ELSEVIER
Science of Computer Programming 36 (2000) 97–127
Science of Computer Programming
www.elsevier.nl/locate/scico

Automated compositional Markov chain generation for a plain-old telephone system

Holger Hermanns [*,1], Joost-Pieter Katoen [1]

Lehrstuhl für Informatik VII, Friedrich-Alexander Universität Erlangen-Nürnberg, Martensstrasse 3, D-91058 Erlangen, Germany

Abstract

Obtaining performance models, like Markov chains and queueing networks, for systems of significant complexity and magnitude is a difficult task that is usually tackled using human intelligence and experience. This holds in particular for performance models of a highly irregular nature. In this paper we argue by means of a non-trivial example – a plain-old telephone system (POTS) – that a stochastic extension of process algebra can diminish these problems by permitting an automatic generation of Markov chains. We introduce a stochastic process algebra that separates the advance of time and action occurrences. For the sake of specification convenience we incorporate an elapse operator that allows the modular description of time constraints where delays are described by continuous phase-type distributions. Using this language we provide a formal specification of the POTS and show how a stochastic process of more than 10^7 states is automatically obtained from this system description. Finally, we aggregate this model compositionally using appropriate stochastic extensions of (strong and weak) bisimulation. As a result we obtain a highly irregular Markov chain of about 700 states in an automated way, which we use to carry out a transient performance analysis of the POTS. © 2000 Elsevier Science B.V. All rights reserved.

Keywords: Automatic Markov chain generation; Compositional aggregation; Constraint-oriented specification; Performance analysis; Plain-old telephone system; Stochastic process algebra

1. Introduction

The construction of system models suited for performance and reliability analysis is a difficult task that to a large extent is solved using human intelligence and experience. Due to the increase in size and complexity of systems, like embedded and distributed systems, this tendency seems even growing: performance modelling becomes a task

* Corresponding author.
[1] Current address: University of Twente, Department of Computer Science, P.O. Box 217, 7500 AE Enschede, The Netherlands.

dedicated to specialists in particular for models with a high degree of irregularity. Traditional performance models like Markov chains and queueing networks are widely accepted as simple and yet adequate models in different areas, but the lack of hierarchical composition significantly hampers the modelling of systems that are developed nowadays.

On the other hand, for describing and analysing the functional system behaviour various specification formalisms have been developed that are strongly focussed on the facility to model systems in a compositional, hierarchical manner. A prominent example of such specification formalisms is the class of *process algebra*. Developed in the mid-80's on a strong mathematical basis, process algebra has emerged as an important framework to achieve compositionality. Process algebra provides a formal apparatus for reasoning about structure and behaviour of systems in a compositional way. In addition, an abstraction mechanism provides means to treat system components as black boxes, making their internal structure invisible. A formal semantics maps process algebra terms onto transition systems consisting of a set of states with a distinguished initial state, and a transition relation that describes how the system evolves from one state to another. Transitions are labelled with actions, the most primitive notions in process algebra. Transition systems can be visualised by drawing states as nodes of a graph and transitions as directed edges (labelled with actions) between them. To overcome the absence of hierarchical, compositional facilities in performance modelling, in the early 90's a research effort started on integrating performance aspects into process algebras – the class of *stochastic process algebras* emerged.

The basic idea underlying these class of process algebras is to change the role of action-prefix, denoted by $a; P$ for action a and process P. Originally, the expression $a; P$ simply means that first an action a is offered, and after the appearance of a the process behaves like P. In stochastic process algebras, like EMPA [2], PEPA [28] or TIPP [19], a real-valued rate is associated to actions that determines probabilistically the delay prior to an action. For rate λ, the term $(a, \lambda); P$ denotes that action a is offered after a certain delay that is determined by an exponential distribution. More precisely, $(a, \lambda); P$ offers a and evolves into P within t time units with probability $1 - e^{-\lambda t}$. As a semantical model for stochastic process algebras, transition systems are used where transitions are labelled with pairs of actions and rates. By omitting the action labels – but keeping the rate information – one obtains a (homogeneous) continuous-time Markov chain for which steady state and transient performance metrics can be obtained using traditional techniques [40]. To put it in a nutshell, stochastic process algebras provide a compositional and algebraic formalism for describing Markov chains.

