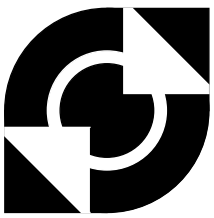
TIOS

Tele-Informatics
&
Open Systems

Performance Analysis and True Concurrency Semantics

Ed Brinksma, Joost-Pieter Katoen,
Rom Langerak & Diego Latella

Memorandum Informatica 94-39
Memorandum TIOS 94-10



University of Twente
Department of Computer Science
& Department of Electrical Engineering
Tele-Informatics and Open Systems Group
P.O.Box 217, 7500 AE Enschede, The Netherlands

Performance Analysis and True Concurrency Semantics*

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

June 29, 1994

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

*This paper will be published as part of the book 'Theories and Experiences on Real-Time System Development – Papers presented at the First AMAST Workshop on Real-Time Systems', edited by T. Rus and C. Rattray.

system functionalities that refer to time and/or probability. Another reason is provided by the necessity of analysing 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 queueing 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 a subset of LOTOS [4], a standardized formal description technique that is closely related to process algebraic languages as CCS [21] and CSP [12]. 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* [15, 16], which is an adaptation of labelled event structures [30] 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 [4]. Constructs like parallel and sequential composition, hiding, and enabling are included without value passing. The language is extended with a probabilistic choice (like in [14]), 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*

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

□

$|[\,\emptyset\,]|$ 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: $\mathbf{hide}\ G\ \mathbf{in}$ 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 $\mathbf{hide}\ G\ \mathbf{in}\ B$. Behaviour $\mathbf{hide}\ G\ \mathbf{in}\ B$ 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 [3, 4, 12, 21].

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. $0 < p < 1$. For technical reasons, p=0, 1 is not considered. Note that alternative specifications for these cases are, however, possible. 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 [29].) 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 $\mathcal{P} : Bex \rightarrow$ Bool is defined as follows:

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

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

□

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

Define predicate $\mathcal{Q}$ on behaviours 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 $B \in Bex$ 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. $B' = B_1 \,[]_p\, B_2 \ \Rightarrow \ \mathcal{Q}(B_1) \ \wedge \ \mathcal{Q}(B_2)$

□

Predicate $\mathcal{Q}$ is true if and only if a behaviour starts with $\mathbf{i}$ or is a probabilistic choice. 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

$$\begin{array}{l} (\mathbf{i}\,;\, a \,[]_{0.3}\, \mathbf{i}\,;\, b) \,|[\, b \,]|\, c\,;\, b \\ a \,[]\, b\,;\, (\mathbf{i}\,;\, a \,[]_{0.99}\, \mathbf{i}\,;\, c) \\ \mathbf{i}\,;\, a \,[]_{0.3}\, (\mathbf{i}\,;\, b \,[]_{0.4}\, \mathbf{i}\,;\, c) \end{array}$$

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 [23],

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

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

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*

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

□

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.

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 synchronisation with the environment. This synchronisation 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 synchronisation on urgent actions is allowed. This keeps our semantical model as simple as possible.

2.3.2 Synchronisation

A set of behaviours may synchronise 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]Note that at precisely 3 time-units after the appearance of a a non-deterministic choice exists between b and to.

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.

2.4 Relation to Other Work

In this section we briefly treat the relation between our language and some existing timed and probabilistic process algebras. We do not intend to give a comparison to the entire spectrum of such algebras, but restrict ourselves to a limited subset.

2.4.1 Probabilistic Extensions of Process Algebras

Probabilistic extensions of process algebras have been studied quite extensively in literature. Extensions of CSP [18], CCS [6, 9], SCCS [8, 29], ACP [2], and LOTOS [20, 26] have been studied. A classification of probabilistic process algebras into reactive, generative, and stratified models is given in [29].

In contrast to our approach several probabilistic process algebras replace the standard choice by a probabilistic one, like $[]_{0.5}$, and allow the use of probabilities in contexts in which probabilistic choices are performed in co-operation with the environment. In probabilistic ACP [2] a probabilistic parallel composition operator is introduced. Note that in an interleaving approach by the following expansion law

$$a \,|||\, b = a\,;\, b \,[]\, b\,;\, a \quad ,$$

parallel composition can be reduced to choice. When $[]$ is replaced by, for example $[]_{0.5}$, parallel composition implicitly becomes probabilistic. In our opinion this does not facilitate probabilistic reasoning as probabilistic information is more easily associated with non-determinism than with parallelism.

The treatment of probabilistic choices in the approaches presented in [9, 18, 20] is quite similar to ours. In the model of [9] —which is based on timed probabilistic transition systems— a

distinction is made between probabilistic and ordinary transitions, such that these types of transitions strictly alternate. Ordinary transitions may involve the participation of the environment, whereas probabilistic transitions do not [18] describes a probabilistic variant of CSP incorporating both types of choices, and allows only internal probabilistic choices as "we do not believe that a probabilistic external choice is particularly useful in its own right". The probabilistic version of LOTOS in [20] incorporates an internal n-ary probabilistic choice.

All probabilistic process algebras that are known to us are based on an interleaving semantics defined by some kind of probabilistic labeled transition system (or the like). The true concurrency semantics we propose is based on event structures. This allows to distinguish between non-determinism and parallel composition, reduces the state-space explosion problem, and more importantly, facilitates the analysis of only parts of a system (locality). The probabilistic extension of the semantical model is an orthogonal extension of the original semantical model.

