Performance Analysis and True Concurrency Semantics

(Extended Abstract)

Ed Brinksma, Joost-Pieter Katoen, Rom Langerak
Dept. of Computing Science
University of Twente
P.O. Box 217, 7500 AE Enschede, The Netherlands

Diego Latella
CNR, Ist. CNUCE
Via Santa Maria 36, 56126 Pisa, Italy

Abstract

This paper addresses the subject of linking functional specifications to performance analysis in a process algebraic context. It presents a timed, probabilistic extension of a process algebraic formalism and its application to performance analysis. More specifically, an extension of a subset of LOTOS is presented equipped with a truly concurrent semantical model based on bundle event structures. It is investigated how semi-Markov chains can be obtained from functional specifications using this semantical model. The use of a true concurrency model enables us to distinguish between non-determinism and parallelism, to reduce the state explosion problem and, moreover, to analyse part of the system without considering other (irrelevant) parts. An example illustrates the proposed approach.

1 Introduction

The study of formal methods for the specification, design, and analysis of distributed systems has been an important research topic over the past decade. Initially, the research in this area has concentrated on the dynamic, functional aspects of such systems, such as their observable behaviour, control flow, and synchronization as properties in relative time. More recently, formal methods for the representation and analysis of such properties in combination with quantitative aspects of system behaviour have come into focus. They include the study of timed, or real-time, systems, where actions can be constrained by conditions on their occurrence in an absolute time frame. Other formalisms aim at the representation of stochastic systems, where actions may be assigned a probability of occurring.

The greater expressive power of such new formalisms – which often are extensions of existing non-timed, non-stochastic models – is partly motivated by the need to define and analyse system functionalities that refer to time and/or probability. Another reason is provided by the necessity of analyzing the performance characteristics of systems designs, such as average response times, jitter, noise, etc. This can be (although it does not need to be) independent of functionality requirements, e.g. in order to establish the efficiency of design alternatives. Often, performance analysis plays a crucial role in the design of distributed systems, and its well-developed methods and techniques have penetrated real-world system design.

So far, models for the performance analysis of distributed systems, such as queuing networks or stochastic Petri nets, are typically obtained in a rather informal way from (functional) system specifications. The resulting weak link between the functional and the performance model of the system makes it difficult to relate the two, and generally results in an independent treatment of the correctness and the performance of a design. Although such a separation of concerns may sometimes be useful, there is a price to pay for a lack of coordination between these different forms of design validation. Functional analysis typically precedes performance analysis, and so it can and does happen that designs are rejected because of unsatisfactory performance characteristics only after considerable effort has been invested in showing their functional correctness. Also, it would be very useful to have well-understood and effective ways of deriving performance models from system specifications. Because of the growing availability of formal specifications such derivations can be made (more) precise in terms of the semantic models of the formalisms that are used.

In this paper we present a timed, stochastic extension of a process algebraic formalism and its application to performance analysis. In particular, it is our goal to see how well the underlying semantic model can be used to obtain semi-Markov chains from functional specifications. Semi-Markov chains are a generalization of ordinary Markov chains, and belong to the standard machinery of performance analysis.

Our formalism is an extension of a subset of LOTOS [2], a standardized formal description technique that is closely related to process algebraic languages as CCS [11] and CSP [6]. We equip this language with a *truly concurrent* semantic model, in contrast with the more traditional semantic models based on the *interleaving* of independent actions. The former class of models retains explicit information about the parallelism between system components. As performance models typically are based on abstractions of the control and/or data flow structure of the systems, the use of true concurrency semantics is thought to be a direct way of narrowing the gap with functional models. Additional advantages of these semantic models are that they are less affected by the problem of "state explosion" (parallelism leads to a sum of the components states, not to their product), and the possibility of *local analysis*. The latter implies that it is relatively easy to study only that part of a system in which one is interested, isolating it from the rest. The specific semantic model of our formalism is an (orthogonal) extension of *bundle event structures* [8, 9], which is an adaptation of labeled event structures [15] to fit the specific requirements of parallel composition with multi-way synchronization, as is used in languages like LOTOS and CSP.

It should be noted that this paper reports on work in progress, and should not be regarded as our final position on linking functional specification to performance analysis in the context of process algebras. In the concluding section we will indicate more specifically what still needs to be done, in our opinion.

2 The Language

The language proposed in this paper is a timed probabilistic extension of (a subset of) basic LOTOS [2]. Constructs like parallel and sequential composition, hiding, and enabling are included without value passing. The language is extended with a probabilistic choice (like in [7]), a delay function, and urgent actions. We first briefly present the pure version of the language, i.e. without representations for time and probability.

2.1 Behaviours

Let $\mathcal{G}$ be a set of action labels and $\mathcal{P}$ a set of process names. Let $a \in \mathcal{G} \cup \{\,\mathbf{i}\,\}$, where $\mathbf{i}$ is a special label representing silent actions, $G \subseteq \mathcal{G}$, $P \in \mathcal{P}$, and $H : \mathcal{G} \to \mathcal{G}$ a relabeling function. The syntax of behaviours is defined as follows.

Definition 2.1 *Behaviours*

$$B ::= \mathbf{stop} \;\big|\; (a\,;\, B) \;\big|\; (B \,[]\, B) \;\big|\; (B \,|[G]|\, B) \;\big|\; \mathbf{hide}\ G\ \mathbf{in}\ B \;\big|\; B[H] \;\big|\; P \qquad \square$$

$|[\emptyset]|$ is also denoted by $|||$, and when G contains the set of all action labels in both arguments, $|[G]|$ is denoted by $||$. The precedences of the operators are, in increasing binding order: **hide** G **in** and $|[G]|$, $[]$, $;$, $[H]$. Parentheses are omitted if this does not introduce ambiguities.

The simplest behaviour is the behaviour that can perform no actions at all, denoted by **stop**. If a is an action and B a behaviour, $a\,;\,B$ denotes a behaviour which may engage in a after which it behaves like B. This operator is called action-prefix.

$B_1 \,[]\, B_2$ denotes the (standard) choice between behaviours B_1 and B_2. It should be noted that this choice is resolved in interaction with the environment, that is, by a behaviour that is composed in parallel with $B_1 \,[]\, B_2$.

Parallel composition of behaviours is denoted by $B_1 \,|[G]|\, B_2$, where G is the set of actions which have to be performed by both behaviours in co-operation. B_1 and B_2 can perform actions, that are not part of the (synchronization) set G, independent of each other.

Abstraction of a set of actions G in a behaviour B is supported by the hiding operator, denoted **hide** G **in** B. Behaviour **hide** G **in** B behaves analogous to B except that actions in the set G are turned into silent actions (denoted by $\mathbf{i}$) such that those actions are no longer visible to the environment of the behaviour.

