The Stochastic Quality Calculus

Kebin Zeng, Flemming Nielson, Hanne Riis Nielson

The Stochastic Quality Calculus

Kebin Zeng, Flemming Nielson, and Hanne Riis Nielson

DTU Compute, Technical University of Denmark, 2800 Kgs. Lyngby

Abstract. We introduce the Stochastic Quality Calculus in order to model and reason about distributed processes that rely on each other in order to achieve their overall behaviour. The calculus supports broadcast communication in a truly concurrent setting. Generally distributed delays are associated with the outputs and at the same time the inputs impose constraints on the waiting times. Consequently, the expected inputs may not be available when needed and therefore the calculus allows to express the absence of data.
The communication delays are expressed by general distributions and the resulting semantics is given in terms of *Generalised Semi-Markov Decision Processes*. By restricting the distributions to be continuous and by allowing truly concurrent communication we eliminate the nondeterminism and arrive at *Generalised Semi-Markov Processes (GSMPs)*; further restriction to exponential distributions gives rise to *numerically analysable GSMPs*, in particular using techniques from stochastic model checking.

1 Introduction

Networked communication is the key for modern *distributed systems* as found in Systems of Systems [1] – encompassing service-oriented systems as well as cyber-physical systems – and including systems that are essential for the infrastructure in the 21^{st} century. *Safety* as well as *security* are key concerns for many of these systems and in particular denial of service attacks have received attention. Massive amounts of requests may be sent to a process thereby making it unavailable for genuine communication and in the case of wireless communication the actual communication may also be disrupted by interference with the frequency band and physically shielding the antennas of senders and receivers.

The Quality Calculus. The classical *"super-optimistic"* programming style of traditional software development no longer suffices – we need to take into account that the expected communications might not occur and that the systems still have to coordinate to the extent possible: we have to turn to a *"realistic-pessimistic"* programming style. The Quality Calculus introduced in [2] is a first step towards a calculus supporting this change of paradigm; the communication

paradigm is point-to-point (as in the π-calculus [3]) and is accompanied by a SAT-based analysis for checking whether the processes are vulnerable to unreliable communication. Probabilistic reasoning is added to the calculus in [4] in a setting where each input binder is annotated with a probability distribution indicating the trustworthiness of the inputs received with respect to a security lattice; a probabilistic trust analysis is then developed in order to identify the extent to which a robust programming style has been adhered to. Furthermore, a broadcast version of the calculus is developed in [5]; here it is additionally extended with cryptographic primitives and the focus is on the development of a rewriting semantics allowing us to reason – in a discrete setting – about unsolicited messages as well as the absence of expected communications.

Our contribution. The Stochastic Quality Calculus (SQC) presented in this paper is an extension of the previous works, and it differs in several aspects. First, it supports *truly concurrent broadcast communication* meaning that several processes may send messages at the same time (also over the same channel) and all processes that are ready to receive these messages must do so. Another main difference it that the *timing aspect* plays a central role. The time for completing a communication depends on the hardware architecture and the communication protocols but also the cyber environment. Hence *real time* considerations are relevant for those communications taking exact duration, and *stochastic time* considerations are relevant for those taking random time influenced by the cyber environment.

In our calculus, we use *generally distributed* random variables to characterise communication delays, so that both continuous stochastic time and real (exact duration) time can be expressed. An output process has the form $t_1!^G t_2.P$ specifying that the value t_2 should be communicated over the channel t_1 within some time determined by the general distribution G. A unique feature of the Stochastic Quality Calculus is an input binder of the form:

$$\&_q^{[a,a')}(c_1?x_1, \cdots, c_n?x_n).$$

It specifies that the process is waiting for n inputs over the channels $c_1, \cdots, c_n$; it is waiting for at least a time units and at most a' time units, where $a < a'$. The quality predicate q determines when sufficient inputs have been achieved and will then allow the process to continue before a' time units have passed (provided that a time units already have passed). The quality predicate q may be $\exists$ meaning that at least one of the n inputs must have been received, it may be $\forall$ meaning that all the n inputs must have been received, but other combinations are also possible. The continuation process will then have to inspect which inputs have been received and take appropriate actions in each case – thereby enforcing the "realistic-pessimistic" programming style alluded to above.

Related work. The challenge of combining concurrency and stochasticity has been addressed in previous stochastic process calculi as PEPA [6] and IMC [7];

the challenge of combining concurrency, stochasticity and mobility have been addressed in the stochastic π-calculus [8] and StoKLAIM [9]; the challenge of combining concurrency and real time have been addressed in timed CCS [10] and PerTiMo [11]. Most stochastic calculi make use of exponential distributions (denoted Exp) to express random delay, and can then use classic techniques and tools for Markov chain analysis. However, it is well-known that Exp distributions often are inadequate to faithfully model many phenomena, where the systems contain real time delays or highly variable distributed durations. The work of e.g. [12, 13] go one step further and incorporate general distributions thereby expressing a rich class of randomness. However, the real time (exact duration) delays are much less frequently incorporated in stochastic process calculi.

A CCS-like process algebraic framework introduced in [14] considers both discrete real time and generally distributed stochastic time. It uses a "spent-lifetime" semantics to track the time passed since activation to perform a race among parallel processes without a clock. Differently, SQC utilizes a clock to keep tracking residual lifetimes for the race, which avoids the complexity of time additivity mentioned in [14]. Besides, the transformational semantics of SQC gives a clear picture of the separation of the stochastic aspect and the real time aspect:

- for local processes, process transitions replace all stochastic time variables by sampled clock values with no consideration of time features;
- for global systems, real timed system transitions generate events and update the clocks with no consideration of stochastic features.