2.4.2 Timed Process Algebras

Like probabilistic extension of process algebras, timed extensions have been (and still are) an active area of research. Timed extensions of CSP [28], CCS [9], ACP [1, 24], and LOTOS [5] have been investigated. An overview of timed process algebras is presented in [25]. We do not intend to compare our timed model to all the others in exhaustive detail. Instead, we focus the discussion on some specific aspects.

Different timed operators have been introduced by various researchers. As our main goal was to investigate the suitability of our semantical model with respect to timing issues, and the relation of this model to existing performance models, we deliberately did not decorate our language with operators like timed interrupts (watchdog operators) and the like. Using the elementary concepts of urgency, delay functions, and standard choice, timeout mechanisms can be expressed in a rather straightforward way (see introduction of syntax). These timeout mechanisms are so-called 'weak' timeouts [25], since at the moment of enabling of the timeout action a non-deterministic choice is possible between the timeout and an alternative.

Urgent and non-urgent actions are incorporated within a timed LOTOS variant in [5] where —identical to our proposal— all actions are non-urgent by default, but can be made urgent. Opposed to [5] we do not allow synchronisation on urgent actions. In other approaches [9, 28] internal actions are considered to be urgent as their occurrence cannot be influenced by the environment and so if a process is willing to perform such actions it should do so immediately (maximal progress or minimal delay). For timed variants of CCS with maximal progress it must be noticed that communications become internal (tau-)actions and, consequently, occur as soon as they are possible. In line with [5] we believe that urgency is a general concept of importance. On the one hand, the maximal progress assumption alone is not expressive enough (also non-internal actions may be urgent), and on the other hand, it is a bit restrictive—when specifying an unreliable communication service that may lose messages, the loss of a message is usually modeled by an internal action, but we don't know when this message is physically lost!

A major characteristic of nearly all timed extensions of process algebra is the incorporation of a specific event in the semantical model, usually denoted by 'tick' or χ, that explicitly models the passage of time (for instance, by a single unit of time in a discrete model). As

time progresses for all processes at the same speed, all processes are forced to synchronize on time-passing actions. Thus, a process that can not perform any actions must always be able to perform χ actions, in order not to block the progress of time, and consequently, the actual progress in other processes of the system. The phenomenon that a process blocks the passage of time in other processes is known as time-lock. As already noticed by [7] this approach seems rather unnatural and counterintuitive as in physics time just passes without explicitly forcing clocks to advance. In the semantical model that we propose no explicit time passing events are incorporated.

Most timed process algebras are based on an interleaving semantics defined in terms of a kind of timed transition systems. The semantics of these algebras tend to be rather complicated by the explicit forcing of passage of time, and the fact that some interleavings may contradict the temporal ordering and have to be eliminated. We believe that true concurrency models, like event structures, are more natural to include a notion of real-time, by the explicit presence —or absence— of a causal relation between actions. A few extensions of true concurrency models are known to us that incorporate time (e.g. [7, 19, 22]). Fidge [7] describes a real-time extension of CCS based on causal trees. Maggiolo-Schettini & Winkowski [19] consider a theoretical model, called timed configurations, where all timed events are treated to be urgent. Murphy [22] treats a timed extension of event structures in which events have a duration, rather than having relative delays between events. To our knowledge, the approach we present in this paper is the first real-time true concurrency approach suited for LOTOS that incorporates both urgent and non-urgent actions (although synchronisations on urgent actions are not allowed). The timed extension of the semantical model is an orthogonal extension of the non-timed semantical model.

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* [15, 16]. Bundle event structures consist of events labelled with actions (an event modelling the occurrence of its action), together with relations of causality and conflict between events. System runs can be modelled 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 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.

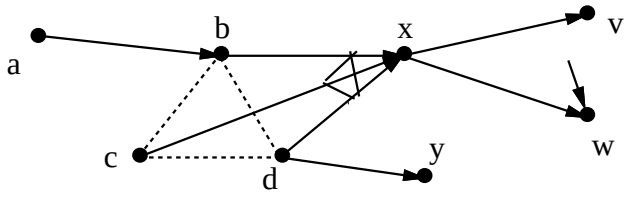


Figure 1: Example bundle event structure.

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 the behaviour of a system is provided by the next definition.

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 $e_1 \ldots e_n$ such that $C = \{ e_1, \ldots, e_n \}$.

□

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. $\forall e' \in Q, X \subseteq E : X \mapsto e \Leftrightarrow X \mapsto e'$.

□

The first condition states that all events in a cluster Q are labeled with an internal event 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 $f : A \to B$, $dom(f)$ 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 $< \mathcal{E}, \pi >$ with:

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

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

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.

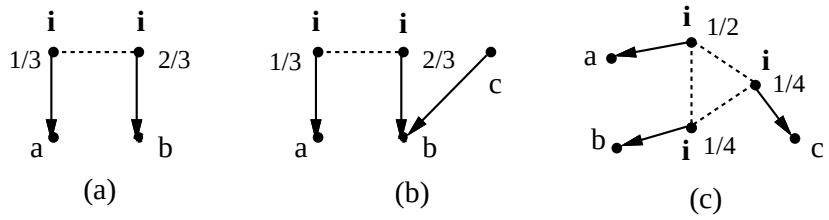


Figure 2: Example probabilistic bundle event structures.

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 3.3 and configuration 3.4 remain valid for probabilistic event structures. In [13] 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 $X \stackrel{t}{\mapsto} e$.