The benefits of this process algebra approach for the specification and generation of Markov chains is shown by several results. In the stochastic setting, bisimulation equivalence, one of the most important notions of equivalence to compare labelled transition systems, is shown to correspond to lumpability, a notion central for the aggregation of Markov chains [28]. In addition, the congruence property of bisimulation allows the aggregation to be carried out compositionally, i.e. component-wise. Several (small and moderate) case studies have shown the practicality of the approach, for instance [1, 13], and important progress has been made in exploiting the compositional structure of the specification for performance analysis purposes [30, 33, 29].

The basic distinction between the aforementioned stochastic process algebras is the treatment of time consumption in case of interaction. Technically, this amounts to the computation of the resulting rate in case two actions like (a, λ) and (a, μ) synchronise. The most natural interpretation is to require both delays to have expired before the interaction (on a) can take place. The thus resulting distribution, the product of two exponential distributions of rate λ and μ respectively, is, however, not an exponential distribution. (The product of two distribution functions amounts to the maximum of their corresponding random variables.) To overcome this problem, several solutions have been suggested that, however, either lack a clear stochastic interpretation or have a restricted applicability. In this paper we maintain the (to our opinion) most natural stochastic interpretation (i.e. maximum of random variables) by explicitly *separating* between the advance of time and the occurrence of actions. This distinction leads to a behaviour where two distinct phases are mixed. Phases, during which one or more actions occur (together with their corresponding state changes), but where no time elapses, alternate with phases where time passes, but during which no actions happen. This separation of discrete and continuous phases is similar to that in timed process algebras as proposed in [35, 42]. We introduce a stochastic extension of Basic[2] LOTOS (Language Of Temporal Ordering Specification), the process algebra standardised by the ISO in 1989. Its formal semantics is defined using a mixture of labelled transition systems and Markov chains, called *interactive Markov chains* [22], in which both action-labelled and rate-labelled transitions co-exist.

We show the applicability of stochastic process algebras to a large example, a plain-old telephone system (POTS) with more than 10^7 states whose functional specification comprises more than thousand lines of process algebra code. For the sake of specification convenience we incorporate an *elapse operator* that allows the modular description of time constraints where delays are described by continuous phase-type distributions [36], a general class of distribution functions that includes exponential, Erlang, Cox and mixtures of exponential distributions. This operator facilitates the description of time constraints in a *modular* way, that is, as separated processes that are composed in parallel with the untimed specification. This art of specifying time constraints fits well in the constraint-oriented specification style [41], a specification style where process descriptions are considered as constraints.

Using our language we provide a formal specification of the time-constrained POTS and show how a stochastic process of more than 10^7 states is automatically obtained from this system description. Subsequently, we aggregate this model compositionally using strong and weak bisimulation and appropriate stochastic extensions thereof. As a result we obtain a highly irregular Markov chain of about 700 states in an automated way, which we use to carry out a transient performance analysis of the POTS.

[2] The term Basic refers to the fact that data aspects are absent.

To summarise, this paper makes the following contributions:

- it introduces a stochastic extension of Basic LOTOS that is based on the separation of advancing time and the occurrence of actions, [3]
- it provides a stochastic formal specification of the POTS using a novel and simple elapse operator that allows phase-type durations while supporting a constraint-oriented specification style, and
- it shows how a highly irregular Markov chain is automatically generated and minimised starting from this specification.

Organisation of the paper: Section 2 provides a brief introduction into the process algebra Basic LOTOS. This introduction includes a formal syntax and semantics, definitions of the most important equivalence notions (bisimulation) and a short description of tool and algorithmic support. Section 3 covers extensively the same ingredients for our stochastic extension of LOTOS and describes how Markov chains are generated and aggregated. Section 4 presents an informal description of the POTS, our example telephone system and presents fragments of the formal LOTOS specification. Section 5 shows how time constraints are included in the POTS specification using the elapse operator in a modular way. Section 6 exemplifies the strategy of compositional Markov chain generation and reduction by means of the POTS, and presents our results obtained by transient performance analysis. Section 7 concludes the paper.

2. The process algebra Basic LOTOS

In this section we present the basic definitions and concepts of Basic LOTOS that are needed for the understanding of this paper. For a more thorough treatment of this process algebra we refer to [4].

