}\ (\text{S-PI-PR})$$

$$\frac{S_1 \xrightarrow[\text{PR}]{\mathbf{gq}(T:t)@\mathsf{p}} \mathscr{S}_1 \quad S_2 \xrightarrow[\text{PR}]{\mathbf{gq}(T:t)@\mathsf{p}} \mathscr{S}_2}{S_1 \parallel S_2 \xrightarrow[\text{PR}]{\mathbf{gq}(T:t)@\mathsf{p}} \mathscr{S}_1 \parallel (\mathcal{X}\, S_2) + (\mathcal{X}\, S_1) \parallel \mathscr{S}_2}\ (\text{S-GQI-PR})$$

Sequences of actions:

$$\frac{}{S \xrightarrow[\text{PR}]{\langle\,\rangle} [S \mapsto 1]}\ (\text{S-BASE-PR}) \qquad \frac{\alpha \in \mathit{OneAct}_{\mathit{Proc}} \quad S \xrightarrow[\text{PR}]{\alpha} \mathscr{S} \quad \mathscr{S} \xrightarrow[\text{PR}]{\langle\,\mathit{seq}\,\rangle}{}^{cv}\ \mathscr{T}}{S \xrightarrow[\text{PR}]{\langle\,\alpha\,.\,\mathit{seq}\,\rangle} \mathscr{T}}\ (\text{S-REP-PR})$$

Table 15: Operational sem. of StocS systems and components (ACTIV-OR), $\xrightarrow[\text{PR}]{}$ transition relation.

8 Case Study

We develop a model of a *bike sharing service*, where we assume a city with m *parking stations*, each one with his *location* $\ell_i \in \mathit{Loc} = \{\ell_1, \ldots, \ell_m\}$, a number of *available bikes* b_i, and a number of *available parking slots* s_i (for $i = 1, \ldots, m$). Parking stations are in one-to-one correspondence with the set of possible locations, which should be considered as (disjoint) areas of influence in the city. We also assume to have n *users* of the bike sharing service: at any time, each user is positioned in *one* location and can be in one of the two states *Pedestrian* and *Biker*. In each of the two states, the user moves around the city (with speed depending on the state) according to its preferences, modeled by two *probability* transition matrices Q_{b} and Q_{p} of size $m \times m$ for the biker and the pedestrian state, respectively. Then, the user becomes a *Biker* or a *Pedestrian*, using transitions named *Borrow* and *Return*.

In our model we identify two components: parking stations (containing slots and bikes) and users (that may or may not have a bike), whose population can vary. By relying on the distinguishing features of the StocS language, we model bike/slot retrieval from stations in a distributed and adaptive way (avoiding a central server being aware of bikes/slots availability for each parking). A user in a location ℓ can borrow (or return) a bike by issuing a request (e.g. by means of a mobile phone application) to the bike sharing system for a parking with an available bike (or slot) within a neighborhood of ℓ. The bike sharing system answers with the location of a parking station having an available bike or an available parking slot, within a neighborhood of ℓ specified by a neighborhood condition $\varphi_n(\ell, \ell')$ (modeling, for example, that a parking station ℓ' is easily reachable from ℓ). This flexibility on the choice allows some control on which parking station is selected among those that are in the neighborhood of the