Events can have several bundles pointing to them. Suppose we have event c with bundles $\{ a \} \stackrel{t1}{\mapsto} c$ and $\{ b \} \stackrel{t2}{\mapsto} c$. 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 modelling 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

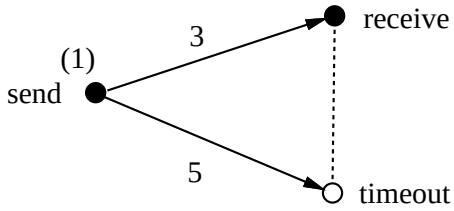


Figure 3: Timer example.

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 modelled 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 $< \mathcal{E}, (\mathcal{T}, \mathcal{D}, \mathcal{U}) >$ with:

- $\mathcal{E}$ is a (probabilistic) bundle event structure $(E, \#, \mapsto, l)$ (respectively $< (E, \#, \mapsto, l), \pi >$)
- $\mathcal{T} : \mapsto \rightarrow \mathcal{R}^+$, 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 $X, Y \subseteq E$:

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

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 $e_i \in E$, $t_i \in \mathcal{R}^+$ $(1 \leq i \leq n)$, satisfying:

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

□

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, $(send, 1)(receive, 7)$ 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 $(send, 1)(receive, 7)$ 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 $e_i \in E$, $t_i \in \mathcal{R}^+$ $(1 \leq i \leq n)$, satisfying in addition to 1-4 of definition 3.11:

5. $X \stackrel{t}{\mapsto} e \ \wedge \ \exists e_j : X \cap \{ e_1, \ldots, e_n \} = \{ e_j \} \ \wedge \ \mathcal{U}(e) \ \wedge \ (\exists e_k : t_k > t_j + t)$
$\Rightarrow \ (\exists e_m : e_m \natural e \ \wedge \ t_m \leq t_j + t)$

6. $\neg (\exists X \subseteq E : X \stackrel{t}{\mapsto} e) \ \wedge \ \mathcal{U}(e) \ \wedge \ (\exists e_k : t_k > \mathcal{D}(e))$
$\Rightarrow \ (\exists e_m : e_m \natural e \ \wedge \ t_m \leq \mathcal{D}(e))$

□

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 τ_e denote the time at which e may occur. (x, τ_x) belongs to a timed event trace if and only if

$$\tau_d + k \leq \tau_x \leq \tau_d + n \ \wedge \ \tau_e + l \leq \tau_x \leq \tau_e + m \quad .$$

Consider the first constraint. As we have a bundle $\{ d \} \stackrel{k}{\mapsto} x$ it immediately follows that $\tau_d + k \leq \tau_x$. In addition, bundle $\{ d \} \stackrel{n}{\mapsto} t_2$ exists and $\mathcal{U}(t_2)$ holds. From $x \natural t_2$ we infer $\tau_x \leq \tau_d + n$. By symmetry, an analogous reasoning leads to the other constraint.

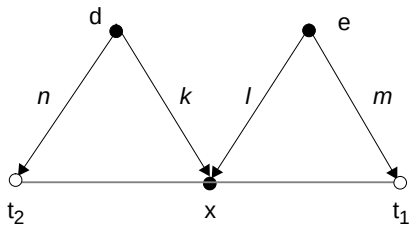


Figure 4: An example timed bundle event structure.

□

4 A Semantics for Timed Probabilistic LOTOS

In this section we present the semantics of $\mathcal{L}$ 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 extension of the semantics presented in [15, 16].

In the rest of this section let $[\![B_i]\!] = \mathcal{E}_i =< (E_i, \#_i, \mapsto_i, l_i), \pi_i, (\mathcal{T}_i, \mathcal{D}_i, \mathcal{U}_i) >,\ i = 1, 2$, with $E_1 \cap E_2 = \emptyset$. We suppose there is an infinite universe E_U of events.

For the semantics of action prefix $(t)\mu; B_1$ a new event e is added with label μ. Event e causally precedes all events in $[\![B_1]\!]$, so for each event e_1 in $[\![B_1]\!]$ a bundle $\{ e \} \mapsto e_1$ is added. The delay of each event e_1 in $[\![B_1]\!]$ is now relative to e, so each bundle $\{ e \} \mapsto e_1$ is associated with a time delay $\mathcal{D}(e_1)$, and the delay $\mathcal{D}(e_1)$ is made zero. The delay $\mathcal{D}(e)$ is set to t. Finally, the urgency predicate is updated in accordance with whether μ is urgent or not.

Definition 4.1 *Timed action prefix*

$[\![(t)\mu; B_1]\!] =< (E, \#_1, \mapsto, l), \pi_1, (\mathcal{T}, \mathcal{D}, \mathcal{U}) >$ where

- $E = E_1 \cup \{ e \}$ for some $e \in E_U \setminus E_1$
- $\mapsto = \mapsto_1 \cup (\{ \{ e \} \} \times E_1)$
- $l = l_1 \cup \{ (e, a) \}$ where $\mu = a$ or $\mu = \widehat{a}$
- $\mathcal{T} = \mathcal{T}_1 \cup \{ ((\{e\}, e_1), t_1) \mid e_1 \in E_1, t_1 = \mathcal{D}_1(e_1) \}$
- $\mathcal{D} = \{ (e_1, 0) \mid e_1 \in E_1 \} \cup \{ (e, t) \}$
- $\mathcal{U} =$ **if** $\mu = \widehat{a}$ **then** $\mathcal{U}_1 \cup \{ (e, true) \}$
  else $\mathcal{U}_1 \cup \{ (e, false) \}$

□

Example 4.2

Suppose $[\![B_1]\!]$ is given in Figure 5(a), then $[\![(2)a; B_1]\!]$ is given in Figure 5(b).