Overview. In Section 2 we introduce the syntax of the Stochastic Quality Calculus. The operational semantics of processes is presented in Section 3; it makes use of general distributions and in Section 4 we show that it amounts to *Generalised Semi-Markov Decision Processes (GSMDPs)* [15] by modelling truly concurrent broadcast communication as discrete events. Some of the discrete events in GSMDPs are controllable, which introduces a decision dimension to execute controllable events nondeterministically; a *policy* (as in Markov decision processes [16]) is introduced to deal with the nondeterminism. The classical problem is then to find an optimal strategy to maximise some reward function on GSMDPs and we refer to [15, 17] for the analysis.

In order to avoid the decision dimension and perform purely stochastic reasoning on the systems, in Section 5 we introduce two *analysable fragments of SQC* that both admit truly concurrent behaviour. The first fragment does not contain the non-determinism thereby obtaining *Generalised Semi-Markov Processes (GSMPs)* [18, 19] that can be analysed using statistical model checking [20, 21]. The second fragment maps further to *numerically analysable GSMPs* [22] that can be analysed using stochastic model checking techniques for Continuous Time Markov Chains [23] and Continuous Stochastic Logic [24]. We conclude and present future work in Section 6.

2 Syntax of SQC

The syntax of the Stochastic Quality Calculus (SQC) consists of *processes* P, input *binders* b and *terms* t, as given in Table 1. A *system* S consists of a number of process definitions and a main process:

$$\begin{array}{ll} \mathsf{define} & A_1 \triangleq P_1 \\ & \vdots \\ & A_n \triangleq P_n \\ \mathsf{in} & P_* \\ \mathsf{using} & c_1, \cdots, c_m \end{array}$$

Here A_i is the name of a process, P_i is its body, P_* is the initial main process and $c_1, \cdots, c_m$ is a list of all the global channel names.

A *process* can have the form $(\nu c)\, P$ introducing a new channel c with scope P, and it can be an *empty process* denoted $\mathsf{0}$; we shall feel free to dispense with trailing occurrences of the process $\mathsf{0}$. An *output process* has the form $t_1!^G t_2.P$ specifying that value t_2 is transmitted over channel t_1 with a communication delay specified by the general distribution G. In SQC, we use *broadcast* transmission, so that all the receivers waiting on channel t_1 receive the value t_2. An *input process* has the general form $b.P$, where b is a *binder* specifying the desired inputs with real time constraints to be satisfied before continuing with P.

A binder may have the form $t?x$ stating that some value should be received over channel t, and stored in the variable x. More generally, a binder has the form $\&_q^I(t_1?x_1, \cdots, t_n?x_n)$ indicating that n inputs are simultaneously active: a *quality predicate* q determines whether sufficient inputs have been received to continue; a non-empty semi-closed *time interval* I determines when control is transferred to the continuation processes. The quality predicate q expresses when enough inputs have been received to continue; it can be $\exists$ meaning that one input is required, or it can be $\forall$ meaning that all inputs are required; formally $\exists(x_1, \cdots, x_n) \Leftrightarrow x_1 \vee \cdots \vee x_n$ and $\forall(x_1, \cdots, x_n) \Leftrightarrow x_1 \wedge \cdots \wedge x_n$. For more expressiveness, we shall allow quality predicates as for example $[1 \vee (2 \wedge 3)](x_1, x_2, x_3) \Leftrightarrow x_1 \vee (x_2 \wedge x_3)$. A non-empty semi-closed time interval I takes the form $[a, a')$ meaning that the binder has the minimum waiting time a and maximum waiting time a' (for $0 \leq a < a' \leq \infty$). As special cases a may be 0 or a' may be ∞ and the subsequent development can be simplified in these cases; we shall feel free to leave the interval field empty when $I = [0, \infty)$.

As an example, the system $\&_{\exists}^{[a,a')}(c_1?x_1, c_2?x_2).P \parallel c_1!^{G_1} t_1 \parallel c_2!^{G_2} t_2$ expresses that two output processes are simultaneously active at time 0, and wait for their output to be accomplished; the quality predicate $\exists$ of the input process will be evaluated at time a for the first time and the input process will continue with P if at least one of the two inputs has arrived. If not, the input process shall wait until one input arrives in the period $[a, a')$; if no input has arrived at

$$
\begin{aligned}
P &::= (\nu c)\, P \;\mid\; \mathsf{0} \;\mid\; t_1!^G t_2.P \;\mid\; b.P \;\mid\; \mathsf{case}\ x\ \mathsf{of}\ \mathsf{some}(y)\colon P_1\ \mathsf{else}\ P_2 \\
&\quad \mid\; P_1 \parallel P_2 \;\mid\; A \\
b &::= t?x \;\mid\; \&_q^I(t_1?x_1, \cdots, t_n?x_n) \\
t &::= y \;\mid\; c
\end{aligned}
$$

Table 1. Syntax of the Stochastic Quality Calculus.

time a', the process shall stop waiting (i.e. time-out) and continue with P even though no input has been received.

In SQC, communication delays are associated with outputs. Thus, a process takes a generally distributed time to send some data out on the channel, but it takes instantaneous time for receiving and storing the data. This is reflected in the syntax where we write $t_1!^G t_2$ for output and $t?x$ for input. Associating delays with outputs rather than inputs is a deliberate design decision that is helpful for identifying analysable fragments of SQC later.

As a consequence of using a general binder, some variables might not obtain proper values as the corresponding inputs are missing. To model this we shall use optional data types as known for example from Standard ML [25]. Let y denote *data variables* and let x denote *optional data variables*. Also, let $\mathsf{some}(\cdots)$ express the presence of some data and none the absence of data. The *case construct* $\mathsf{case}\ x\ \mathsf{of}\ \mathsf{some}(y)\colon P_1\ \mathsf{else}\ P_2$ has the following meaning: if x evaluates to a value $\mathsf{some}(c)$ then we bind c to y and continue with P_1; otherwise x evaluates to none and we continue with P_2.