2.1. Syntax

Basic LOTOS has a parallel operator that allows multi-party synchronisation (like in CSP). It has a special internal action **i** modelling internal (i.e. unobservable) activity. Another distinctive feature is the so-called *disabling* operator which is quite convenient for protocol specification (e.g. a data-phase can be disabled by a disconnection phase). Let P and Q be Basic LOTOS expressions, L be the universe of observable actions, a be an action from $L \cup \{\mathbf{i}\}$, $A \subseteq L$ be a set of actions, f be a function from $L \cup \{\mathbf{i}\}$ to $L \cup \{\mathbf{i}\}$ such that $f(\mathbf{i}) = \mathbf{i}$, and x be a process identifier. Then the syntax of Basic LOTOS is recursively given by Table 1. $P\,|[\emptyset]|\,Q$ is abbreviated as $P\,|||\,Q$. The syntax of a process definition is $x := P$. A set of process definitions is called a *process environment*. A behaviour expression is always considered in the context of a process environment. To avoid too many parentheses we adopt the convention that the operators have a decreasing binding power in the following order: ;, [], $|[A]|$, $[>$, $\gg$, **hide** A **in** and $[f]$.

[3] In fact we introduce a subset of the language that is originally proposed by the first author in [26].

Table 1
Syntax of Basic LOTOS

Inaction	**stop**	Disabling	$P\,[>Q$
Successful termination	**exit**	Parallel composition	$P\,\|[A]\|\,Q$
Action-prefix	$a;\,P$	Hiding	**hide** A **in** P
Choice	$P\,[]\,Q$	Relabelling	$P[f]$
Sequential composition (enabling)	$P \gg Q$	Process instantiation	x

2.2. Semantics

The formal semantics associates to each Basic LOTOS term P a labelled transition system where states are identified with expressions – the initial state being the term P – and for which the transition relation is defined as the smallest relation that satisfies the rules in Table 2. The distinguished action δ that appears in the rule for **exit** models the successful termination of an expression. It is, for example, used in the semantics of sequential composition (like $P \gg Q$) to pass the control from P to Q in case P successfully terminates. Notice that parallel processes are forced to synchronise on δ; this allows, for example, in $(P\,|||\,Q) \gg R$, process R to start its execution only once both P and Q successfully terminated.

2.3. Bisimulation

By Table 2 a labelled transition system is associated to each Basic LOTOS expression. Often, these transition systems are too concrete in the sense that we would like to identify certain behaviour expressions with different transition systems, e.g. on the basis of observability criteria. For these reasons several equivalences and pre-orders have been defined; for an overview see [14, 15]. One of the most important classes of equivalence relations are the so-called *bisimulation* equivalences. We consider two variants, known as strong [38, 34] and weak [34] bisimulation. These relations are recalled below, defined in a format borrowed from [16]. For the sake of convenience let $\mathscr{L}$ denote the set of expressions defined by the syntax of Table 1. In the following definitions let action $a \in L \cup \{\mathbf{i}, \delta\}$.

Definition 1 (*Strong bisimulation*). An equivalence relation $\mathscr{S}$ on $\mathscr{L}$ is a *strong bisimulation* iff for any pair $(P,Q) \in \mathscr{L} \times \mathscr{L}$ we have that $(P,Q) \in \mathscr{S}$ implies for all actions a and all equivalence classes $C \in \mathscr{L}/_{\mathscr{S}}$:

$$\gamma_{\mathrm{s}}(P,a,C) = \gamma_{\mathrm{s}}(Q,a,C) \quad \text{with } \gamma_{\mathrm{s}}(R,a,C) = \begin{cases} 1 & \text{if } \{R' \in C \mid R \xrightarrow{a} R'\} \neq \emptyset, \\ 0 & \text{otherwise.} \end{cases}$$

Processes P and Q are *strongly bisimilar*, denoted $P \sim Q$, if $(P,Q) \in \mathscr{S}$ with $\mathscr{S}$ a strong bisimulation.