$B[H]$ denotes a behaviour which is obtained by renaming the actions in B according to H.

P denotes a process instantiation; we assume a behaviour is always considered in the context of a set of process definitions of the form $P := B$ where B is a behaviour (possibly containing occurrences of P).

More elaborate treatments of similar process algebraic languages are presented in [1, 2, 6, 11].

2.2 Probabilistic Behaviours

In order to express probabilities our formalism is extended with a *probabilistic choice*, denoted $[]_p$, for $p \in [0,1]$, i.e. $0 \leq p \leq 1$. Under the assumption that the choice between B_1 and B_2 cannot be influenced by the environment, behaviour $B_1 \,[]_p\, B_2$ non-deterministically behaves like B_1 (with probability p) or like B_2 (with probability $1{-}p$). The precise conditions on B_1 and B_2 are described below. The syntactic category of behaviours constructed using the operators of the previous section, together with $[]_p$, is denoted by *Bex.*

$[]_p$ and $[]$ bind equally strong. Throughout this paper p, q, and r denote probabilities in $[0,1]$.

2.2.1 Probabilistic versus Standard Choice

In our language we distinguish between a standard and a probabilistic choice. We believe that this distinction is important—from a functional specification perspective it is necessary to express choices for which it is left unspecified what the probability of an alternative is. Such quantitative knowledge may either be absent or it may be deliberately left unspecified. Therefore, one should not be forced to associate such a quantity with an alternative. For these reasons we have decided to extend the language with a probabilistic choice rather than replacing the standard choice by a probabilistic one.

2.2.2 Internal Probabilistic Choice

The assumption that the probabilistic choice between B_1 and B_2 cannot be influenced by the environment is forced by syntactical constraints on B_1 and B_2. These constraints guarantee that $B_1 \, []_p \, B_2$ induces an *independent* stochastic experiment, that is, a set of alternatives where each alternative is assigned a certain probability which is independent of its context.

The use of probabilities in process algebras where $[]_p$ can be influenced by the environment leads to a considerable complicated semantics as stochastic experiments must be considered that are not independent. (For an overview of the different ways in which probabilistic choices can be resolved in interaction with the environment see [14].) Consequently, in these cases the probability of a behaviour depends on the context in which it is considered. In addition, there are many applications for which the use of probabilities in the above sense is not necessary since the phenomena one usually wants to describe by probabilistic behaviours (like faults, for instance) are typically out of the environment's control.

In the following we define the syntactical constraints on probabilistic behaviours. As a prerequisite we introduce an auxiliary predicate that determines whether an expression has a probabilistic choice at the 'component' level. This predicate is defined by structural induction over the expression.

Definition 2.2 *Predicate $\mathcal{P}$*

Predicate $\mathcal{P} : \mathit{Bex} \rightarrow \mathrm{Bool}$ is defined as follows:

$$\begin{array}{lcl}
\mathcal{P}(B_1 \,|[G]|\, B_2) & = & \mathcal{P}(B_1) \;\vee\; \mathcal{P}(B_2) \\
\mathcal{P}(\mathbf{hide}\ G\ \mathbf{in}\ B) & = & \mathcal{P}(B) \\
\mathcal{P}(B[H]) & = & \mathcal{P}(B) \\
\mathcal{P}(P) & = & \mathcal{P}(B) \text{ where } P := B \\
\mathcal{P}(B_1 \, []_p \, B_2) & = & \text{true}
\end{array}$$

The predicate is false for **stop**, action prefix and choice. □

Definition 2.3 *Language $\mathcal{L}$*

Let $\mathcal{Q}$ be defined by $\mathcal{Q}(B) = (\exists\, B', B'', q : B = B' \, []_q \, B'' \;\vee\; B = \mathbf{i}\,;\, B')$.
$\mathcal{L}$ is the largest subset of *Bex* such that for any subexpression B' of $B \in \mathit{Bex}$ which is either a standard or probabilistic choice the following holds

1. $B' = B_1 \, [] \, B_2 \;\Rightarrow\; \neg\mathcal{P}(B_1) \;\wedge\; \neg\mathcal{P}(B_2)$
2. $B' = B_1 \, []_p \, B_2 \;\Rightarrow\; \mathcal{Q}(B_1) \;\wedge\; \mathcal{Q}(B_2)$

□

The first condition states that in a standard choice both argument behaviours may not contain a probabilistic choice at the 'component' level. Thus, for instance[1], $a\,[]\,(\mathbf{i}\,;\,b\,[]_{0.4}\,\mathbf{i}\,;\,c) \notin \mathcal{L}$, since the probability of choosing, for example, alternative $\mathbf{i}\,;\,b$ cannot be determined. The second condition states that any argument of a probabilistic choice must either be a probabilistic choice or must start with an internal action $\mathbf{i}$.

Examples of expressions that belong to $\mathcal{L}$ are $(\mathbf{i}\,;\,a\,[]_{0.3}\,\mathbf{i}\,;\,b)\,|[\,b\,]|\,c\,;\,b,\ a\,[]\,b\,;\,(\mathbf{i}\,;\,a\,[]_{0.99}\,\mathbf{i}\,;\,c)$, and $\mathbf{i}\,;\,a\,[]_{0.3}\,(\mathbf{i}\,;\,b\,[]_{0.4}\,\mathbf{i}\,;\,c)$. Notice that probabilistic choices can be used in context of parallel compositions.

Probabilistic choices are restricted to be performed between behaviours the first actions of which are required to be unobservable actions. For instance, $a\,;\,B_1\,[]_p\,a\,;\,B_2$ and $a\,;\,B_1\,[]_p\,\mathbf{i}\,;\,B_2$ are not taken into consideration here, although their standard counterparts express instances of internal non-determinism. The reason for this choice is to keep our model as simple as possible. On the other hand, we also have the following equations, where $\approx_{te}$ denotes testing equivalence [12],

$$\begin{array}{rcl} a\,;\,B_1\,[]\,a\,;\,B_2 & \approx_{te} & \mathbf{i}\,;\,a\,;\,B_1\,[]\,\mathbf{i}\,;\,a\,;\,B_2 \\ a\,;\,B_1\,[]\,\mathbf{i}\,;\,B_2 & \approx_{te} & \mathbf{i}\,;\,((a\,;\,B_1)\,[]\,B_2)\,[]\,\mathbf{i}\,;\,B_2 \quad . \end{array}$$

Thus all forms of internal non-determinism can be rewritten in the required format of our formalism, preserving the notion of testing equivalence. As a consequence the proposed model is expressive enough as long as reasoning modulo testing equivalence is acceptable.