Continuing with the processes we also have parallel composition $P_1 \parallel P_2$. A process can also be a *recursive call* A to one of the defined processes.

We forbid recursion through parallel composition, thereby ensuring that the resulting semantics has a finite state space. We shall say that $A \triangleq P$ has no recursion through parallel composition if the syntax tree for P does not contain any process name B in a descendant of a $\parallel$ construct, such that B might (perhaps indirectly) call A.

We also forbid the creation of new channels in recursion, so that we have a finite number of channels that can be used. Let $\mathsf{fc}(P)$ be the set of free channel names in the process P; for $A_i \triangleq P_i$ we define $\mathsf{fc}(A_i)$ as the least solution to the simultaneous equation system $\mathsf{fc}(A_i) = \mathsf{fc}(P_i)$ (for $i = 1 \cdots n$). We say that $A \triangleq P$ has no creation of new channels in recursion if the syntax tree for P does not contain any process name in a descendant of a (νc) construct.

Finally, for a system of the form displayed above we shall require that the initial main process P_* as well as the bodies P_i have no free variables over neither data nor optional data, and that their free constants are among $c_1, \cdots, c_m$.

$P \equiv P$	$(\nu c)\, P \equiv P \quad \text{if } c \notin \mathsf{fc}(P)$
$A \equiv P \text{ if } A \triangleq P$	$P_1 \equiv P_2 \Rightarrow C[P_1] \equiv C[P_2]$
$P \parallel \mathsf{0} \equiv P$	$P_1 \parallel P_2 \equiv P_2 \parallel P_1$
$P_1 \equiv P_2 \Rightarrow P_2 \equiv P_1$	$P_1 \equiv P_2 \wedge P_2 \equiv P_3 \Rightarrow P_1 \equiv P_3$
$(\nu c)\,(P_1 \parallel P_2) \equiv ((\nu c)\, P_1) \parallel P_2 \quad \text{if } c \notin \mathsf{fc}(P_2)$	
$(\nu c_1)\,(\nu c_2)\, P \equiv (\nu c_2)\,(\nu c_1)\, P \quad \text{if } c_1 \neq c_2$	

Table 2. The structural congruence.

[INPUT]	$\dfrac{b ::_{\mathsf{tt}} \theta}{b.P \to P\theta}$	[OUTPUT]	$\dfrac{\omega \sim G}{c!^{G} d.P \to c!^{\omega} d.P}$
[CONGRU]	$\dfrac{P \equiv Q \quad Q \to R}{P \to R}$	[PARA]	$\dfrac{P_1 \to P_1'}{P_1 \parallel P_2 \to P_1' \parallel P_2}$
[CASE1]	$\mathsf{case\ some}(c)\ \mathsf{of\ some}(y) \colon P_1\ \mathsf{else}\ P_2 \to P_1[y \mapsto c]$		
[CASE2]	$\mathsf{case\ none\ of\ some}(y) \colon P_1\ \mathsf{else}\ P_2 \to P_2$		

Table 3. Transition relation for processes.

3 Semantics of SQC

The semantics consists of a structural congruence and a transition relation for processes and on top of this we define a transition relation for systems. To facilitate this we need to define the semantics of binders and this includes a transformation, a test on when the binder is satisfied and a transition relation for binders. Throughout this section we need to take care of the generally distributed output delays and the timing requirements.

The structural congruence defined in Table 2 is standard. The contexts C are defined by:

$$C ::= [\,] \;\mid\; (\nu c)\, C \;\mid\; C \parallel P \;\mid\; P \parallel C$$

As usual we apply α-conversion whenever needed to avoid accidental capture of names during substitution.

The transition relation for processes has the form $P \to P'$ and describes how a process P evolves into another process P'. The relation is defined in Table 3 and we notice that the rules [CONGRU] and [PARA] are standard. The two axioms [CASE1] and [CASE2] are straightforward and this leaves us with the rules for input and output; before explaining those we need some preliminaries.

Transforming the binders. We take a transformational approach to give the semantics to binders such that all the real time constraints from the binders are encoded as *Dirac distributed* delays. For clarity we shall write TSQC for the transformed version of SQC and observe that TSQC will be a fragment of SQC.

In the following let $\Re_{\geq 0}$ be the set of nonnegative real numbers, let δ_a denote a Dirac distribution with parameter $a \in \Re_{\geq 0}$, and let ω be a clock value generated according to the δ_a distribution (written $\omega \sim \delta_a$); then we have $\mathbb{P}(\omega \leq a) - \mathbb{P}(\omega < a) = 1$. Thus real time delays can be expressed because a Dirac distribution δ_a is a special case of a general distribution G.

The idea is now to transform an SQC process of the form $\&_q^I(t_1?x_1, \cdots, t_n?x_n).P$ into a process using specific channels for keeping track of the time and to let a modified version of the binder react when the time signals arrive. Depending on the interval I used in $\&_q^I(t_1?x_1, \cdots, t_n?x_n)$, different transformations are considered; here we shall only consider the case where I is $[a, a')$ (and $0 < a < a' < \infty$) and note that simpler transformations are mandatory in the special cases where a is 0 and/or a' is ∞. Then we shall replace the SQC process $\&_q^{[a,a')}(t_1?x_1, \cdots, t_n?x_n).P$ with the TSQC process

$$(\nu c_a)\,(\nu c_{a'})\left(\&_{\dot{q}}(t_1?x_1, \cdots, t_n?x_n, c_a?x_a, c_{a'}?x_{a'}).P \parallel c_a!^{\delta_a}\bullet \parallel c_{a'}!^{\delta_{a'}}\bullet\right),$$

Here two fresh outputs with Dirac distributed delays make use of fresh channels c_a and $c_{a'}$ to send the signals corresponding to the beginning and the end of the time interval. The binder itself is modified to listen for these signals and x_a and $x_{a'}$ are fresh variables used only to store the real time signal $\bullet$. Finally, the quality predicate is modified to record this by taking $\dot{q}(x_1, \cdots, x_n, x_a, x_{a'}) = (q(x_1, \cdots, x_n) \wedge x_a) \vee x_{a'}$.