Table 2
Standard semantics of Basic LOTOS

$\dfrac{}{\mathbf{exit}\xrightarrow{\delta}\mathbf{stop}} \quad \dfrac{}{a;\, P\xrightarrow{a}P}$	$\dfrac{P\xrightarrow{\delta}P'}{P \gg Q\xrightarrow{\mathbf{i}}Q}$
$\dfrac{P\xrightarrow{a}P'}{P \gg Q\xrightarrow{a}P' \gg Q}\ (a\neq\delta)$	$\dfrac{P\xrightarrow{a}P'}{\begin{array}{l}P\ [> Q\xrightarrow{a}P'\ [> Q\\ Q\,[>P\xrightarrow{a}P'\end{array}}\ (a\neq\delta)$
$\dfrac{P\xrightarrow{a}P'}{\begin{array}{l}P\,\vert[A]\vert\,Q\xrightarrow{a}P'\,\vert[A]\vert\,Q\\ Q\,\vert[A]\vert\,P\xrightarrow{a}Q\,\vert[A]\vert\,P'\end{array}}\ (a\notin A\cup\{\delta\})$	$\dfrac{P\xrightarrow{a}P'}{\begin{array}{l}P\,[]\,Q\xrightarrow{a}P'\\ Q\,[]\,P\xrightarrow{a}P'\end{array}}$
$\dfrac{P\xrightarrow{a}P',\, Q\xrightarrow{a}Q'}{P\,\vert[A]\vert\,Q\xrightarrow{a}P'\,\vert[A]\vert\,Q'}\ (a\in A\cup\{\delta\})$	$\dfrac{P\xrightarrow{\delta}P'}{\begin{array}{l}P\,[>Q\xrightarrow{\delta}P'\\ Q\,[>P\xrightarrow{\delta}P'\end{array}}$
$\dfrac{P\xrightarrow{a}P'}{\mathbf{hide}\ A\ \mathbf{in}\ P\xrightarrow{a}\mathbf{hide}\ A\ \mathbf{in}\ P'}\ (a\notin A)$	$\dfrac{P\xrightarrow{a}P'}{P[f]\xrightarrow{f(a)}P'[f]}$
$\dfrac{P\xrightarrow{a}P'}{\mathbf{hide}\ A\ \mathbf{in}\ P\xrightarrow{\mathbf{i}}\mathbf{hide}\ A\ \mathbf{in}\ P'}\ (a\in A)$	$\dfrac{P\xrightarrow{a}P'}{x\xrightarrow{a}P'}\ (x:=P)$

Stated in words, two processes P and Q are strongly bisimilar iff all possible transitions from P can be simulated (i.e. mimicked) by equally labelled transitions in Q that result in bisimilar states, and vice versa. Strong bisimulation treats internal actions ($\mathbf{i}$) in the same way as observable actions. For example, the following two processes are distinguished by strong bisimulation:

$$a;\, \mathbf{i};\, \mathbf{stop} \not\sim a;\, \mathbf{stop}$$

From an external observer's point of view, however, these two processes cannot be distinguished, since the occurrence of $\mathbf{i}$ is internal. Weak bisimulation – as the name suggests a weak version of strong bisimulation – takes this observability perspective and does not distinguish between the two aforementioned processes. Its definition is obtained from the definition of $\sim$ by replacing the transition relation $\xrightarrow{a}$ by $\overset{a}{\Longrightarrow}$. For a an external action (i.e. $a \neq \mathbf{i}$), $\overset{a}{\Longrightarrow}$ denotes an a-transition that is preceded and/or followed by an arbitrary number (possibly zero) of internal actions, i.e. $\overset{a}{\Longrightarrow} = \xrightarrow{\mathbf{i}^*} \xrightarrow{a} \xrightarrow{\mathbf{i}^*}$. Otherwise, if a is internal ($a = \mathbf{i}$), then $\xrightarrow{a}$ abbreviates $\xrightarrow{\mathbf{i}^*}$ (and not $\xrightarrow{\mathbf{i}^+}$).

Definition 2 (*Weak bisimulation*). An equivalence relation $\mathcal{S}$ on $\mathcal{L}$ is a *weak bisimulation* iff for any pair $(P,Q) \in \mathcal{L} \times \mathcal{L}$ we have that $(P,Q) \in \mathcal{S}$ implies for all actions a and all equivalence classes $C \in \mathcal{L}/_{\mathcal{S}}$:

$$\gamma_{\mathrm{W}}(P,a,C) = \gamma_{\mathrm{W}}(Q,a,C) \quad \text{with } \gamma_{\mathrm{W}}(R,a,C) = \begin{cases} 1 & \text{if } \{R' \in C \mid R \overset{a}{\Longrightarrow} R'\} \neq \emptyset, \\ 0 & \text{otherwise.} \end{cases}$$

Processes P and Q are *weakly bisimilar*, denoted $P \approx Q$, if $(P,Q) \in \mathcal{S}$ with $\mathcal{S}$ a weak bisimulation.