$P_u \triangleq \mathit{Pedestrian}$

$$
\begin{array}{rl@{\qquad}rl}
\mathit{Pedestrian} \triangleq & \mathbf{get}(\texttt{p\_next}, L)@\mathsf{self}. & \mathit{Biker} \triangleq & \mathbf{get}(\texttt{b\_next}, L)@\mathsf{self}. \\
& \mathit{Borrow} & & \mathit{Return} \\
\\
\mathit{Borrow} \triangleq & \mathbf{qry}(\texttt{loc}, L)@\mathsf{self}. & \mathit{Return} \triangleq & \mathbf{qry}(\texttt{loc}, L)@\mathsf{self}. \\
& \mathbf{get}(\texttt{bike\_res}, \mathit{ID})@\mathsf{near}(L). & & \mathbf{get}(\texttt{slot\_res}, \mathit{ID})@\mathsf{near}(L). \\
& \mathbf{put}(\texttt{go}, \mathit{ID})@\mathsf{self}. & & \mathbf{put}(\texttt{go}, \mathit{ID})@\mathsf{self}. \\
& \mathbf{get}(\texttt{bike})@\mathsf{loc}(\mathit{ID}). & & \mathbf{put}(\texttt{bike})@\mathsf{loc}(\mathit{ID}). \\
& \mathbf{put}(\texttt{b})@\mathsf{self}. & & \mathbf{put}(\texttt{p})@\mathsf{self}. \\
& \mathit{Biker} & & \mathit{Pedestrian}
\end{array}
$$

Figure 5: User behavior as a StocS process.

current user location (including itself), which can be used to re-balance slot/bike availability by redirecting users to parking station that have many available bikes (or slots).

In our model, parking stations are themselves in charge of answering to bike/slot requests. This distributed solution is more efficient, robust, and compositional (i.e. the service can scale without changing the infrastructure) than the centralized one. A further advantage is that we can easily model a control over bike redistribution to minimize the cost of bike reallocation by means of trucks. In particular, the choice between parking stations is realized by using different response rates of parking stations: those that have more bikes available will have higher response rates to bikes requests. Similarly, parking stations that have more slots available will have higher response rates to slots requests. By using a **get** action, we put these responses into a race and, on average, users will be redirected to those services that need bike or slot re-balancing. We will discuss this and other features of the model in the rest of the document. In this example we consider the ACT-OR semantics and we assume that message transmission is free of errors, that is $f_{\mathsf{err}}(\sigma, \alpha, \delta) = 0$ for any interface evaluations σ, δ and action α.

We first describe the behavior of the user, given in Figure 5. Then we describe the knowledge operators, defined by rules in Figure 6, and the actions rates, defined by cases in Figure 7, both for the user and the parking station components. The parking stations have no behavior, i.e. they are passive components.

A single user is represented as a component $I_u\,[\,K_u,\ P_u\,]$, whose knowledge state K_u is an element $\langle s, \ell\rangle$ in $\{\mathtt{b}, \mathtt{p}\} \times \mathit{Loc}$ denoting the user state (i.e. either being a pedestrian or a biker) and the user location, and whose interface I_u, which defines the predicates biker, $\mathsf{pedestrian}$, and $\mathsf{loc}(\ell)$ as follows: $I_u(\langle \mathtt{b}, \ell\rangle) \models \mathsf{biker}$, $I_u(\langle \mathtt{p}, \ell\rangle) \models \mathsf{pedestrian}$, and $I_u(\langle s, \ell\rangle) \models \mathsf{loc}(\ell)$, for every $\ell \in \mathit{Loc}$ and $s \in \{\mathtt{b}, \mathtt{p}\}$. Let us summarize the role of the user knowledge operators, defined in Figure 6. The $\oplus$ operator allows: to change state by $\oplus(\mathtt{b})$ (change to biker state) and by $\oplus(\mathtt{p})$ (change to pedestrian state), and to move to a specified location ℓ' by $\oplus(\mathtt{go}, \ell')$. The $\ominus$ operator allows to move to a location according to the average user behavior in the pedestrian state, by $\ominus(\texttt{p\_next}, L)$, and in the biker state, by $\ominus(\texttt{b\_next}, L)$. Finally, the $\vdash$ operator allows to retrieve the current user location.

The users behaviour is given in Figure 5. Each user starts in the state *Pedestrian*, where movement is possible through a local **get** of the item $\texttt{p\_next}$. The choice of the location is resolved internally using the probability matrix $Q_{\mathtt{p}}$, as shown in the knowledge rule $\textsc{kr}^u_4$. The effect of the action is the change of location of the user into ℓ_j, which is also returned as a binding for the variable L. Note that the template is $T = (\texttt{p\_next}, L)$ and the retrieved item is $t = (\texttt{p\_next}, \ell_j)$. The location ℓ_j is used to compute the rate of the action (i.e. of the movement), as shown in the definition $\textsc{rr}^u_3$, specifying the rate function for this specific action.