We shall apply the above transformation rules for all the binders in the system, and we call $\&_{\dot{q}}(t_1?x_1, \cdots, t_n?x_n, c_a?x_a, c_{a'}?x_{a'})$ a *transformed binder* of TSQC. To shorten the notation (by keeping $c_a?x_a$, $c_{a'}?x_{a'}$ implicit), we write $\&_{\dot{q}}(t_1?x_1, \cdots, t_n?x_n)$ for the transformed binder.

Evaluation of binders. Now we introduce the relation

$$b ::_v \theta$$

to record whether all required inputs in b have been performed, by means of $v \in \{\mathsf{tt}, \mathsf{ff}\}$, as well as the composite substitution that has been constructed, by means of θ; it is defined in Table 4 and assumes that we have extended syntax

$$\begin{aligned} d &::= c \mid \bullet \\ b &::= t?x \mid \&_{\dot{q}}(sb_1, \cdots, sb_n) \\ sb &::= t?x \mid [x \mapsto \mathsf{some}(d)] \end{aligned}$$

$$t?x ::_{\mathsf{ff}} [x \mapsto \mathsf{none}] \quad [x \mapsto \mathsf{some}(c)] ::_{\mathsf{tt}} [x \mapsto \mathsf{some}(c)] \quad [x \mapsto \mathsf{some}(\bullet)] ::_{\mathsf{tt}} [x \mapsto \mathsf{some}(\bullet)]$$

$$\frac{sb_1 ::_{v_1} \theta_1 \quad \cdots \quad sb_n ::_{v_n} \theta_n}{\&_{\dot{q}}(sb_1, \cdots, sb_n) ::_v \theta_n \cdots \theta_1}, \quad \text{where } v = \dot{q}(v_1, \cdots, v_n).$$

Table 4. Evaluation of binders.

$$\text{[AX1]}\ (c_i!^{\omega} d_i)_{i=1}^m \vdash c?x \overset{\omega,(c_i)_{i=1}^m}{\longrightarrow} [x \mapsto \mathsf{some}(d_i)],\ \text{if } c = c_i$$

$$\text{[AX2]}\ (c_i!^{\omega} d_i)_{i=1}^m \vdash c?x \overset{\omega,(c_i)_{i=1}^m}{\longrightarrow} c?x,\ \text{if } c \notin \{c_1, \ldots, c_m\}$$

$$\text{[AX3]}\ (c_i!^{\omega} d_i)_{i=1}^m \vdash [x \mapsto \mathsf{some}(d)] \overset{\omega,(c_i)_{i=1}^m}{\longrightarrow} [x \mapsto \mathsf{some}(d)]$$

$$\text{[AX4]}\ \frac{\bigwedge_{j \in \{1,\ldots,n\}} (c_i!^{\omega} d_i)_{i=1}^m \vdash sb_j \overset{\omega,(c_i)_{i=1}^m}{\longrightarrow} sb'_j}{(c_i!^{\omega} d_i)_{i=1}^m \vdash \&_{\dot{q}}(sb_1, \cdots, sb_n) \overset{\omega,(c_i)_{i=1}^m}{\longrightarrow} \&_{\dot{q}}(sb'_1, \cdots, sb'_n)}.$$

Table 5. Transition relation for binders.

where $[x \mapsto \mathsf{some}(d)]$ is the substitution θ that maps the variable x to $\mathsf{some}(d)$. We shall use id for the identity substitution and $\theta_2\theta_1$ for the composition of two substitutions, so $(\theta_2\theta_1)(x) = \theta_2(\theta_1(x))$ for all x.

Returning to the definition of the transition relation $P \to P'$ in Table 3 we now see that the rule [INPUT] simply checks whether the binder is satisfied and if so it will apply the corresponding substitution to the continuation process.

Active and inactive outputs. To define the transition relation for binders we shall distinguish between *active* and *inactive* outputs. An output is *active* when it starts to send data over some channel, otherwise it is *inactive*. As soon as an output becomes active, we replace its delay distribution G with a clock $\omega \in \Re_{\geq 0}$ to track the residual time for the output to complete. The clock ω is initialised by a sampled nonnegative value according to the distribution of the channel (written $\omega \sim G$). Note that the case where $\omega = 0$ indicates that the output completes instantaneously. To formalise that an inactive output becomes active, we extend the syntax by replacing the delay distribution with an explicit clock value like $c!^{\omega} d$. This is depicted by the rule [OUTPUT] in Table 3.