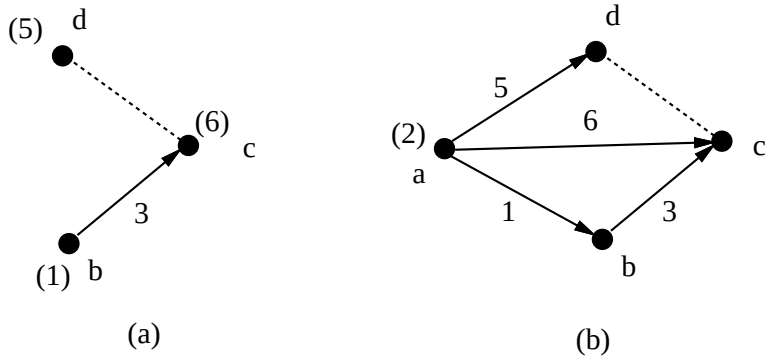


Figure 5: Action prefix semantics.

□

The definition of the semantics of stop, choice, hiding and relabelling is a straightforward extension of the bundle event structure semantics of [15, 16] and is given below. We first define the set of initial events of a timed probabilistic event structure.

Definition 4.3 *Set of initial events*

For $\mathcal{E} =< (E, \#, \mapsto, l), \pi, (\mathcal{T}, \mathcal{D}, \mathcal{U}) >$, the set of initial events of $\mathcal{E}$, denoted $init(\mathcal{E})$, is defined by

$$\{\, e \in E \mid \neg\, (\exists\, X \subseteq E : X \mapsto e) \,\} \quad .$$

□

Definition 4.4 *Stop, choice, hiding and relabelling*

$[\![\mathbf{stop}]\!] =< (\emptyset, \emptyset, \emptyset, \emptyset), \emptyset, (\emptyset, \emptyset, \emptyset) >$

$[\![B_1 \,[]\, B_2]\!] =< (E_1 \cup E_2, \#, \mapsto_1 \cup \mapsto_2, l_1 \cup l_2), \pi_1 \cup \pi_2, (\mathcal{T}_1 \cup \mathcal{T}_2, \mathcal{D}_1 \cup \mathcal{D}_2, \mathcal{U}_1 \cup \mathcal{U}_2) >$ where

- $\# = \#_1 \cup \#_2 \cup (init(\mathcal{E}_1) \times init(\mathcal{E}_2))$

$[\![\mathbf{hide}\ G\ \mathbf{in}\ B_1]\!] =< (E_1, \#_1, \mapsto_1, l), \pi_1, (\mathcal{T}_1, \mathcal{D}_1, \mathcal{U}_1) >$ where

- $l(e) = \mathbf{if}\ l_1(e) \in G\ \mathbf{then\ i\ else}\ l_1(e)$

$[\![B_1[H]]\!] =< (E_1, \#_1, \mapsto_1, l), \pi_1, (\mathcal{T}_1, \mathcal{D}_1, \mathcal{U}_1) >$ where

- $l(e) = \mathbf{if}\ l_1(e) \in H\ \mathbf{then}\ H(e)\ \mathbf{else}\ l_1(e)$

□

Apart from the probability part π the semantics of the probabilistic expression $B{=}B_1 \,[]_p\, B_2$ is equivalent to the standard choice. For non-initial events of B, π is defined as the union of π_1 and π_2. For initial events the situation is slightly more complicated. All probabilities of initial events of B_1 must be multiplied with p and those of B_2 with $1{-}p$. In order to do so we have to distinguish between events that are assigned a probability within B_1 and B_2 and those that are not. We thus obtain:

Definition 4.5 *Probabilistic choice*

$[\![B_1 \,[]_p\, B_2]\!] =< (E_1 \cup E_2, \#, \mapsto_1 \cup \mapsto_2, l_1 \cup l_2), \pi, (\mathcal{T}_1 \cup \mathcal{T}_2, \mathcal{D}_1 \cup \mathcal{D}_2, \mathcal{U}_1 \cup \mathcal{U}_2) >$ where

- $\# = \#_1 \cup \#_2 \cup (init(\mathcal{E}_1) \times init(\mathcal{E}_2))$
- $\pi = (\pi_1 \cup \pi_2 \setminus \{\, (e, r) \mid e \in init(\mathcal{E}_1) \cup init(\mathcal{E}_2) \ \wedge\ r \in (0,1) \,\})$
  $\cup \{\, (e, p) \mid e \in init(\mathcal{E}_1) \ \wedge\ e \notin dom(\pi_1) \,\}$
  $\cup \{\, (e, pq) \mid e \in init(\mathcal{E}_1) \ \wedge\ (e, q) \in \pi_1 \,\}$
  $\cup \{\, (e, 1{-}p) \mid e \in init(\mathcal{E}_2) \ \wedge\ e \notin dom(\pi_2) \,\}$
  $\cup \{\, (e, (1{-}p)q) \mid e \in init(\mathcal{E}_2) \ \wedge\ (e, q) \in \pi_2 \,\} \quad .$

□

Finally we define the semantics of the parallel composition operator. We want to define $[\![B_1 \,|[\, G \,]|\, B_2]\!] {=} \mathcal{E}$. The events of $\mathcal{E}$ are constructed in the following way: an event e that does not need to synchronize (i.e. $l(e) \notin G$) is paired with the auxiliary symbol $*$, and an event that occurs in both processes is paired with all events in the other process that are equally labeled. Thus events are pairs of events of $[\![B_1]\!]$ and $[\![B_2]\!]$, or with one component equal to $*$. Two events are now put in conflict if any of their components are in conflict, or if different

events have a common component different from $*$. A bundle is introduced such that if we take the projection on the i-th component (i=1, 2) of all events in the bundle we obtain a bundle in $[\![B_i]\!]$.