User:

$$
\begin{array}{llll}
\mathrm{KR}_1^u : \langle s, \ell_i \rangle \xrightarrow{\oplus(\mathtt{b})} [\,\langle \mathtt{b}, \ell_i \rangle \mapsto 1\,] & & \mathrm{KR}_4^u : \langle s, \ell_i \rangle \xrightarrow{\ominus(\mathtt{p\_next},L)} \sum_{j=1}^{m} [\,(\langle s, \ell_j \rangle, (\mathtt{p\_next}, \ell_j)) \mapsto Q_{\mathtt{p}}(i,j)\,] \\
\mathrm{KR}_2^u : \langle s, \ell_i \rangle \xrightarrow{\oplus(\mathtt{p})} [\,\langle \mathtt{p}, \ell_i \rangle \mapsto 1\,] & & \mathrm{KR}_5^u : \langle s, \ell_i \rangle \xrightarrow{\ominus(\mathtt{b\_next},L)} \sum_{j=1}^{m} [\,(\langle s, \ell_j \rangle, (\mathtt{p\_next}, \ell_j)) \mapsto Q_{\mathtt{b}}(i,j)\,] \\
\mathrm{KR}_3^u : \langle s, \ell_i \rangle \xrightarrow{\oplus(\mathtt{go},\ell')} [\,\langle s, \ell' \rangle \mapsto 1\,] & & \mathrm{KR}_6^u : \langle s, \ell_i \rangle \xrightarrow{\vdash(\mathtt{loc},L)} [\,(\mathtt{loc}, \ell_i) \mapsto 1\,]
\end{array}
$$

Parking Station:

$$
\begin{array}{llll}
\mathrm{KR}_1^p : & \langle b_a, b_r, s_a, s_r, \ell_i \rangle & \xrightarrow{\oplus(\mathtt{bike})} & [\,\langle b_a+1, b_r, s_a, s_r-1, \ell_i \rangle \mapsto 1\,] \quad \text{if } s_r > 0 \\
\mathrm{KR}_2^p : & \langle b_a, b_r, s_a, s_r, \ell_i \rangle & \xrightarrow{\ominus(\mathtt{bike\_res},L)} & [\,(\langle b_a-1, b_r+1, s_a, s_r, \ell_i \rangle, (\mathtt{bike\_res}, \ell_i)) \mapsto 1\,] \quad \text{if } b_a > 0 \\
\mathrm{KR}_3^p : & \langle b_a, b_r, s_a, s_r, \ell_i \rangle & \xrightarrow{\ominus(\mathtt{slot\_res},L)} & [\,(\langle b_a, b_r, s_a-1, s_r+1, \ell_i \rangle, (\mathtt{slot\_res}, \ell_i)) \mapsto 1\,] \quad \text{if } s_a > 0 \\
\mathrm{KR}_4^p : & \langle b_a, b_r, s_a, s_r, \ell_i \rangle & \xrightarrow{\ominus(\mathtt{bike})} & [\,(\langle b_a, b_r-1, s_a+1, s_r, \ell_i \rangle, \mathtt{bike}) \mapsto 1\,] \quad \text{if } b_r > 0
\end{array}
$$

Figure 6: Knowledge behavior (User/Parking Station). For $s \in \{\mathtt{b}, \mathtt{p}\}$, $i = 1, \ldots, m$, $\ell' \in \mathit{Loc}$, and $b_a, b_r, s_a, s_r \in \mathbb{N}$.