Furthermore, we assume that all the distributed processes share a global clock, and that the time of an output will be updated after each system transition (to be introduced below). In this manner, all active outputs implicitly remember the entire history of the time that has passed; we shall discuss this point later when we define the rule for system transitions.

$$\text{[MATCH]}\quad \frac{(c_i!^{\omega} d_i)_{i\in I} \vdash b_k \overset{\omega,(c_i)_{i\in I}}{\longrightarrow} b'_k \text{ for all } k \in K}{\mathsf{define}\cdots\mathsf{in}\ P\ \mathsf{using}\ \boldsymbol{f} \overset{\omega,(c_i)_{i\in I}}{\Longrightarrow} \mathsf{define}\cdots\mathsf{in}\ P'\ \mathsf{using}\ \boldsymbol{f},\boldsymbol{e}}\ ,\ \text{if}\begin{cases}\{\boldsymbol{f}\}\cap\{\boldsymbol{e}\}=\emptyset,\\ I\neq\emptyset,\end{cases}$$

where $P \to^{*} \equiv (\nu \boldsymbol{e})\left((\parallel_{i\in I} c_i!^{\omega} d_i.P_i) \parallel (\parallel_{j\in J} c_j!^{\omega+\omega_j} d_j.P_j) \parallel (\parallel_{k\in K} b_k.P_k)\right)$ for $\omega_j \in \Re_{>0}$,
and $\quad P' = (\parallel_{i\in I} P_i) \parallel (\parallel_{j\in J} c_j!^{\omega_j} d_j.P_j) \parallel (\parallel_{k\in K} b'_k.P_k)$.

Table 6. Transition relation for systems.

Transition relation for binders. The transition relation for binders takes the form

$$(c_i!^{\omega} d_i)_{i=1}^{m} \vdash b \overset{\omega,(c_i)_{i=1}^{m}}{\longrightarrow} b'$$

where $(c_i!^{\omega} d_i)_{i=1}^{m}$ records the m concurrently active outputs that will modify the binder b so that it becomes b' and where the annotation $\omega, (c_i)_{i=1}^{m}$ records the communication over the channels at the specified time. The relation is defined by the axioms [AX1] to [AX4] in Table 5 and is explained below.

The first axiom [AX1] records the change of a simple input $c?x$ by receiving some value d_i from the channel c_i in $(c_i)_{i=1}^{m}$ at time ω. In the special case where the $(c_i!^{\omega} d_i)_{i=1}^{m}$ contains both $c!^{\omega} d_1$ and $c!^{\omega} d_2$ (i.e. two broadcasts take place over the same channel at the same moment), it is nondeterministic whether d_1 or d_2 shall be received for the input, so the axiom specialises to

$$c!^{\omega} d_1, c!^{\omega} d_2 \vdash c?x \overset{\omega,(c,c)}{\longrightarrow} [x \mapsto \mathsf{some}(d_i)],\ \text{where } i \in \{1,2\}.$$

This is a deliberate design decision and alternatives, as for example that none of the two values are received due to the collision, can be modelled if wanted.

The axiom [AX2] records that no data is received for a simple input at time ω because of the mismatch between inputs and outputs whereas [AX3] records that an already received input retains its value. From the individual records, we express the change of a transformed binder as the collection of individual input changes as recorded in the rule [AX4].

The transition relation for systems. Let us now turn to the semantics of systems; it is defined in Table 6 by a transition relation of the form

$$\mathsf{define}\cdots\mathsf{in}\ P\ \mathsf{using}\ \boldsymbol{f} \overset{\omega,(c_i)_{i\in I}}{\Longrightarrow} \mathsf{define}\cdots\mathsf{in}\ P'\ \mathsf{using}\ \boldsymbol{f'}$$

The annotation on the arrow describes the truly concurrent communications $(c_i)_{i\in I}$ happening at time ω. The rule [MATCH] expresses that the main process P of the system first executes all the local steps from each distributed component; for this we use the reflexive transitive closure of the transition relation for

processes $\rightarrow^*$ composed with the structural congruence $\equiv$. Note that the execution of the subprocesses may introduce new channels into the system; they are denoted $\boldsymbol{e}$.

The distributed components of the processes resulting from this can be classified into three cases:

(1) $\|_{i\in I} c_i!^{\omega} d_i.P_i$ are processes that will perform outputs at time ω;
(2) $\|_{j\in J}\ c_j!^{\omega+\omega_j} d_j.P_j$ are the processes that will perform outputs at time $\omega + \omega_j$ for $\omega_j \in \Re_{>0}$ and hence later than at time ω; and
(3) $\|_{k\in K} b_k.P_k$ are the processes that are ready to perform inputs.

Given a set of broadcasts over $(c_i)_{i\in I}$ at time ω, the resulting system will be composed of the processes obtained from (1), (2) and (3). The processes of (1) will have completed their outputs so they will become $\|_{i\in I} P_i$. The processes of (2) are still waiting to perform their outputs and their clock values will now be reduced so they become $\|_{j\in J} c_j!^{\omega_j} d_j.P_j$. Finally, the processes of (3) will simply be updated to record which inputs they have performed as in $\|_{k\in K} b'_k.P_k$.

This completes the semantics of SQC as transformed into TSQC.

4 SQC is GSMDP

The transition systems obtained from the operational semantics turn out to constitute Generalised Semi-Markov Decision Processes (GSMDPs).

Definition 1. *A Generalised Semi-Markov Decision Process (motivated by [15]) is a tuple* $(S, E, A, \{E_s\}_{s\in S}, \{G_e\}_{e\in E}, \{\Omega_e\}_{e\in E}, \Longrightarrow)$*, where*

- S *is a finite and non-empty set of states,*
- E *is a finite and non-empty set of events,*
- $A \subseteq E$ *is a set of so-called controllable events,*
- $E_s \subseteq E$ *specifies for a state* $s \in S$ *a set of events enabled at* s*,*
- G_e *specifies for an event* $e \in E$ *a general probability distribution,*
- Ω_e *specifies for an event* $e \in E$ *a real-valued clock; when the event* e *becomes enabled,* $\omega_e \in \Re_{\geq 0}$ *is initialised by sampling from the distribution* G_e *to express the residual time until event* e *occurs,*
- $\Longrightarrow$: $S \times \{\Omega_e\}_{e\in E} \times E \times S$ *is a transition relation: given* $s, s' \in S$*,* $\omega \in \{\Omega_e\}_{e\in E}$*,* $e \in E_s$*, the transition* $s \overset{\omega,e}{\Longrightarrow} s'$ *expresses that a transition from* s *to* s' *is triggered by an event* e *with* ω *being the sojourn time in state* s*; the sojourn time* ω *in state* s *is determined by the smallest clock value over* $\{\Omega_e\}_{e\in E_s}$*.*