Events are assigned a probability when one of their components is equal to $*$ and the other component is assigned a probability in $[\![B_i]\!]$. Note that no redefinition of the probabilities is necessary, in contrast to what is usual in interleaving approaches.

The time associated with a bundle is equal to the maximum of the times associated with the bundles we get by projecting on the i-th components (i=1, 2) of the events in the bundle, if this projection yields a bundle in $[\![B_i]\!]$. The delay of an event is the maximum of the delays of its components that are not equal to $*$. Finally, an event is urgent when one of its components is equal to $*$ and the other component is urgent.

Definition 4.6 *Parallel composition*

$[\![B_1 \,|[G]|\, B_2]\!] =< (E, \#, \mapsto, l), \pi, (\mathcal{T}, \mathcal{D}, \mathcal{U}) >$, where

- $E = (E_1^f \times \{*\}) \cup (\{*\} \times E_2^f) \cup \{ (e_1, e_2) \in E_1^s \times E_2^s \mid l(e_1) = l(e_2) \}$, where
  - $E_j^s = \{ e \in E_j \mid l(e) \in G \}, j = 1, 2$ (synchronization events)
  - $E_j^f = E_j \setminus E_j^s$ (non-synchronizing events)
- $(e_1, e_2) \# (e_1', e_2') \Leftrightarrow (e_1 \#_1 e_1') \ \vee \ (e_2 \#_2 e_2') \ \vee$
  $(e_1 = e_1' \neq * \ \wedge \ e_2 \neq e_2') \ \vee \ (e_2 = e_2' \neq * \ \wedge \ e_1 \neq e_1')$
- $X \mapsto (e_1, e_2) \Leftrightarrow$
  $\exists\, X_1 \subseteq E_1 : (X_1 \mapsto_1 e_1 \ \wedge \ X = \{ (e_j, e_k) \in E \mid e_j \in X_1 \}) \vee$
  $\exists\, X_2 \subseteq E_2 : (X_2 \mapsto_2 e_2 \ \wedge \ X = \{ (e_j, e_k) \in E \mid e_k \in X_2 \})$
- $l((e_1, e_2)) =$ **if** $e_1 = *$ **then** $l_2(e_2)$ **else** $l_1(e_1)$
- $\pi = \{ ((e, *), p) \mid (e, p) \in \pi_1 \} \cup \{ ((*, e), p) \mid (e, p) \in \pi_2 \}$
- $\mathcal{T}((X, (e_1, e_2))) = max\{h_1, h_2\}$ where
  $h_1 =$ **if** $\exists\, X_1 \subseteq E_1 : (X_1 \overset{t1}{\mapsto} e_1 \ \wedge \ X = \{ (e_j, e_k) \in E \mid e_j \in X_1 \})$
  then $t1$ **else** 0
  $h_2 =$ **if** $\exists\, X_2 \subseteq E_2 : (X_2 \overset{t2}{\mapsto} e_2 \ \wedge \ X = \{ (e_j, e_k) \in E \mid e_k \in X_2 \})$
  then $t2$ **else** 0
- $\mathcal{D}((e_1, e_2)) =$ **if** $e_1 = *$ **then** $\mathcal{D}_2(e_2)$
  else if $e_2 = *$ **then** $\mathcal{D}_1(e_1)$ **else** $max\{\mathcal{D}_1(e_1), \mathcal{D}_2(e_2)\}$
- $\mathcal{U}((e_1, e_2)) =$ **if** $e_1 = *$ **then** $\mathcal{U}_2(e_2)$ **else** $\mathcal{U}_1(e_1)$

□

Example 4.7

In order to illustrate the way time delays are handled by the parallel operator, suppose $[\![B_1]\!]$ is given in Figure 6(a) and $[\![B_2]\!]$ is given in Figure 6(b). Then $[\![B_1 \,|[a, b]|\, B_2]\!]$ is given in Figure 6(c).

□

Because of space limitations we do not give here in full detail the semantics of $[\![P]\!]$ where $P := B$. This semantics can be defined in a similar way as in [15] for the model without

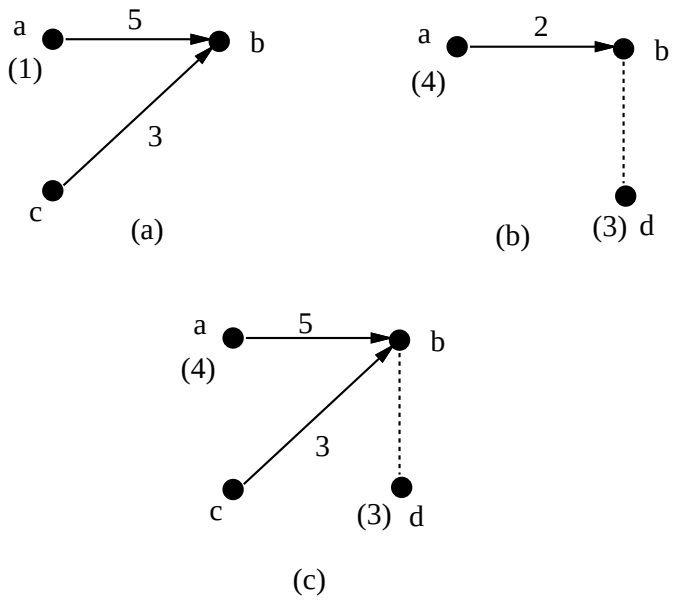


Figure 6: Parallel composition semantics.