2.3 Timed Probabilistic Behaviours

In the sequel let time $t \in \mathcal{R}^+$ (that is, a positive real) and $\mu \in \{\,a, \widehat{a}\,\}$ with $a \in \mathcal{G} \cup \{\,\mathbf{i}\,\}$. The syntax of timed probabilistic behaviours is now defined as follows.

Definition 2.4 *Timed Probabilistic Behaviours*

$$B ::= \mathbf{stop} \;\big|\; ((t)\,\mu\,;\,B) \;\big|\; (B\,[]\,B) \;\big|\; (B\,|[G]|\,B) \;\big|\; \mathbf{hide}\ G\ \mathbf{in}\ B \;\big|\; B[H] \;\big|\; P \;\big|\; (B\,[]_p\,B) \ \Box$$

Actions are considered to be atomic and to occur instantaneously. That is, actions have no duration. The elementary timing construct of our language is a *delay function* construct that expresses the relative delay of an action. Behaviour $a\,;\,(t)\,b$ behaves identical to $a\,;\,b$, except that it is able to engage in b from t time-units after the occurrence of a. We abbreviate $(0)\,a$ by a. For initial actions the time is related to the beginning of the system at hand.

We use the positive real numbers for denoting delays. In this way a dense time domain is created. Notice that this is not essential for our model—an arbitrary time domain with an ordering operator would suffice.

By default, actions are treated to be *non-urgent*: actions are enabled t time-units after the occurrence of their causal predecessors (if any), but may occur at any time after that. In contrast, action $\widehat{a}$ denotes action a that is forced to occur as soon as it is enabled—in this case a is called an *urgent* or immediate action. An urgent action does not have priority over a non-urgent one; when actions of both types are possible at the same time, the choice between the actions is non-deterministic.

[1] Here, we have omitted trailing **stop** expressions. This convention is used in the sequel.

As an example of the use of urgent actions in combination with delays consider the following expression:

$$a\,;\,((t_1)\,B_1\,[]\,(t_2)\,\widehat{to}\,;\,B_2)\quad .$$

After the occurrence of a it specifies a non-deterministic choice between two behaviours, B_1 and $\widehat{to}\,;\,B_2$. The first behaviour is enabled t_1 time-units after a's occurrence, the second behaviour after t_2 time-units. When B_1 performs an action before the second argument is enabled (that is, t time-units after a's occurrence, with $t \in [t_1, t_2]$) the entire behaviour subsequently behaves like B_1. Otherwise, precisely t_2 time-units after the appearance of a it behaves like $\widehat{to}\,;\,B_2$. to represents a timeout action and can be seen as an annotated internal action.

2.3.1 Urgency

Actions are by default defined to be non-urgent as they can still be subject of synchronization with the environment. This synchronization may happen at any moment after the enabling of the action. It is our believe that the concept of urgent actions is important, for instance to express timeouts – which are forced to occur at a certain time, irrespective of the rest of the system – and to express strict performance requirements, e.g. that a response should occur within a certain time after issuing a request.

For the moment we assume that urgent actions model activities whose occurrence can be controlled completely internally (like timeouts). That is, urgent actions are supposed to appear typically without participation of the environment. Therefore, no synchronization on urgent actions is allowed. This keeps our semantical model as simple as possible.

2.3.2 Synchronization

A set of behaviours may synchronize on a common action as soon as all participants are ready to engage in this action. That is, when all individual timing constraints on this common action are met. One may thus consider that the enabling of a common action is constrained by the various individual timing requirements. For instance, for $a\,;\,(3)\,c$ and $b\,;\,(7)\,c$, action c is enabled in the composite behaviour

$$a\,;\,(3)\,c\,|[c]|\,b\,;\,(7)\,c\quad ,$$

if both a has occurred at least 3 time-units before and b has occurred at least 7 time-units before. In a similar way, in behaviour

$$a\,;\,(t_1)\,b\,|[a,b]|\,a\,;\,(t_2)\,b\quad ,$$

b is enabled after $max(t_1, t_2)$ time-units from the occurrence of a.

Using the timeout construct described earlier one can easily specify that a behaviour can perform an action only for a certain limited period of time. For instance, behaviour

$$a\,;\,((2)\,b\,[]\,(3)\,\widehat{to_1})\quad ,$$

is able to perform action b only in the period $[2, 3]$ after the occurrence of a[2]. When such a behaviour is placed in an environment that cannot allow an action to occur during this time interval, this action does not occur. (Note that this notion does not lead to a so-called *time-lock* in which the passage of time is blocked. As we do not have explicit time advancing actions in our model this phenomenon can not occur here.) For instance, if the aforementioned behaviour is placed in the following context

$$a\,;\,((2)\,b\,[\,]\,(3)\,\widehat{to_1})\,|[\,a, b\,]|\,a\,;\,((0)\,b\,[\,]\,(1.5)\,\widehat{to_2}) \quad ,$$

action b will never occur as the context is only able to perform b in the period $[0, 1.5]$ after the appearance of a. Instead, in both argument behaviours a timeout will appear.

2.3.3 Time delay and probabilistic choice

In order for probabilistic choice to model a stochastic experiment, we pose the restriction that all the initial internal actions in a probabilistic choice should have the same time delay. Without this restriction an expression like

$$(0)\,\mathbf{i}\,;\,a\,[\,]_p\,(1)\,\mathbf{i}\,;\,b$$

would be allowed. It would be difficult interpret this expression as a stochastic experiment, as before time 1 only the first internal action can happen, and not the second.

2.3.4 Finite Variability

A behaviour possesses the finite variability property if it can perform only finitely many actions in a finite amount of time. Such behaviours are also known as non-Zeno behaviours. We deal with Zeno behaviours in the same way as behaviours that contain a potential deadlock (or livelock)—it is allowed to construct specifications in which deadlocks or Zeno behaviours may occur, and it is sufficient to be able to verify that a specification has such possible undesired behaviour.

3 Timed Probabilistic Event Structures

3.1 Bundle Event Structures

As mentioned before the semantics of $\mathcal{L}$ is based on a true concurrency model and is defined by using an extension of *bundle event structures* [8, 9]. Bundle event structures consist of events labeled with actions (an event modeling the occurrence of its action), together with relations of causality and conflict between events. System runs can be modeled as partial orders of events satisfying certain constraints posed by the causality and conflict relations between the events.