The dynamics of GSMDPs are described by *execution paths*. An execution path ρ for a GSMDP is a sequence $\rho = s_0 \overset{\omega_0,e_0}{\Longrightarrow} s_1 \overset{\omega_1,e_1}{\Longrightarrow} s_2 \overset{\omega_2,e_2}{\Longrightarrow} \cdots$ with $s_i \in S$, $e_i \in E_{s_i}$ and $\omega_i \geq 0$ being the sojourn time in state s_i. The *length* of an execution path equals the number of transitions along the path, which can be either finite or infinite. The decision dimension of GSMDPs arises in the situation where $s \overset{\omega,e}{\Longrightarrow} s_1$, $s \overset{\omega,e}{\Longrightarrow} s_2$ and $s_1 \neq s_2$. A *policy* (as in Markov Decision Processes [16]) picks an event in the set of enabled controllable events of a state to generate a partial execution path. Thereafter, the optimal strategy over different policy executions over a GSMDP can be obtained using the techniques of [15, 17].

For a system expressed in TSQC, we model a sequence (more precisely a multiset) of truly concurrent broadcast communications as an event of a GSMDP. Therefore, when more than one output take place at the same time on the same channel, there exists a nondeterministic choice of the potential successive system configurations.

Theorem 1. *The semantics of systems in SQC amounts to GSMDPs.*

Proof. To see this we shall now define a GSMDP for a system of the Stochastic Quality Calculus as transformed into TSQC. First let $S/\equiv$ denote the quotient set obtained by using the structural congruence $\equiv$ over a set S of systems. We then define the GSMDP $(S, E, A, \{E_s\}_{s\in S}, \{G_e\}_{e\in E}, \{\Omega_e\}_{e\in E}, \Longrightarrow)$ as follows:

- The set of states S is the quotient
  $\{\mathsf{define}\cdots\mathsf{in}\ P\ \mathsf{using}\cdots \mid \mathsf{define}\cdots\mathsf{in}\ P\ \mathsf{using}\cdots$ is generated by the rules in Table 6 from $\mathsf{define}\cdots\mathsf{in}\ P_*\ \mathsf{using}\cdots\ \}/\equiv$.
  The obtained state space is finite, because recursion through parallel composition is forbidden.
- An event e in the state $s = \mathsf{define}\cdots\mathsf{in}\ P\ \mathsf{using}\cdots$ is defined as a sequence $(c_i)_{i\in I}$ such that each c_i is used as an active output $c_i!^{\omega}d_i$ in the body P of s and all c_i share the same clock value ω. The set E_s is the set of events e enabled in the state s.
- The set of events is the union of all the enabled events: $E = \bigcup_{s\in S} E_s$.
- The set of controllable events is the union: $A = \bigcup_{s\in S} A_s$, where $A_s = \{e \mid s \overset{\omega,e}{\Longrightarrow} s_1, s \overset{\omega,e}{\Longrightarrow} s_2$ and $s_1 \neq s_2$ for some $s, s_1, s_2, \omega,$ and $e \in E_s\}$.
- The distribution of an event is defined as follows:
  - if the event is c where $c!^{G}d$ occurs in $\mathsf{define}\cdots\mathsf{in}\ P_*\ \mathsf{using}\cdots$, the distribution is G;
  - if the event is $(c_i)_{i\in I}$ where $(c_i!^{\omega}d_i)_{i\in I}$ occurs in $\mathsf{define}\cdots\mathsf{in}\ P_*\ \mathsf{using}\cdots$, the distribution of the event $(c_i)_{i\in I}$ is δ_ω;
  - for all c_a where c_a is introduced by the transformation of a binder b in $\mathsf{define}\cdots\mathsf{in}\ P_*\ \mathsf{using}\cdots$, the distribution is δ_a.
- The real valued clock Ω_e associated with the event e is ω whenever e is a sequence $(c_i)_{i\in I}$ where $(c_i!^{\omega}d_i)_{i\in I}$ occurs in $\mathsf{define}\cdots\mathsf{in}\ \mathsf{P}_*\ \mathsf{using}\cdots$, or is introduced by the transformation of a binder in the same system.

– $\Longrightarrow = \{ \overset{\omega,(c_i)_{i\in I}}{\Longrightarrow} \mid \overset{\omega,(c_i)_{i\in I}}{\Longrightarrow}$ generated by the rules in Table 6$\}$. □

The GSMDP semantics of SQC captures the most general behaviour of the systems expressed by the Stochastic Quality Calculus:

1. Output makes use of general probability distributions (expressing both stochastic and real time delay), and runs concurrently using clocks.
2. Input makes use of stochastic quality binders (expressing both real time and quality constraints), and enforces the "realistic-pessimistic" programming style.
3. A sequence of broadcast communications may occur at the same time, such that the data is determined nondeterministically when more than one communication makes use the same channel at the same time.

The classical analysis of GSMDPs is to find an optimal strategy to maximise some reward function, which may use techniques such as policy iteration [16,17].

5 Analysable Fragments of SQC

In order to perform more standard stochastic reasoning on the systems, we introduce two analysable fragments of SQC sitting at different levels of generality, so that more established analysing methods become available.