Local actions:

$$
\mathrm{RR}_1^u : \mathcal{R}(\sigma, \mathbf{put}(\mathsf{go}, \ell')@\mathsf{self}, \sigma) = \begin{cases} \lambda_{\mathtt{b}} \cdot f_{dist}(\ell, \ell') & \text{if } \sigma \models \mathsf{biker} \wedge \mathsf{loc}(\ell) \\ \lambda_{\mathtt{p}} \cdot f_{dist}(\ell, \ell') & \text{if } \sigma \models \mathsf{pedestrian} \wedge \mathsf{loc}(\ell) \end{cases}
$$

$$
\begin{array}{lll}
\mathrm{RR}_2^u : \mathcal{R}(\sigma, \mathbf{put}(\mathtt{p})@\mathsf{self}, \sigma) = \mathcal{R}(\sigma, \mathbf{put}(\mathtt{b})@, \sigma) = \lambda_{fast} \\
\mathrm{RR}_3^u : \mathcal{R}(\sigma, \mathbf{get}((\mathtt{p\_next}, L) : (\mathtt{p\_next}, \ell'))@\mathsf{self}, \sigma) = \lambda_{\mathtt{p}} \cdot f_{distance}(\ell, \ell') \quad \text{if } \sigma \models \mathsf{loc}(\ell) \\
\mathrm{RR}_4^u : \mathcal{R}(\sigma, \mathbf{get}((\mathtt{b\_next}, L) : (\mathtt{p\_next}, \ell'))@\mathsf{self}, \sigma) = \lambda_{\mathtt{b}} \cdot f_{distance}(\ell, \ell') \quad \text{if } \sigma \models \mathsf{loc}(\ell) \\
\mathrm{RR}_5^u : \mathcal{R}(\sigma, \mathbf{qry}((\mathtt{loc}, L) : (\mathtt{loc}, \ell'))@\mathsf{self}, \sigma) = \lambda_{fast}
\end{array}
$$

Remote actions:

$$
\begin{array}{l}
\mathrm{RR}_6^u : \mathcal{R}(\sigma, \mathbf{put}(\mathtt{bike})@\mathsf{loc}(\ell), \delta) = \mathcal{R}(\sigma, \mathbf{get}(\mathtt{bike})@\mathsf{loc}(\ell), \delta) = \lambda_{park} \\
\mathrm{RR}_7^u : \mathcal{R}(\sigma, \mathbf{get}((\mathtt{bike\_res}, L) : (\mathtt{bike\_res}, \ell'))@\mathsf{near}(\ell), \delta) = \frac{b_a}{b_a + s_a} \quad \text{if } \delta \models (\mathtt{bikes} = b_a) \wedge (\mathtt{slots} = s_a) \\
\mathrm{RR}_8^u : \mathcal{R}(\sigma, \mathbf{get}((\mathtt{slot\_res}, L) : (\mathtt{slot\_res}, \ell'))@\mathsf{near}(\ell), \delta) = \frac{s_a}{b_a + s_a} \quad \text{if } \delta \models (\mathtt{bikes} = b_a) \wedge (\mathtt{slots} = s_a)
\end{array}
$$

Figure 7: Rate Function, for any $\ell, \ell' \in \mathit{Loc}$ and $b_a, s_a \in \mathbb{N}$.

We distinguish the movement of the pedestrians from that of the bikers ($\mathtt{p\_next}/\mathtt{b\_next}$) to allow the more realistic assumption that bikers moves more quickly than pedestrians (in definitions RR_3^u and RR_4^u, as well as the two cases of definition RR_1^u, we use different rates $\lambda_{\mathtt{p}}$ and $\lambda_{\mathtt{b}}$). The role of the information made available in the interfaces is important not only for the evaluation of the predicates, but also for the computation of the rates, as shown in Figure 7.