Causality is represented by a relation between a set of events X (also called bundle set) and an event e. The interpretation is that if e happens in a system run, at least one of the events in X must have happened before. In addition we demand that all events in X are parewise

[2] Note that at precisely 3 time-units after the appearance of a a non-deterministic choice exists between b and to_1.

in conflict, so if e happens, exactly one event in X has happened before, which enables us to uniquely define a causal ordering between the events in a system run. *Conflict* is a symmetric binary relation between events and the intended meaning is that when two events are in conflict, they can never both happen in a single system run. When there is neither a conflict nor a causal relation between events they are said to be *independent*. This means that if independent events are enabled they can occur in any order or in parallel.

Definition 3.1 *Bundle Event Structure*

A *bundle event structure* $\mathcal{E}$ is a 4-tuple $(E, \#, \mapsto, l)$ with:

- E, a set of *events*
- $\# \subseteq E \times E$, the *conflict relation*
- $\mapsto \subseteq 2^E \times E$, the *causality relation*
- $l : E \to Act$, the *action-labeling* function, where *Act* is a set of action labels

such that the following properties hold:

1. $\forall X \subseteq E, e \in E : X \mapsto e \;\Rightarrow\; (\forall e1, e2 \in X : e1 \neq e2 \;\Rightarrow\; e1\#e2)$
2. $\#$ is *irreflexive* and *symmetric* .

□

Bundle event structures are graphically represented in the following way. Events are denoted as dots; near the dot the action label is given. Conflicts are indicated by dotted lines between representations of events. A bundle (X, e) is indicated by drawing an arrow from each element of X to e and connecting all arrows by small lines.

Example 3.2

Figure 1 is an example of a bundle event structure.

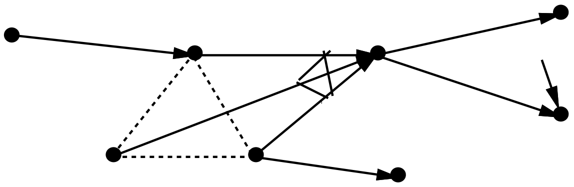

Figure 1: Example bundle event structure.

So we have, among others, bundles $\{e_a\} \mapsto e_b$, $\{e_b, e_c, e_d\} \mapsto e_x$ and $\emptyset \mapsto e_w$. □

The concept of a sequential observation of a system's behaviour is defined as follows.

Definition 3.3 *Event trace*

An *event trace* of $\mathcal{E}$ is a sequence of distinct events $e_1 \ldots e_n \in E$, satisfying:

1. $\forall e_i, e_j : \neg(e_i \# e_j)$, and
2. $\forall X \subseteq E : X \mapsto e_i \;\Rightarrow\; \{e_1, \ldots, e_{i-1}\} \cap X \neq \emptyset$

□

The first condition states that an event trace should be conflict-free, for obvious reasons. The last condition states that each event in the event trace has a causal predecessor occurring earlier in the sequence. Event traces correspond to linearizations of system runs.

The concept of the state of a system corresponds to the concept of *configuration*:

Definition 3.4 *Configuration*

A set $C \subseteq E$ is called a *configuration* if there is an event trace $e_1 \dots e_n$ such that $C = \{ e_1, \dots, e_n \}$. □

Example 3.5

The bundle event structure of example 3.2 has e.g. event traces *abxv*, *adxy* and *dyax*. The last two event traces both lead to the configuration $\{a, d, x, y\}$. Note there is no configuration containing event w. □

3.2 Probabilistic Event Structures

A *probabilistic* bundle event structure is a bundle event structure where some events are labeled with probabilities.

Events that are assigned a probability are grouped together in order to model a discrete probability space. That is, such events must be grouped into a *cluster* of mutually conflicting events. Each cluster represents a stochastic experiment. In order for clusters to represent stochastic experiments all events in a cluster must be caused by the same bundle sets. In this way, it is guaranteed that if an event in a cluster is enabled, all events in the cluster are enabled. The probability label associated to an event e, is the conditional probability of e to happen provided that e is enabled.

Notation: we denote the reflexive closure of $\#$ by $\natural$, so $e \natural e'$ means $e \# e'$ or $e = e'$.

Definition 3.6 *Cluster*

For $(E, \#, \mapsto, l)$, a *cluster* is a set Q of events, $Q \subseteq E$, satisfying $\forall e \in Q$:

1. $l(e) = \mathbf{i}$
2. $Q = \{ e' \mid e' \natural e \}$
3. $Q \setminus \{ e \} \neq \emptyset$
4. $\forall e' \in Q, X \subseteq E : X \mapsto e \Leftrightarrow X \mapsto e' \quad .$

□

The first condition states that all events in a cluster Q are labeled with an internal event $\mathbf{i}$. Condition 2 states that all distinct events in Q are in conflict with one another and are not in conflict with events not in Q. From condition 3 it follows that a cluster consists of at least two events; this condition is just convenient for technical reasons and poses no real practical constraint. The last condition states that all events in a cluster must be pointed at by identical bundle sets and therefore are either all enabled or all not enabled.

Notation: let for any function $f : A \rightarrow B$, $dom(f)$ be defined as $\{ a \in A \mid \exists b \in B : f(a) = b \}$.

Definition 3.7 *Probabilistic Bundle Event Structure*

A *probabilistic* bundle event structure is a tuple $\langle \mathcal{E}, \pi \rangle$ with:

- $\mathcal{E} = (E, \#, \mapsto, l)$ a bundle event structure
- $\pi : E \rightarrow [0, 1]$ a partial function, the *probability labeling* function,

such that the following properties hold for all $e \in dom(\pi)$ and $Q = \{\, e' \mid e' \natural e \,\}$:

1. Q is a cluster
2. $Q \subseteq dom(\pi)$
3. $\sum_{e' \in Q} \pi(e') = 1$

□

The first two conditions state that the domain of π completely consists of clusters. The last condition states that the sum of probabilities assigned to all events in cluster Q must be one. In this way, selecting one event out of a cluster can be interpreted as a stochastic experiment.

Example 3.8

Figure 2 shows three examples of probabilistic bundle event structures.

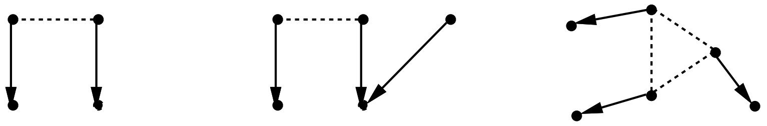

Figure 2: Example probabilistic bundle event structures.