time and probabilities, by using standard fixed point theory in the following way. A complete partial order is defined on timed probabilistic event structures, with $[\![\mathbf{stop}]\!]$ as the bottom element $-$. Then for each definition $P := B$ a function $\mathcal{F}_B$ is defined that substitutes an event structure for each occurrence of P in B, interpreting all operators in B as operators on event structures. It is proven that $\mathcal{F}_B$ is continuous, which means that $[\![P]\!]$ can be defined as the least upper bound of the chain $\mathcal{F}_B(-)$, $\mathcal{F}_B(\mathcal{F}_B(-))$, ... For details we refer to [15]; the approach carries over to the timed probabilistic setting in a straightforward manner.

Lemma 4.8 $\forall B \in \mathcal{L}$: $[\![B]\!]$ is a timed probabilistic bundle event structure.

Proof In [13] it is proven that for $[\![B]\!]$, $dom(\pi)$ consists of clusters Q, for which $\sum_{e' \in Q} \pi(e') =$

1. So we only need to prove that for $[\![B]\!]$,

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

(definition 3.10) and this is easily done by induction on the structure of B.

□

The probabilistic and timed extensions of the behaviours are "upward compatible" with the unextended behaviours, in the sense that the semantics of an unextended behaviour is somehow preserved by the semantics of the extended behaviour. This is made precise by the next theorem. For all $B \in \mathcal{L}$ let B_L denote the behaviour obtained from B by replacing all probabilistic choices by standard choices, and discarding all timing and urgency information. Furthermore, let $\mathcal{E}(B_L)$ be the (ordinary) bundle event structure corresponding to B_L as defined in [15]. We now have the following theorem.

Theorem 4.9 *Correctness Theorem*

$$\forall B \in \mathcal{L} : [\![B]\!] =< (E, \#, \mapsto, l), \pi, (\mathcal{T}, \mathcal{D}, \mathcal{U}) > \Rightarrow \mathcal{E}(B_L) = (E, \#, \mapsto, l).$$

Proof Trivial, since removal of the parts concerning $\pi, \mathcal{T}, \mathcal{D}$ and $\mathcal{U}$ in the definition of $[\![\;]\!]$ leads to the bundle event semantics of LOTOS in [15], and the semantics of $[]_p$ is equal to the semantics of [] when the part concerning π is removed.

□

5 Performance Analysis

In this section we provide a simple example to illustrate how unreliable time-dependent systems can be specified using our language, 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.

Although the mapping of regular infinite bundle event structures onto finite representations is currently being investigated —see [17] for an initial proposal for dealing with this— 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 by example.

5.1 Semi-Markov Chains

In this section we consider some aspects of discrete semi-Markov processes, also called semi-Markov chains. For a more elaborated presentation we refer to [10, 27]. It is assumed that the reader is familiar with the notion of (ordinary) Markov chains.

In an ordinary Markov chain the residence time of a state, that is, the probability distribution of staying in this state for a certain amount of time, is restricted to be geometrically distributed. Opposed to this, a *semi-Markov* chain (SMC) allows an arbitrary distribution of residence times. Thus an SMC does not possess the Markovian property that given the current state of the chain the future is independent of the past—when predicting the future for an SMC not only the current state is of importance, but also the amount of time already spent in that state.

State transitions in an SMC are identical to the way in which transitions are carried out in an ordinary Markov chain. In fact, when one abstracts from the residence time distributions in an SMC one obtains a corresponding ordinary Markov chain. This Markov chain is usually called "embedded". Calculating certain quantities for an SMC often boils down to first performing some calculations on its embedded Markov chain, and subsequently interpreting these results for the SMC by taking into account the specific residence time distributions. This applies, for instance, to the calculation of stationary probabilities, as shown below.

Consider some SMC $\mathcal{S}$. Let P_{ij} be the transition probability going from state s_i to s_j. Under the condition that the next state is s_j, the number of time-units until the transition from s_i to s_j has distribution F_{ij}.

Definition 5.1 *State-time probability*

For state s_i of $\mathcal{S}$, the probability $R_i(k)$ of being in s_i for k time-units is

$$R_i(k) = \sum_j P_{ij} * F_{ij}(k) \quad .$$

□

Let r_i denotes the mean of R_i. r_i is usually called the average residence time in state s_i. The fraction of time $\mathcal{S}$ is in a certain state (on the long run) is defined as follows.

Definition 5.2 *Stationary probability of SMC (I)*

For state s_i, and T_i the average number of time-units between successive transitions to state s_i, the fraction of time $\mathcal{S}$ is in s_i, denoted ϕ_i, is defined by

$$\phi_i = \frac{r_i}{T_i} \quad .$$

□

ϕ_i can also be interpreted as the stationary probability of $\mathcal{S}$ being in state s_i.