The process *Borrow* first retrieves the current location L (using rule KR_6^u and a rate given by RR_5^u). Then, it performs a *bike* reservation ($\mathtt{bike\_res}$) from a parking station *ID* satisfying predicate $\mathsf{near}(L)$: this triggers on the side of the parking station a knowledge action that decrements the number of available bikes, increments the number of reserved bikes, and returns the id of the selected parking station (by using rule KR_2^p and with rate RR_7^u). Here the

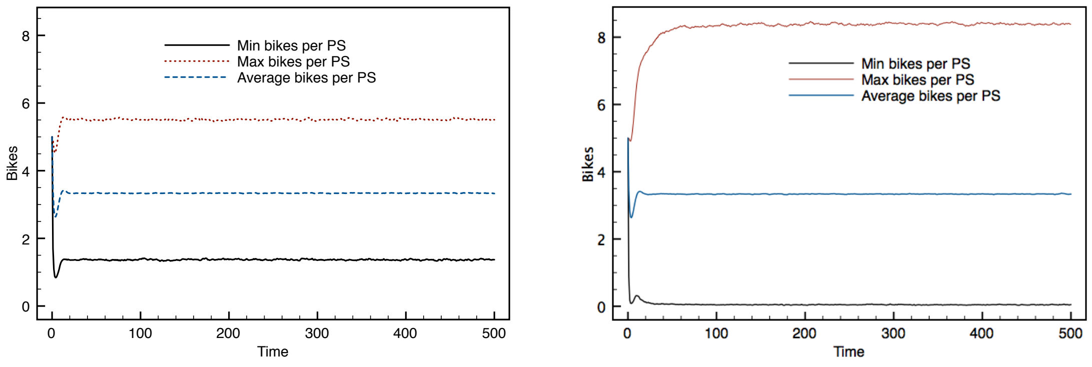

Figure 8: Simulation of bike sharing service.

template is $T = (\texttt{bike\_res}, L)$ and the retrieved item is $t = (\texttt{bike\_res}, \ell_i)$, which produces the binding $L \mapsto \ell_i$. Note that the actual rate of this action depends on available bikes: the higher is the number of available bikes, the higher is the execution rate. As an effect of this race condition, the *ID* of the near station containing *more bikes* is received by the user with a *higher probability* than a near station with *fewer bikes*, causing a more balanced distribution of bikes in the system. When the parking lot is reserved, the user moves towards the parking station (which modifies the knowledge state according to KR_3^u). The rate of this action depends on the distance between the user and the parking station and on the pedestrian/biker state of the user (according to RR_1^u). After moving, the user takes a `bike`: this operation is performed via a **get** action whose effect on the parking knowledge state is given by rule KR_4^p, that decrements the bikes available and increments the slots available (the action has rate given by RR_6^u). Finally, the user status is updated to biker (using KR_1^u and with rate RR_2^u).

Bikers move around the city and, then, leave their bikes in a parking station by executing the process *Return*. Their behavior is similar to that of pedestrians, except for the fact that they perform complementary operations: they use a local **get** of the item `b_next` for moving (see rule KR_5^u and definition RR_4^u) instead of `p_next`, they perform a *slot* reservation `slot_res` (using rule KR_3^p and definition RR_8^u) instead of a bike reservation `bike_next`, and they return a `bike` to a parking station using a **put** action, rather than a **get** action (using rule KR_1^p and definition RR_6^p). After these actions, they return in state *Pedestrian* (using KR_2^u and with rate RR_2^u).

A parking station is represented as a component $I_p[K_p, \mathbf{nil}]$ that has no behavior (it is passive). Its knowledge state is a vector $\langle b_a, b_r, s_a, s_r, \ell\rangle \in \mathbb{N}^4 \times Loc$ denoting the number of available bikes (b_a), of reserved bikes (b_r), of available parking slots (s_a), and of reserved parking slots (s_r), as well as the parking location ℓ. The parking station interface I_p defines the predicates $\mathsf{loc}(\ell)$ and $\mathsf{near}(\ell)$ as follows: $I_p(\langle b_a, b_r, s_a, s_r, \ell\rangle) \models \mathsf{loc}(\ell)$ and $I_p(\langle b_a, b_r, s_a, s_r, \ell\rangle) \models \mathsf{near}(\ell')$ if $\varphi_n(\ell, \ell')$ holds, for every $\ell, \ell' \in Loc$ and $b_a, b_r, s_a, s_r \in \mathbb{N}$.