They correspond to the probabilistic behaviour expressions $\mathbf{i}\,;\, a\,[]_{1/3}\,\mathbf{i}\,;\, b$ (figure a), $(\mathbf{i}\,;\, a\,[]_{1/3}\,\mathbf{i}\,;\, b)\,|[\,b\,]|\,c\,;\, b$ (figure b), and $\mathbf{i}\,;\, a\,[]_{1/2}\,(\mathbf{i}\,;\, b\,[]_{1/2}\,\mathbf{i}\,;\, c)$ (figure c). □

The definitions of event trace and configuration remain valid for probabilistic event structures. In [7] it is shown how probabilities can be attached to classes of configurations where for each class probabilistic choices in the configurations are resolved in the same way.

3.3 Timed Event Structures

In this section we add time delays to the bundle event structure model, in two ways. First we want to associate time with bundles: suppose we have an event c with a bundle $\{\, a, b \,\} \mapsto c$, we want to associate a time delay t with this bundle. The interpretation is that if a or b has happened at a certain time, then c is enabled t time-units later. It need not happen immediately, so it may happen at any time that is at least t units later than the time at which a or b has happened. The motivation for this non-urgency is that in general an event may be subject to interaction (e.g. with the environment) which may introduce further delays. So we assume a mapping $\mathcal{T}$ that associates a non-negative real number with each bundle. We use the following notational convention: a bundle (X, e) with $\mathcal{T}((X, e)) = t$ is denoted by $X \overset{t}{\mapsto} e$.

Events can have several bundles pointing to them. Suppose we have event c with bundles $\{\, a \,\} \overset{t1}{\mapsto} c$ and $\{\, b \,\} \overset{t2}{\mapsto} c$. The interpretation is that if a happens at time t_a and b happens at time t_b, then c can happen after time $max\{t_a + t1, t_b + t2\}$.

In addition we want model time delays for events that do not have a bundle pointing to them. Therefore we assume a mapping $\mathcal{D}$ that associates with each event a non-negative real number; the interpretation is that an event e with $\mathcal{D}(e) = t$ can only happen after t time-units.

Finally we want to be able to model urgent events that happen immediately when they are enabled. This is important for example for modeling timers, since timers expire at a precise time. Urgent events are graphically indicated as open dots, non-urgent events as closed dots. The time delay associated with a bundle is depicted next to the bundle; the time delay associated with an event is depicted next to the event, enclosed between brackets (to avoid confusion with probabilities). We adopt the convention that zero time delays are not indicated.

Example 3.9

The timed event structure in Figure 3 gives an example of a timer. The *send* event may

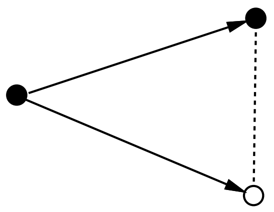

Figure 3: Timer example.

happen after 1 time-unit. The *receive* event may happen between 3 and 5 time-units after *send*; if not, *timeout* happens at exactly 5 time-units after *send*. □

Urgency is modeled by a predicate $\mathcal{U}$ on events: if e is an urgent event then $\mathcal{U}(e)$ is true. We pose the constraint that an urgent event has at most one bundle pointing to it. This constraint is technically convenient as it considerably simplifies several definitions below, and it poses no real practical limitations in the semantics for our timed process algebra as urgent actions are not subject to synchronization. For similar reasons we pose the constraint that for an urgent event e with a bundle pointing to it, $\mathcal{D}(e) = 0$.

Definition 3.10 *Timed (Probabilistic) Bundle Event Structure*

A *timed* (probabilistic) bundle event structure is a tuple $\langle \mathcal{E}, (\mathcal{T}, \mathcal{D}, \mathcal{U}) \rangle$ with:

- $\mathcal{E}$ is a (probabilistic) bundle event structure $(E, \#, \mapsto, l)$ (respectively $\langle (E, \#, \mapsto, l), \pi \rangle$)
- $\mathcal{T} : \mapsto \rightarrow \mathcal{R}^+$, the *timing* function
- $\mathcal{D} : E \rightarrow \mathcal{R}^+$, the *delay* function
- $\mathcal{U} : E \rightarrow$ Bool, the *urgency* predicate

such that the following property holds for all $e \in E$ and $X, Y \subseteq E$:

$$(\mathcal{U}(e) \;\wedge\; X \mapsto e \;\wedge\; Y \mapsto e) \;\Rightarrow\; X = Y \;\wedge\; \mathcal{D}(e) = 0 \quad .$$

□

As a generalization of the notion of event trace we define the notion of timed event trace.

Definition 3.11 *Timed Event Trace (I)*

A *timed event trace* of $\mathcal{E}$ is a sequence of distinct timed events $(e_1, t_1) \ldots (e_n, t_n)$ with $e_i \in E$, $t_i \in \mathcal{R}^+$ $(1 \le i \le n)$, satisfying:

1. $i < j \;\Rightarrow\; t_i \le t_j$
2. $\forall\, e_i, e_j : \neg\,(e_i \# e_j)$
3. $\forall\, X \subseteq E : X \overset{t}{\mapsto} e_i \;\Rightarrow\; \exists\, e_j : X \cap \{\, e_1, \ldots, e_{i-1} \,\} = \{\, e_j \,\} \;\wedge$
   (a) $\quad \mathcal{U}(e_i) \;\Rightarrow\; t_i = t_j + t$
   (b) $\neg\mathcal{U}(e_i) \;\Rightarrow\; (t_i \ge t_j + t \;\wedge\; t_i \ge \mathcal{D}(e_i))$
4. $\neg\,(\exists\, X \subseteq E : X \overset{t}{\mapsto} e_i) \;\Rightarrow$
   (a) $\quad \mathcal{U}(e_i) \;\Rightarrow\; t_i = \mathcal{D}(e_i)$
   (b) $\neg\mathcal{U}(e_i) \;\Rightarrow\; t_i \ge \mathcal{D}(e_i)$

□

The above definition adds several conditions to the definition of event trace 3.3 in order to take care of the correct timing of events. Clauses 3(a) and 3(b) constrain the timing of events that do have a bundle pointing to them. Note that for urgent events we do not have to take the delay into consideration, as according to definition 3.10 urgent events with a bundle pointing to them have delay 0. Clauses 4(a) and 4(b) take care of the timing of events without bundles.