Let ψ_i be the stationary probability of the embedded Markov chain of $\mathcal{S}$ being in state s_i. An alternative interpretation is that ψ_i denotes the (stationary) probability of $\mathcal{S}$ being in s_i at a transition instant, that is, at a moment of transition. Stated otherwise, ψ_i can be considered as the *fraction of instants* at which the system is in state s_i, considering an infinite amount of transition instants. In order to obtain the *fraction of time* the system is in state s_i, the residence times must be taken into account as follows.

Definition 5.3 *Stationary probability of SMC (II)*

For state s_i of SMC $\mathcal{S}$ with average residence time r_i, and ψ_i the stationary probability of s_i of the embedded Markov chain of $\mathcal{S}$ we have:

$$\phi_i = \frac{\psi_i * r_i}{\sum_j \psi_j * r_j} \quad .$$

□

ψ_i corresponds to the stationary probabilities of the embedded Markov chain and can be calculated in the following way, provided the embedded Markov chain is ergodic.

Definition 5.4 *Stationary probability of embedded Markov chain*

For an ergodic Markov chain $\mathcal{M}$ the stationary probabilities of state s_i of $\mathcal{M}$ can be calculated as follows

$$\begin{aligned} \psi_i &= \sum_j P_{ji} * \psi_j \\ \sum_i \psi_i &= 1 \quad . \end{aligned}$$

□

In the following example we follow the following procedure. For an SMC $\mathcal{S}$ we first solve the stationary probabilities, ψ_i, of its embedded Markov chain using the last definition. (Note that the considered SMCs has an ergodic embedded Markov chain.) Using definition 5.1 the average residence times of $\mathcal{S}$ are obtained. These results are used to obtain expressions for ϕ_i (using definition 5.3), and for T_i (using definition 5.2).

5.2 Unreliable Coffee-Machine

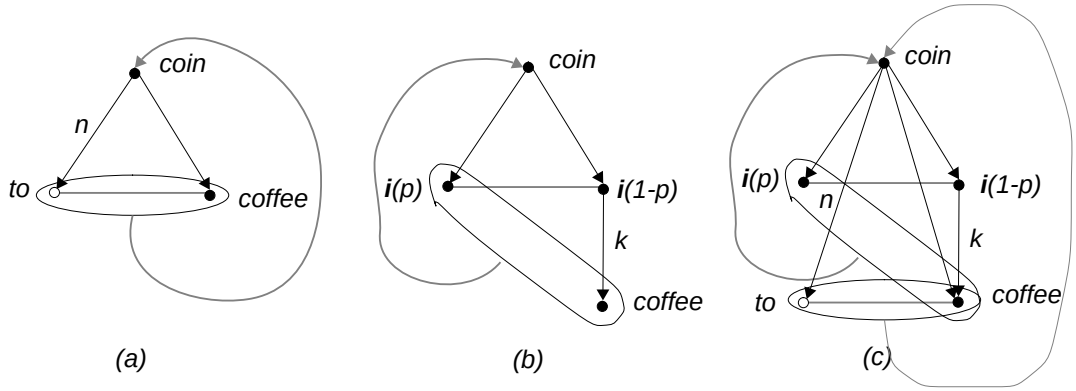


Figure 7: $[\![U]\!]$, $[\![C]\!]$, and $[\![S]\!]$.

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,

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

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,

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

The overall 'system' is specified by

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

In order to make synchronisations 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 posesses a certain regular structure. A first attempt of exploiting this regularity can be found in [17].

Figure 8 illustrates for $[\![U]\!]$ how the unfolding is performed. Each successive unfolding is obtained by instantiating the original structure. In each structure the circle identifies the initial events of the next instantiation. The sequence of structures obtained by unfolding is very similar to the fixpoint semantics of recursive processes as defined in [15].

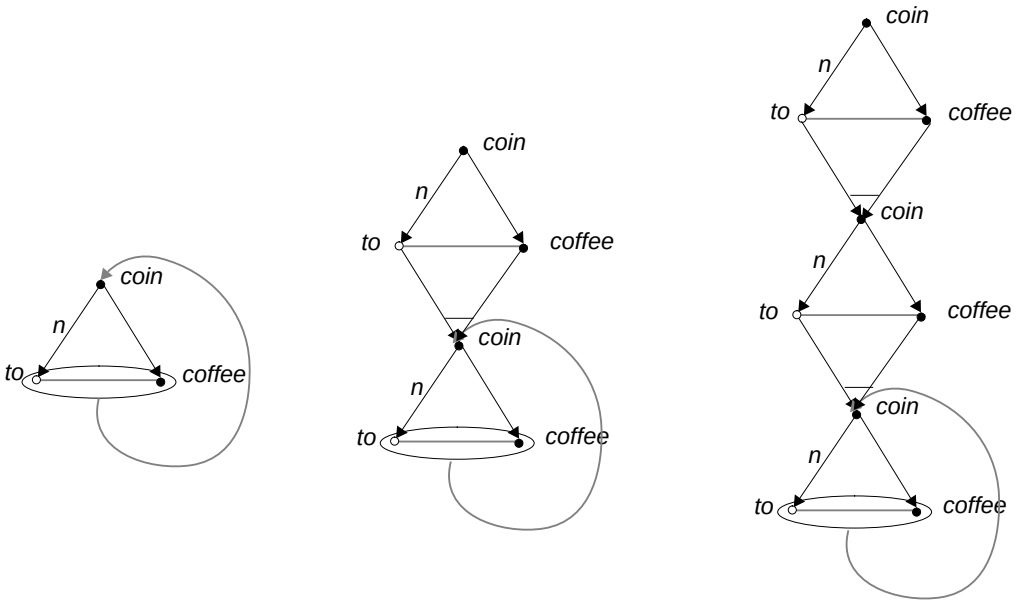


Figure 8: Unfoldings of $[\![U]\!]$.