An initial state of this model is a term

$$\|_{i=1}^{m} \;(\; (I_u[\langle \ell_i, \mathsf{p}\rangle, P_u])[k_i] \parallel I_p[\langle b_i, 0, s_i, 0, \ell_i\rangle, \mathbf{nil}] \;)$$

which denotes, for $i = 1, \ldots, m$: (i) k_i pedestrians in locations ℓ_i, and (ii) b_i available bikes and s_i available parking slots in parking station at location ℓ_i. Note that the number of reserved bikes as well as the number of reserved slots is set to zero at the initial state of the system in every parking station. The overall number of bikes in the system is preserved by the knowledge-update rules. Furthermore, each parking station has a slot+bike capacity set initially by the value $b_i + s_i$ which is never exceeded, again thanks to the rules above.

A fully specified model is obtained by fixing the parameters (all of them in $\mathbb{R}_{\geq 0}$): λ_{fast} denoting a fast rate for internal update actions, λ_{park} denoting a (uniform) interaction rate with the parking (e.g. locking/unlocking the bike), λ_{p} denoting the rate of movement of a pedestrian, and λ_{b} denoting the rate of movement of a biker, and also a *distance* function $f_{dist} : Loc \times Loc \to \mathbb{R}_{\geq 0}$ that encodes the connection topology of the locations. By giving concrete values to those parameters, to populations, and to bikes/slots in the initial state we obtain a finite state CTMC model for the described system.

Preliminary simulation analyses of the considered system have been performed with jRESP [3]. These simulations compare the case where rates of bikes and slots reservations depend on the number of available resources and the case where these rates are constant, and show that the average number of available bikes/slots per parking station is the same, while variance is lower. That is, when the bikes/slots reservation rate depends on the available resources, the bikes are more evenly distributed over the different parking stations.

9 Conclusions and Future Work

We have introduced StocS, a stochastic extension of SCEL, for the modeling and analysis of performance aspects of ensemble based autonomous systems. One of the original features of the language is the use of stochastic predicate based multi-cast communication which poses particular challenges concerning stochastically timed semantics. Four variants of the semantics, considering different levels of abstraction, have been presented and their main aspects of the formal semantics have been provided. A case study concerning shared bikes systems was presented to illustrate the use of the various language primitives of StocS. The development of both numeric and statistical model-checking tools for StocS is in progress. In particular, (ACT-OR) and (INT-OR) semantics are well suited for formal analysis techniques (e.g. probabilistic model-checking), while the more detailed and complex (NET-OR) can be used for simulation-based techniques (e.g. statistical model-checking). The formal relationship between the different semantics is a non trivial issue and it is left for future work, as well as the development of fluid semantics and verification techniques to address large scale collective systems along the lines of work in [2, 3].

References

[1] Luca Bortolussi, Vashti Galpin, and Jane Hillston. Hype with stochastic events. In Mieke Massink and Gethin Norman, editors, *QAPL*, volume 57 of *EPTCS*, pages 120–133, 2011.

[2] Luca Bortolussi and Jane Hillston. Fluid model checking. In Maciej Koutny and Irek Ulidowski, editors, *CONCUR*, volume 7454 of *Lecture Notes in Computer Science*, pages 333–347. Springer, 2012.

[3] Luca Bortolussi, Jane Hillston, Diego Latella, and Mieke Massink. Continuous approximation of collective system behaviour: A tutorial. *Perform. Eval.*, 70(5):317–349, 2013.

[4] Roberto Bruni, Andrea Corradini, Fabio Gadducci, Alberto Lluch-Lafuente, and Andrea Vandin. A conceptual framework for adaptation. In Juan de Lara and Andrea Zisman, editors, *FASE*, volume 7212 of *Lecture Notes in Computer Science*, pages 240–254. Springer, 2012.

[5] Federica Ciocchetta and Jane Hillston. Bio-pepa: A framework for the modelling and analysis of biological systems. *Theor. Comput. Sci.*, 410(33-34):3065–3084, 2009.