Definition 3.11 ensures that the events in a timed event trace have correct times associated with them, but it does not ensure that urgent events appear in the timed event trace at places where they should appear. This can be illustrated by looking at Figure 3. According to definition 3.11, $(\mathit{send}, 1)(\mathit{receive}, 7)$ is a timed event trace. However, if the *receive* event has not happened before time 6, then the *timeout* event should have happened at time 6; therefore $(\mathit{send}, 1)(\mathit{receive}, 7)$ should not be considered a legal timed event trace. This is expressed by the next two additional constraints:

Definition 3.12 *Timed Event Trace (II)*

A *timed event trace* of $\mathcal{E}$ is a sequence of distinct timed events $(e_1, t_1) \ldots (e_n, t_n)$ with $e_i \in E$, $t_i \in \mathcal{R}^+$ $(1 \le i \le n)$, satisfying in addition to 1-4 of definition 3.11:

5. $X \overset{t}{\mapsto} e \;\wedge\; \exists\, e_j : X \cap \{\, e_1, \ldots, e_n \,\} = \{\, e_j \,\} \;\wedge\; \mathcal{U}(e) \;\wedge\; (\exists\, e_k : t_k > t_j + t)$
   $\Rightarrow\; (\exists\, e_m : e_m \natural e \;\wedge\; t_m \le t_j + t)$
6. $\neg\,(\exists\, X \subseteq E : X \overset{t}{\mapsto} e) \;\wedge\; \mathcal{U}(e) \;\wedge\; (\exists\, e_k : t_k > \mathcal{D}(e))$
   $\Rightarrow\; (\exists\, e_m : e_m \natural e \;\wedge\; t_m \le \mathcal{D}(e))$

□

Constraint 5 says the following: if there is an urgent event with a bundle with time delay t pointing to it, and this event is enabled at some time t_j in the event trace, and there is an event in the event trace occurring later than $t_j + t$ (implying that the event trace spans time $t_j + t$) then before $t_j + t$ either the urgent event, or some event in conflict with it, should have happened.

Constraint 6 says the following: if there is an urgent event with no bundle pointing to it (so it is enabled at time 0), with an associated delay of t, and there is an event in the event trace occurring later than t (implying that the event trace spans time t) then before t either the urgent event, or some event in conflict with it, should have happened.

Example 3.13

Consider the timed probabilistic bundle event structure depicted in Figure 4. For event e let t_e denote the time at which e may occur. (x, t_x) belongs to a timed event trace if and only if

$$t_d + k \leq t_x \leq t_d + n \ \wedge \ t_e + l \leq t_x \leq t_e + m \quad .$$

Consider the first constraint. As we have a bundle $\{\, d \,\} \overset{k}{\mapsto} x$ it immediately follows that $t_d + k \leq t_x$. In addition, bundle $\{\, d \,\} \overset{n}{\mapsto} a$ exists and $\mathcal{U}(a)$. From $x \natural a$ we infer $t_x \leq t_d + n$. By symmetry, an analogous reasoning leads to the other constraint. □

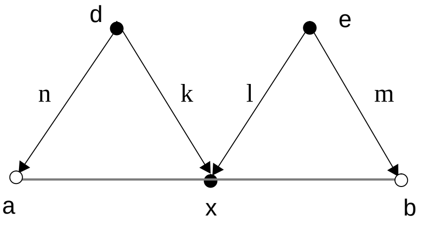

Figure 4: An example timed bundle event structure.

4 A Semantics for Timed Probabilistic LOTOS

The semantics of $\mathcal{L}$ can be given in terms of timed probabilistic bundle event structures. A mapping $[\![\ ]\!]$ is defined that maps each expression B of $\mathcal{L}$ to a timed probabilistic bundle event structure $[\![\, B \,]\!]$. This function is an orthogonal extension of the semantics presented in [8, 9] and is completely described in [3]. An interesting feature of this mapping is that removing the parts concerning the timing of events and bundles and the probability part in the definition of $[\![\ ]\!]$ leads to the bundle event semantics of ordinary LOTOS given in [8]. In this extended abstract we do not treat all details of the mapping, but rather present it by example.

In Figure 5 the corresponding timed event structures of the following expressions are depicted:

(a) $((2)\ a\,;\ (3)\ d \,[\,]\, (1)\ b\,;\ (2)\ e) \,|||\, (27)\ c$,

(b) $(2)\ a\,;\ (4)\ b \,|[\,b\,]|\, (7)\ b$, and

(c) $(2)\ a\,;\ (5)\ c \,|[\,c\,]|\, (7)\ b\,;\ (1)\ c$.

Case (b) illustrates that by parallel composition even events that have a bundle pointing to them can have a non-zero delay. As a second example the semantics of $B_1 \,|[\, a, b \,]|\, B_2$ is given for $B_1 = (1)\ a\,;\ (5)\ b \,|[\,b\,]|\, c\,;\ (3)\ b$ and $B_2 = (4)\ a\,;\ (2)\ b \,|[\,b\,]|\, (b \,[\,]\, (3)\ d)$. In addition, the corresponding timed bundle event structure of

$$((2)\ a\,;\ (7)\ x \,[\,]\, (4)\ a\,;\ (11)\ y) \,|[\,a\,]|\, ((5)\ a\,;\ (2)\ b) \quad .$$

is determined. Figure 6 illustrates the semantics of these expressions.

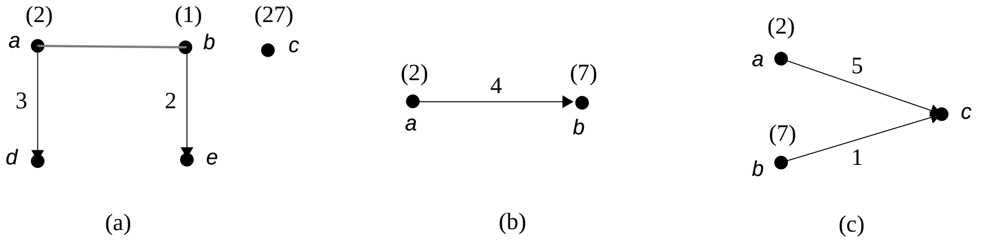

Figure 5: Some example semantics (I).

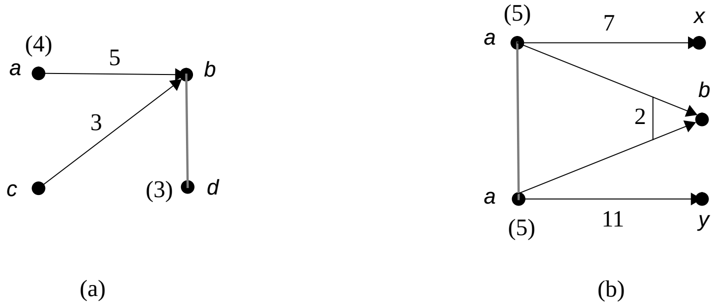

Figure 6: Some example semantics (II).