5.1 SQC with continuous distributions

To exclude the nondeterminism, we introduce a fragment of SQC disallowing non-continuous distributions for outputs.

Definition 2. *The* Stochastic Quality Calculus with continuous distributions (SQC_{con}) *is the untransformed Stochastic Quality Calculus in Table 1 with the condition that all the probability distributions for the outputs have continuous Cumulative Density Functions (CDFs).*

Note that the continuous CDF restriction of SQC_{con} does not exclude the real time (Dirac) output delays that are created during the binder transformation and that are explicit in the TSQC form of SQC_{con}. However, these real time (Dirac) output delays take place over freshly created channels. Therefore, when more than one binder in SQC_{con} uses the same real time constraint, more than one communication may occur at the same time, but only over freshly created channels. Hence, by allowing truly concurrent transitions where all such communications take place at the same time we avoid any non-deterministic behaviour.

Proposition 1. *The system expressed by SQC_{con} does not have nondeterministic communications.*

Proof. By the continuity of continuous probability distributions (be aware that Dirac distributions do not have continuous CDFs), the probability of two continuous random delays to be exactly the same is strictly zero, and hence we feel free to ignore the possibility that more than one output complete at the same time using the same channel. Hence nondeterministic communications are excluded (with probability 1). □

Without the nondeterminism, the semantics of $\mathrm{SQC}_{\mathrm{con}}$ turns out to become a Generalised Semi-Markov Process, which is a subclass of GSMDPs.

Definition 3. *A Generalised Semi-Markov Process (GSMP) (motivated by [19]) is a Generalised Semi-Markov Decision Process (cf. Definition 1) with the controllable event set $A = \emptyset$ and the transition relation $\Longrightarrow$ becoming a functional relation (meaning that $s \overset{\omega,e}{\Longrightarrow} s_1$ and $s \overset{\omega,e}{\Longrightarrow} s_2$ imply $s_1 = s_2$).*

A GSMP can be analysed using discrete event simulation or statistical model checking. We refer to [17, 20] for the technical details.

Theorem 2. *The semantics of systems in the Stochastic Quality Calculus with continuous distributions amounts to GSMPs.*

Proof. The semantics of SQC is a GSMDP, and $\mathrm{SQC}_{\mathrm{con}}$ is a fragment of SQC without the nondeterminism. Removing the nondeterminism from a GSMDP, one obtains a GSMP system corresponding to $\mathrm{SQC}_{\mathrm{con}}$. The construction of the GSMP from $\mathrm{SQC}_{\mathrm{con}}$ follows Theorem 1, and we shall omit the details due to space limitation. □

5.2 SQC with exponential distributions

Despite the fact that statistical methods can be used to analyse the GSMPs obtained from $\mathrm{SQC}_{\mathrm{con}}$, the general continuous probability distributions make numerical methods infeasible. Statistical methods scale better with the size of the state space, however a precise result often requires a large number of samples. Therefore, for state space $|S| < 10^5$ (based on a set of case studies over CTMCs in [21]), numerical methods seem to be preferable. To enable numerical methods, we introduce another fragment of SQC by restricting output delays further to be only exponentially distributed (denoted Exp). We first introduce a subclass of GSMPs with only Exp and real time events, such that the numerical analysis approach in [22] is applicable, which indicates that numerical verification techniques like stochastic model checking are also applicable.

Definition 4. *A numerically analysable GSMP is a generalised semi-Markov process, such that the general probability distributions are restricted to* Exp *distributions and Dirac distributions.*

Numerical analysable GSMPs retain both stochastic (Exp distributions) and real time (Dirac distributions) behaviour. Next, we introduce a fragment of SQC that amounts to numerically analysable GSMP.

Definition 5. *The* Stochastic Quality Calculus with Exp distributions ($\mathrm{SQC}_{\mathrm{exp}}$) *is the untransformed Stochastic Quality Calculus in Table 1 with the condition that all the probability distributions for the outputs are exponentially distributed (denoted* Exp*).*

Theorem 3. *The semantics of systems in the Stochastic Quality Calculus with* Exp *distributions amounts to numerically analysable GSMPs.*

Proof. Theorem 2 shows that the semantics of $\mathrm{SQC}_{\mathrm{con}}$ amounts to GSMPs, where the probability distributions have continuous CDFs. By Definition 5, the systems in $\mathrm{SQC}_{\mathrm{exp}}$ are special cases of $\mathrm{SQC}_{\mathrm{con}}$, where the continuous probability distributions are restricted to be Exp distributions, which amounts to numerically analysable GSMPs by definition. □

Both analysable fragments of SQC retain real time constraints on inputs to enforce the "realistic-pessimistic" programming style. Therefore, it is not possible to reduce the semantics further into continuous–time Markov chains. However, the stochastic behaviour (Exp distributed) and the real time behaviour (Dirac distributed) can often be split into a pure stochastic behaviour (Exp distributed) in the form of a Continuous Time Markov Chain [23] and a pure real time behaviour (Dirac distributed) embodied in a query in Continuous Stochastic Logic [24]. These ideas are developed in [26].

6 Conclusion

The motivation behind the Quality Calculus [2] is that many of the security errors in modern distributed systems are due to an overly "optimistic" programming style. Programmers tend to think of benign communication environments, and hence focus on getting the software to perform as many functions as possible. To a much lesser extent, they consider malign environments and the need to focus on avoiding errors that can be provoked by outside attackers. We believe future programming languages need to support a more robust ("pessimistic") programming style: What conceivably might go wrong, probably will go wrong.

A major cause of disruption is due to the *networked communication* between distributed software components.