[3] `http://jresp.sourceforge.org`

[6] Rocco De Nicola, Gian Luigi Ferrari, Michele Loreti, and Rosario Pugliese. A language-based approach to autonomic computing. In Bernhard Beckert, Ferruccio Damiani, Frank S. de Boer, and Marcello M. Bonsangue, editors, *FMCO*, volume 7542 of *Lecture Notes in Computer Science*, pages 25–48. Springer, 2011.

[7] Rocco De Nicola, Diego Latella, Michele Loreti, and Mieke Massink. Rate-based transition systems for stochastic process calculi. In Susanne Albers, Alberto Marchetti-Spaccamela, Yossi Matias, Sotiris E. Nikoletseas, and Wolfgang Thomas, editors, *ICALP (2)*, volume 5556 of *Lecture Notes in Computer Science*, pages 435–446. Springer, 2009.

[8] Rocco De Nicola, Diego Latella, Michele Loreti, and Mieke Massink. A uniform definition of stochastic process calculi. *ACM Comput. Surv.*, 46(1):5:1–5:35, July 2013.

[9] Vashti Galpin, Jane Hillston, and Federica Ciocchetta. A semi-quantitative equivalence for abstracting from fast reactions. In Ion Petre and Erik P. de Vink, editors, *CompMod*, volume 67 of *EPTCS*, pages 34–49, 2011.

[10] Holger Hermanns and Joost-Pieter Katoen. The how and why of interactive markov chains. In Frank S. de Boer, Marcello M. Bonsangue, Stefan Hallerstede, and Michael Leuschel, editors, *FMCO*, volume 6286 of *Lecture Notes in Computer Science*, pages 311–337. Springer, 2009.

[11] Nora Koch, Matthias Hölzl, Annabelle Klarl, Philip Mayer, Tomas Bures, Jaques Combaz, Alberto Lluch Lafuente, Rocco De Nicola, Stefano Sebastio, Francesco Tiezzi, Andrea Vandin, Fabio Gadducci, Valentina Monreale, Ugo Montanari, Michele Loreti, Carlo Pinciroli, Mariachiara Puviani, Franco Zambonelli, Nikola Šerbedžija, and Emil Vassev. JD3.2: Software engineering for self-aware SCEs, 2013. ASCENS Deliverable JD3.2.

[12] Diego Latella, Michele Loreti, Mieke Massink, and Valerio Senni. On StocS: a Stochastic extension of SCEL. Technical Report 11, ASCENS Project, 2013. `http://www.ascens-ist.eu/`.

[13] Rocco De Nicola, Matthias Hölzl, Michele Loreti, Alberto Lluch Lafuente, Ugo Montanari, Emil Vassev, and Franco Zambonelli. JD2.1: Languages and knowledge models for self-awareness and self-expression, 2012. ASCENS Deliverable JD2.1.

[14] Catuscia Palamidessi. Comparing the expressive power of the synchronous and asynchronous pi-calculi. *Mathematical Structures in Computer Science*, 13(5):685–719, 2003.

[15] Nikola Šerbedžija, Nicklas Hoch, Carlo Pinciroli, Michal Kit, Tomas Bures, Valentina Monreale, Ugo Montanari, Philip Mayer, and José Velasco. D7.3: Third report on wp7 - integration and simulation report for the ascens case studies, 2013. ASCENS Deliverable D7.3.

[16] Nikola Šerbedžija, Mieke Massink, Carlo Pinciroli, Manuele Brambilla, Diego Latella, Marco Dorigo, Mauro Birattari, Philip Mayer, José Angel Velasco, Nicklas Hoch, Henry P. Bensler, Dhaminda Abeywickrama, Jaroslav Keznikl, Ilias Gerostathopoulos, Tomas Bures, Rocco De Nicola, and Michele Loreti. D7.2: Second report on wp7 - integration and simulation report for the ascens case studies, 2012. ASCENS Deliverable D7.2.