5 Performance Analysis

In this section we provide a simple example to illustrate how unreliable time-dependent systems can be specified using our formalism, and, more importantly, how the true concurrency semantics can be used as a starting-point for providing a model for performance analysis. The example is rather intuitive in the sense that no formal mapping between the event structures and the analysis model, that is, semi-Markov chains, is given. It is assumed that the reader is familiar with the notion of semi-Markov chains (see e.g. [4, 13]).

Although the mapping of regular infinite bundle event structures onto finite representations is currently being investigated a simple form of recursion is used in the example to describe the iterative behaviour of processes. The mapping of infinite structures generated by this type of recursion, tail recursion, onto finite representations is straightforward, as will be shown below.

As an example of using our formalism for constructing a performance model we consider a coffee-machine C and a user U. U represents an impatient user—after inserting a coin he wants to have coffee at its disposal within n time-units, $n \in \mathcal{R}^+$. For simplicity it is assumed that consuming coffee takes no time. Formally,

$$U := \textbf{hide}\ \mathit{to}\ \textbf{in}\ \mathit{coin}\,;\ (\mathit{coffee}\,;\ U\ [\,]\ (n)\ \widehat{\mathit{to}}\,;\ U) \quad .$$

The coffee-machine is quite realistic in the sense that it sometimes refuses to offer any coffee even after a coin has been inserted. Let p be the probability the machine behaves in this unreliable way. Furthermore, producing coffee is assumed to take k time-units ($k \in \mathcal{R}^+$). Formally,

$$C := \mathit{coin}\,;\ (\mathbf{i}\,;\ C\ [\,]_p\ \mathbf{i}\,;\ (k)\ \mathit{coffee}\,;\ C) \quad .$$

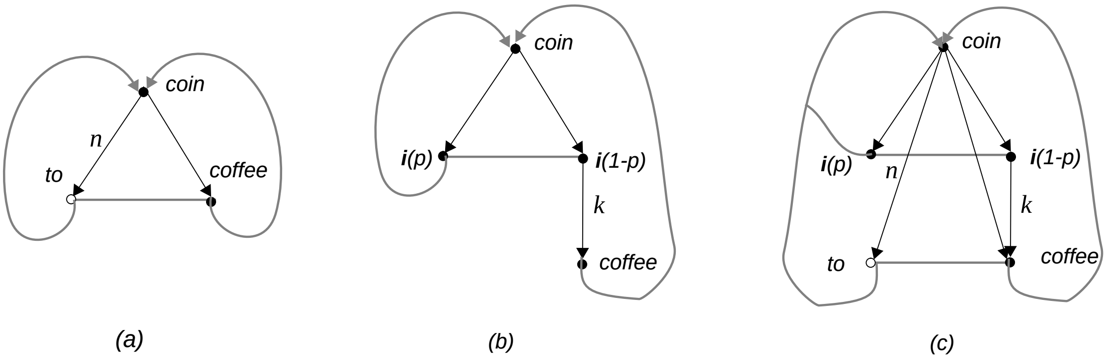

Figure 7: $[\![\,U\,]\!]$, $[\![\,C\,]\!]$, and $[\![\,S\,]\!]$.

The overall 'system' is specified by

$$S := U\,|[\,coin, coffee\,]|\,C \quad .$$

In order to make synchronizations on *coffee* possible we assume in the sequel that $n > k$. The corresponding timed probabilistic event structures (without considering hiding) are depicted in Figure 7. In fact, these figures only explicitly depict the finite part of the event structure corresponding to the "body" of the processes. Recursive calls should be considered as unfoldings of the depicted event structures. In this way we obtain a finite representation of an infinite event structure; this finite representation is possible in those cases were the infinite event structure possesses a certain regular structure. A first attempt of exploiting this regularity can be found in [10].

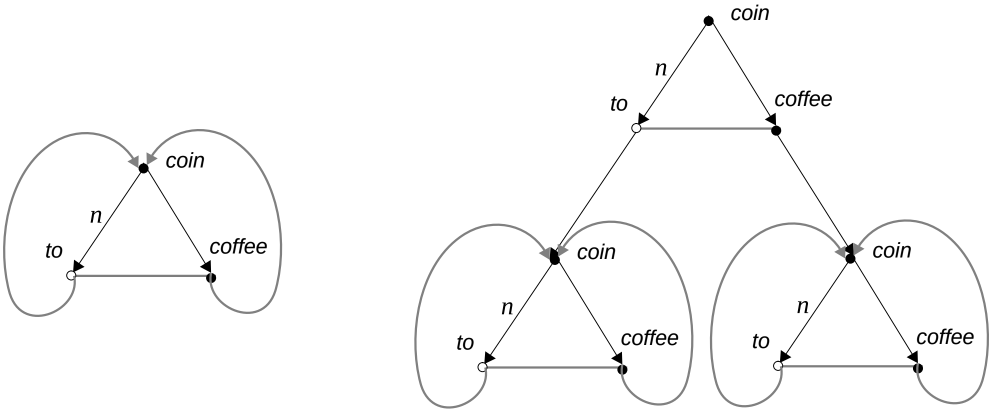

Figure 8: Unfoldings of $[\![\,U\,]\!]$.

Figure 8 illustrates for $[\![\,U\,]\!]$ how the unfolding is performed. Each successive unfolding is obtained by instantiating the original structure. The sequence of structures obtained by unfolding is very similar to the fixpoint semantics of recursive processes as defined in [8].

Assume now that we want to calculate the average number of cups of coffee, N_c, offered per time-unit. In order to determine this quantity the following grouping of events is introduced:

$$s_1 = \{\, \mathit{coin}, \mathbf{i}(p), \mathit{to}\,\}, s_2 = \{\, \mathbf{i}(1{-}p), \mathit{coffee}\,\} \quad .$$

s_1 represents the case in which no coffee is offered, s_2 represents the case in which actually coffee is offered, i.e. the successful case.

The way in which events are grouped imposes a particular "view" on the system which is characterized by abstracting from details that are irrelevant for the performance analysis one performs. For instance, for our purpose, it is not necessary to keep events $\mathbf{i}(p)$ and *to* separated as they both lead to the same situation, not offering any coffee. The groups of events and probabilistic transitions between them can be considered as a semi-Markov chain (see Figure 9).

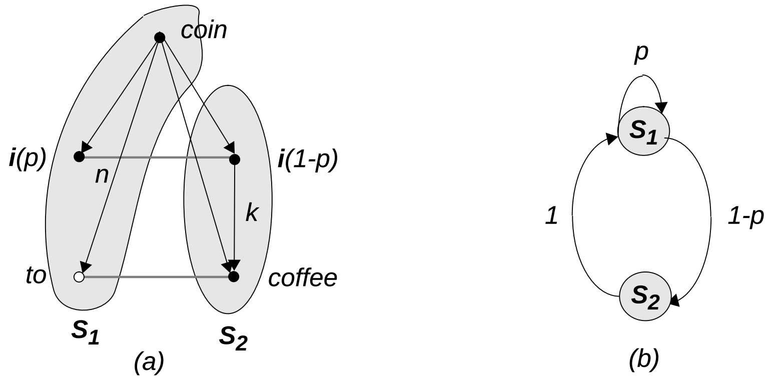

Figure 9: Grouping of events (a) and semi-Markov chain (b).