In the presence of composition operators, it is highly desirable that equivalence relations are *substitutive*. Intuitively, substitutivity allows to replace components by equivalent ones within a large specification without changing the overall behaviour. Indeed, strong and weak bisimulation possess this desirable property. (Strictly speaking, $\approx$ is not substitutive with respect to the choice operator, but a slight variant of $\approx$, known as observation congruence [34], is.) Substitutive equivalences are also called *congruences*. Practically important, such equivalences facilitate compositional aggregation, where the state space of components may be reduced in size, without affecting any significant property of the whole specification. Compositional aggregation has successfully been applied to a variety of systems, we refer to [8] for an impressive industrial case study of this technique.

2.4. Algorithmic and tool support

Various algorithms have been developed to compute strong and weak bisimulation on finite-state processes. An efficient way for computing strong bisimulation is based on the partition refinement algorithms of [37]. It has a worst-case time complexity that is logarithmically proportional in the number of states and linearly in the number of transitions. Weak bisimulation has a time complexity that is cubic in the number of states, which is due to the most time-consuming step, the computation of the transitive closure of $\xrightarrow{\mathbf{i}}$. Various tools for process algebras do exist that support the specification and analysis, see the overview in [31]. For our example telephone system we use the components CÆSAR [12] and ALDÉBARAN [6] that are distributed as part of the CADP tool-box (CÆSAR/ALDÉBARAN Development Package). ALDÉBARAN is a verification tool-set that supports efficient, compositional equivalence checking of labelled transition systems, and model checking. CÆSAR supports the translation of (full) LOTOS specifications into a transition system format that can be processed by ALDÉBARAN.

3. Compositional Markov chain specification

In this section we introduce the basic definitions and properties of the calculus we investigate. It includes a distinct type of prefixing to specify exponentially distributed delays.

3.1. Syntax

The basic idea of our stochastic process algebra is to extend the syntax of $\mathscr{L}$ with a (Markovian) timed prefix, denoted by (λ); P, where λ is a parameter that uniquely determines an exponential distribution. The process (λ); P evolves into P within t time units with probability $1-e^{-\lambda t}$. That is, it behaves like P after a certain delay that is determined by a (continuous) random variable, X say, such that $\text{Prob}(X \leqslant t)$ equals $1-e^{-\lambda t}$ for positive t, and equals zero otherwise. The mean duration prior to P is $1/\lambda$, the expectation of random variable X. The real-valued parameter λ is called the *rate* of the distribution. The prefix (λ); P can be considered as the probabilised version of the timed prefix (t); P that occurs in several timed process algebras, like in TCCS [35].

3.2. Semantics

Due to the explicit separation between the (probabilistic) advance of time and the occurrence of actions, the semantics of our stochastic extension of Basic LOTOS is defined using two transition relations. The transition relation $\xrightarrow{a}$ corresponds to action occurrences and is defined using the inference rules of Table 2. The transition relation $\overset{\lambda}{\longmapsto}$ represents the passage of time; this relation is the smallest relation defined using the rules of Table 3.[4] The inference rules associate to each expression a transition system in which we have action- and rate-labelled transitions. This model is adopted from [22]. It can be considered as an extension of (continuous-time) Markov chains with the possibility of interaction; accordingly it is referred to as *interactive Markov chains*. We remark that using our stochastic process algebra we can describe labelled transition systems (by omitting the time-prefix (λ); P), as well as pure Markov chains (by omitting action-prefix).

Let us consider and justify the semantics of some operators in more detail as their interpretation has changed with respect to the untimed setting.

Action – prefix: Observe that an action-prefix cannot be involved in a timed transition. This entails that actions happen as soon as they are possible.

Parallel composition – delaying: Parallelly composed processes can delay completely independently. This is different from a deterministic time setting where such processes typically are forced to synchronise on the advance of time, see, e.g. [35]. The

[4] Since the multiplicity of transitions is important (like in probabilistic process calculi) $\overset{\lambda}{\longmapsto}$ is a multi-relation.