Assume 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:

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

s_1 represents the case in which no coffee is offered, s_2 represents the case in which actually coffee is offered, 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).

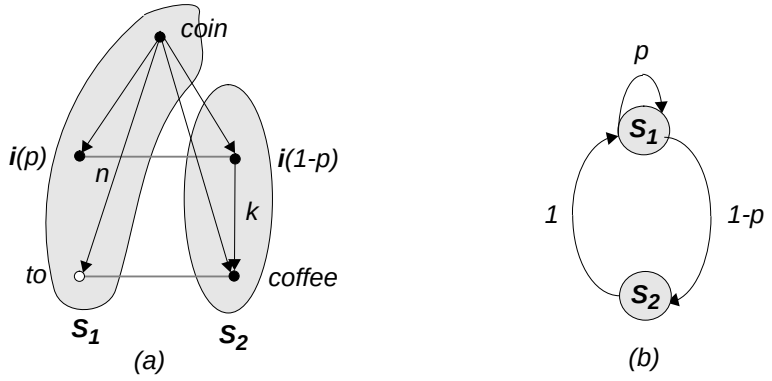


Figure 9: Grouping of events (a) and semi-Markov chain (b).

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 definition 5.4 we obtain for the stationary probabilities the following results:

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

Subsequently, using definition 5.3 and the definitions of r_i we obtain

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

(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).

The model presented is an extension of the bundle event structure semantics for LOTOS reported in [15]. 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 (precongruences) 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 [11]. 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 [17]. 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.

Acknowledgement 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 J.A. Bergstra. Real time process algebra. *Formal Aspects of Computing*, 3(2):142–188, 1991.

[2] J.C.M. Baeten, J.A. Bergstra, and S.A. Smolka. Axiomatizing probabilistic processes: ACP with generative probabilities. In W.R. Cleaveland et. al., editor, *Proceedings of CONCUR'92: Theories of Concurrency*, LNCS 630, pages 472–485. Springer-Verlag, 1992.

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

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

[5] T. Bolognesi and F. Lucidi. Timed process algebra with urgent interactions and a unique powerful binary operator. In J.W. de Bakker (et. al.), editor, *Real-Time: Theory in Practice*, LNCS 600, pages 124–148. Springer-Verlag, 1992.

[6] I. Christoff. Testing equivalences and fully abstract models for probabilistic processes. In J.C.M. Baeten and J.W. Klop, editors, *Proceedings of CONCUR'90: Theories of Concurrency*, LNCS 458, pages 126–140. Springer-Verlag, 1990.

[7] C. Fidge. A constraint-oriented real-time process calculus. In M. Diaz and R. Groz, editors, *Formal Description Techniques V*, pages 363–378. North-Holland, 1993.

[8] A. Giacalone, C.-C. Jou, and S.A. Smolka. Algebraic reasoning for probabilistic concurrent systems. In M. Broy and C.B. Jones, editors, *Proceedings of the Working Conference on Programming Concepts and Methods*. North-Holland, 1990.

[9] H. Hansson and B. Jonsson. A calculus for communicating systems with time and probabilities. In *Proceedings of 11th IEEE Real-Time Systems Symposium*, pages 278–287. IEEE Computer Society Press, 1990.

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

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

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

[13] J.-P. Katoen, R. Langerak, and D. Latella. Modeling systems by probabilistic process algebra: An event structures approach. Technical Report 93-29, University of Twente, 1993.

[14] J-P. Katoen, R. Langerak, and D. Latella. Modelling systems by probabilistic process algebra—an event structures approach. In R. Tenney et. al. editor, *Formal Description Techniques VI.* North-Holland, 1994.

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

[16] 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.

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

[18] G. Lowe. Representing nondeterminism and probabilistic behaviour in reactive processes. Technical Report PRG-TR-12-93, Oxford University Computing Laboratory, 1993.

[19] A. Maggiolo-Schettini and J. Winkowski. Towards an algebra for timed behaviours. *Theoretical Computer Science*, 103:335–363, 1992.

[20] C. Miguel, A. Fernández, and L. Vidaller. LOTOS extended with probabilistic behaviours. *Formal Aspects of Computing*, 5:253–281, 1993.

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

[22] D. Murphy. Time and duration in noninterleaving concurrency. *Fundamenta Informaticae*, 19:403–416, 1993.

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

[24] X. Nicollin, J-L. Richier, J. Sifakis, and J. Voiron. ATP: an Algebra for Timed Processes. In M. Broy and C.B. Jones, editors, *Proceedings of the Working Conference on Programming Concepts and Methods*, pages 415–442. North-Holland, 1990.

[25] X. Nicollin and J. Sifakis. An overview and synthesis on timed process algebras. In J.W. de Bakker (et. al.), editor, *Real-Time: Theory in Practice*, LNCS 600, pages 526–548, 1992.

[26] N. Rico and G. v. Bochmann. Performance description and analysis for distributed systems using a variant of LOTOS. In B. Jonsson et. al., editor, *Protocol Specification, Testing, and Verification IX*, pages 199–213. North-Holland, 1991.

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

[28] S. Schneider. An operational semantics for Timed CSP. *Information and Computation*, 1994. (to appear).

[29] 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.

[30] 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.