Assuming that an event takes place as soon as it is enabled, we obtain the following average residence times for the identified states of the semi-Markov chain: $r_1 = np + 0(1 - p)$ and $r_2 = k$. Using standard means we obtain for the stationary probabilities of the embedded Markov chain the following results:

$$\psi_1 = \frac{1}{2 - p} \quad , \psi_2 = \frac{1 - p}{2 - p} \quad .$$

Subsequently, these results can be 'translated' using the definition of r_i to stationary probabilities of the semi-Markov chain, thus obtaining

$$\phi_1 = \frac{np}{np + k(1 - p)} \quad , \phi_2 = \frac{k(1 - p)}{np + k(1 - p)} \quad .$$

(Notice that for k equal to n one obtains $\phi_1 = p$ and $\phi_2 = 1{-}p$.) The average number of time-units between successive transitions to s_i, T_i, is equal to r_i/ϕ_i. As S_2 represents the successful case, for N_c we obtain

$$N_c = \frac{1 - p}{np + k(1 - p)} \quad .$$

6 Concluding Remarks

In this paper we have presented a timed, probabilistic extension of a process algebraic formalism based on LOTOS, and studied its usefulness for obtaining semi-Markov chains from the semantic models defined by specifications. To our knowledge this constitutes the first attempt to deal with time and probabilities in the context of true concurrency. One of the features of the model is the absence of actions that represent the passage of time, which in one way or another make their appearance in all interleaving models. Here, time is dealt with in way comparable to ordinary physical models, viz. by means of parameterization (e.g. for recording the delays). A comparison between the presented approach and some others is given in the full paper [3].

The model presented is an extension of the bundle event structure semantics for LOTOS reported in [8]. There, for Basic LOTOS it is shown that this semantic model is compatible with the standard operational semantics for LOTOS, thus providing evidence for the adequacy of the model. A similar result for the extended model must still be worked out. A problem here is that the standard interleaving semantics do not have a canonical extension to include time and/or probabilities. Here, one is confronted with alternative approaches and technicalities that make the exercise less straightforward than in the unextended case.

The semantics presented here can be seen as a truly concurrent operational model for system behaviour involving time and probability. A point for further study is to see how to obtain from this more abstract semantics in the form of equivalences (congruences) and pre-orders (pre-congruences) that would reflect natural notions of transformation and implementation for timed and/or stochastic systems well. The existence of useful compatible semantics would provide further evidence for the adequacy of our approach.

Another direction to extend the work would be the further enhancement of expressive power. A direction that seems interesting from an application point of view, would be to work with probability density functions, as can be found in [5]. This would cater for more dynamic stochastic behaviour.

We have illustrated the use of our semantic model to obtain a performance model in the form of a semi-Markov chain in a simple example. There, the explicit presence of parallelism in the semantics helps in obtaining the performance model. It should be noted, however, that this connection is most readily exploited in the form of graphs (as used in the example), whereas the semantics of infinite behaviours is in reality given by infinite event structures. Under a regularity assumption, which applies in the case of the example, such infinite structures can be finitely represented by graphs, which are subsequently transformed into performance models. It would be most interesting and useful, however, to represent infinite behaviour directly in terms of such a graph-based semantics. A first attempt in this direction can be found in [10]. Although the structure of a performance model ultimately depends on the magnitudes one is interested in, such graph models could be a basis to study generic transformations to obtain Markov-like performance models from them, and guidelines and heuristics for applying them. Certainly, application of our method should first be attempted on larger, more realistic examples (e.g. broadband networks, multi-media), to develop a better feeling for what is really required.

Acknowledgment The work presented in this paper has been partially funded by C.N.R. - Progetto Bilaterale: Estensioni probabilistiche e temporali dell'algebra di processi LOTOS basate su strutture di eventi, per la specifica e analisi quantitative di sistemi distribuiti.

References

[1] J.C.M. Baeten and W.P. Weijland. *Process Algebra*. Cambridge University Press, 1990.

[2] T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks and ISDN Systems*, 14:25–59, 1987.

[3] E. Brinksma, J.-P. Katoen, R. Langerak, and D. Latella. Performance analysis and true concurrency semantics. In T. Rus and C. Rattray, editors, *Theories and Experiences on Real-Time System Development*, AMAST Series in Computing, chapter 12, pages 309–337. World Scientific, 1994 (to appear).

[4] D.P. Heyman and M.J. Sobel. *Stochastic Models in Operations Research*, volume 1 - Stochastic Processes and Operating Characteristics. McGraw-Hill, New York, 1982.

[5] J. Hillston. PEPA: Performance enhanced process algebra. Technical Report CSR-24-93, University of Edinburgh, 1993.

[6] C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.

[7] J.-P. Katoen, R. Langerak, and D. Latella. Modelling systems by probabilistic process algebra—an event structures approach. In R.L. Tenney, P.D. Amer, and M.Ü. Uyar, editors, *Formal Description Techniques, VI*, pages 253–269. North-Holland, 1994.

[8] R. Langerak. *Transformations and Semantics for LOTOS*. PhD thesis, University of Twente, 1992.

[9] R. Langerak. Bundle event structures: a non-interleaving semantics for LOTOS. In M. Diaz and R. Groz, editors, *Formal Description Techniques V*, pages 331–346. North-Holland, 1993.

[10] D. Latella. Recursive bundle event structures. Technical Report 93-27, University of Twente, 1993.

[11] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[12] R. De Nicola. Extensional equivalences for transition systems. *Acta Informatica*, 24:211–237, 1987.

[13] S.M. Ross. *Stochastic Processes*. John Wiley & Sons, New York, 1983.

[14] R. van Glabbeek, S.A. Smolka, B. Steffen, and C.M.N. Tofts. Reactive, generative, and stratified models of probabilistic processes. In *Proceedings of 5th IEEE Symposium on Logic in Computer Science*, pages 130–141, 1990.

[15] G. Winskel. An introduction to event structures. In J.W. de Bakker (et. al.), editor, *Linear Time, Branching Time and Partial Order in Logics and Models for Concurrency*, LNCS 354, pages 364–397. Springer-Verlag, 1989.