This paper developed the Stochastic Quality Calculus (SQC) to model and reason about truly concurrent broadcast communications. A distinguishing feature of SQC is to consider both stochastic and real time communications that express the influences from cyber physical environment. Furthermore, the stochastic quality binder enforces the "realistic-pessimistic" programming style: to program the ideal as well as default behaviour depending on the availability or absence of desired data.

The semantics of SQC amounts to generalised semi-Markov decision processes. To avoid the decision dimension and enable purely stochastic analysis, we introduce two analysable fragments of SQC to express the systems in different levels of generality. The first fragment of SQC amounts to Generalised Semi-Markov Processes (GSMP) that can be analysed using statistical model checking, while the second amounts to a numerically analysable GSMP that can be further analysed using stochastic model checking.

These analyses lay a foundation for supporting a new discipline of robust programming. We believe that with the quantitative information obtained from the analysis, it will be possible to better determine whether or not the software continues to deal appropriately with risks and threats in the new application environment.

Acknowledgment. The research has been supported by IDEA4CPS, granted by the Danish Research Foundations for Basic Research (DNRF86-10). The research question addressed was largely motivated by the European Artemis project SESAMO (www.SESAMO-PROJECT.eu).

References

[1] CONNECT, U.A.D.: Report from the European Union workshop on Directions in Systems of Systems Engineering as part of Horizon 2012. (July 2012)
[2] Riis Nielson, H., Nielson, F., Vigo, R.: A calculus for quality. In: Formal Aspects of Component Software. Volume 7684 of LNCS. Springer Berlin Heidelberg (2013) 188–204
[3] Milner, R.: Communicating and Mobile Systems: the Pi-Calculus. Cambridge University Press (1999)
[4] Riis Nielson, H., Nielson, F.: Probabilistic analysis of the quality calculus. In: Formal Techniques for Distributed Systems. Volume 7892 of LNCS. Springer Berlin Heidelberg (2013) 258–272
[5] Vigo, R., Nielson, F., Riis Nielson, H.: Broadcast, denial-of-service, and secure communication. In: IFM. Volume 7940 of LNCS., Springer (2013) 412–427
[6] Hillston, J.: A compositional approach to performance modelling. Cambridge University Press, New York, NY, USA (1996)

[7] Brinksma, E., Hermanns, H.: Process Algebra and Markov Chains. Springer-Verlag Berlin Heidelberg (2001) 183–231
[8] Priami, C.: Stochastic π-calculus. The Computer Journal **38**(7) (1995) 578–589
[9] De Nicola, R., Katoen, J.P., Latella, D., Massink, M.: Stoklaim: A stochastic extension of klaim. CNR-ISTI Technical Report number ISTI-2006-TR-01 (2006)
[10] Yi, W.: CCS + time= an interleaving model for real time systems. In: Automata, Languages and Programming. Springer (1991) 217–228
[11] Ciobanu, G., Koutny, M.: PerTiMo: A Model of Spatial Migration with Safe Access Permissions. Newcastle University, Computing Science (2011)
[12] Bravetti, M., Bernardo, M., Gorrieri, R.: Towards performance evaluation with general distributions in process algebras. In: CONCUR'98 Concurrency Theory. Springer (1998) 405–422
[13] Nielsen, B.F., Nielson, F., Riis Nielson, H.: Model checking multivariate state rewards. In: QEST 2010, Seventh International Conference on the Quantitative Evaluation of Systems, IEEE Computer Society (2010) 7–16
[14] Markovski, J.: Real and stochastic time in process algebras for performance evaluation. PhD thesis, Ph. D. Thesis, Eindhoven University of Technology (2008)
[15] Doshi, B.T.: Generalized semi-markov decision processes. Journal of Applied Probability (1979) 618–630
[16] Puterman, M.L.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. 1st edn. John Wiley & Sons, Inc., New York, NY, USA (1994)
[17] Younes, H.L., Simmons, R.G.: Solving generalized semi-markov decision processes using continuous phase-type distributions. In: Proceedings of the national conference on artificial intelligence. (2004) 742–748
[18] Matthes, K.: Zur theorie der bedienungsprozesse. In: Trans. of the 3rd Prague Conf. on Information Theory, Stat. Dec. Fns. and Random Processes. (1962) 513–528
[19] Glynn, P.W.: A GSMP formalism for discrete event systems. Proceedings of the IEEE **77**(1) (1989) 14–23
[20] Younes, H.L.: Ymer: A statistical model checker. In Etessami, K., Rajamani, S., eds.: Computer Aided Verification. Volume 3576 of LNCS. Springer Berlin Heidelberg (2005) 429–433
[21] Younes, H., Kwiatkowska, M., Norman, G., Parker, D.: Numerical vs. statistical probabilistic model checking. International Journal on Software Tools for Technology Transfer (STTT) **8**(3) (2006) 216–228
[22] Lindemann, C., Thümmler, A.: Numerical Analysis of Generalized Semi-Markov Processes. Dekanat Informatik, Univ. (1999)
[23] Kwiatkowska, M., Norman, G., Parker, D.: Stochastic model checking. In Bernardo, M., Hillston, J., eds.: Formal Methods for Performance Evaluation. Volume 4486 of Lecture Notes in Computer Science., Springer (2007) 220–270
[24] Aziz, A., Sanwal, K., Singhal, V., Brayton, R.K.: Verifying continuous time Markov chains. In: Computer Aided Verification, CAV '96. Volume 1102 of Lecture Notes in Computer Science., Springer (1996) 269–276
[25] Milner, R.: A proposal for standard ML. In: Proceedings of the 1984 ACM Symposium on LISP and functional programming, ACM (1984) 184–197
[26] Nielson, F., Nielson, H.R., Zeng, K.: Stochastic Model Checking for the Stochastic Quality Calculus. In: Submitted for Publication. (2014)