Table 3
Timed transitions for our stochastic process algebra

$\dfrac{}{(\lambda);\, P \overset{\lambda}{\mapsto} P}$	$\dfrac{P \overset{\lambda}{\mapsto} P'}{\begin{array}{l} P \,[]\, Q \overset{\lambda}{\mapsto} P' \\ Q \,[]\, P \overset{\lambda}{\mapsto} P' \end{array}}$	$\dfrac{P \overset{\lambda}{\mapsto} P'}{P \gg Q \overset{\lambda}{\mapsto} P' \gg Q}$
$\dfrac{P \overset{\lambda}{\mapsto} P'}{\begin{array}{l} P \,[>Q \overset{\lambda}{\mapsto} P' \,[>Q \\ Q \,[>P \overset{\lambda}{\mapsto} P' \end{array}}$	$\dfrac{P \overset{\lambda}{\mapsto} P'}{P[f] \overset{\lambda}{\mapsto} P'[f]}$	$\dfrac{P \overset{\lambda}{\mapsto} P'}{x \overset{\lambda}{\mapsto} P'}\ (x := P)$
$\dfrac{P \overset{\lambda}{\mapsto} P'}{\begin{array}{l} P \,\vert[A]\vert\, Q \overset{\lambda}{\mapsto} P' \,\vert[A]\vert\, Q \\ Q \,\vert[A]\vert\, P \overset{\lambda}{\mapsto} Q \,\vert[A]\vert\, P' \end{array}}$	$\dfrac{P \overset{\lambda}{\mapsto} P'}{\textbf{hide } A \textbf{ in } P \overset{\lambda}{\mapsto} \textbf{hide } A \textbf{ in } P'}$	

justification for independent delaying relies on the so-called *memory-less property* of exponential distributions.[5]

Definition 3 (*Memory-less property*). Let X be an exponentially distributed random variable and t, t' be positive reals. X possesses the memory-less property iff $\text{Prob}(X \leqslant t+t' \mid X > t) = \text{Prob}(X \leqslant t')$.

To understand the meaning of this property in our context consider the process

$$(\lambda);\, a;\, \textbf{stop} \,|||\, (\mu);\, b;\, \textbf{stop}$$

and suppose that the delay of the left process finishes first (with rate λ). Due to the memory-less property, the remaining delay until action b may occur is determined by an exponential distribution with rate μ, exactly the delay prior to the enabling of b before the delay of the first process has been finished. Stated differently, the delay of the left process does not have any impact on the remaining delay in the other process – the advance of time governed by memory-less distributions is independent. One of the consequences of this independent delaying is that an expansion law is obtained rather straightforwardly. This is neither the case for a deterministic time setting [18] nor for a stochastic calculus that supports arbitrary distributions [9]. The expansion law allows the elimination of parallel composition in terms of more elementary operators (like action-prefix and choice) and is of central importance for the verification and correctness-preserving transformation of process algebra specifications.

Synchronisation: Interactions can only appear when all participants are ready to engage in it. For instance, the process

$$(\lambda);\, a;\, \textbf{stop} \,|[a]|\, (\mu);\, a;\, \textbf{stop}$$

[5] In fact, the exponential distribution is the only continuous probability distribution function that possesses this property.

is able to offer action a after a delay with rate λ and a delay with rate μ (due to the memory-less property these delays can be in arbitrary order). Technically speaking, action a can be offered after the maximum of two exponentially distributed random variables. Notice that this type of synchronisation naturally follows from our choice to explicitly separate time- and action-transitions – all delays prior to an interaction must have elapsed before the interaction can take place. We consider this interpretation as a natural assumption, like many others do [7, 9, 20, 35].

In stochastic process algebras like TIPP [19], PEPA [28] and EMPA [2] that are also focussed on exponential distributions, the action-prefix $a; P$ is replaced by $(a,\lambda); P$. The basic difference between these algebras is the calculation of the resulting rate in case of synchronisation:

$$(a,\lambda);\ \mathbf{stop}\,|[a]|\,(a,\mu);\ \mathbf{stop} = (a,\ ?);\ \mathbf{stop}$$

TIPP proposes the product of rates, EMPA forbids this type of synchronisation and requires one component to determine the rate only while the other components need to be passive (i.e. willing to accept any rate), and finally PEPA computes the maximum of mean delays while incorporating the individual synchronisation capacities of processes. None of these algebras uses the maximum (as we do), since the class of exponential distributions is not closed under product. [6] The absence of this closure property does not pose a problem for our approach since – due to the separation of the time- and action-transitions – we can model the product of two exponential distributions explicitly as the corresponding phase-type distribution, see Definition 7.

Choice: The process $P\,[]\,Q$ behaves either as P or Q, but not both. At execution the fastest process, i.e. the process that is enabled first, is selected. This is known as the race condition. If this fastest process is not uniquely determined, a non-deterministic selection among the fastest processes is made.

Selecting the fastest process among two time-prefixed processes boils down to considering the *minimum* of the exponential distributions involved. The minimum of two exponential distributions is an exponential distribution with the sum of the individual rates. Thus, the delay until the resolution of the choice between two time-prefixed processes is exponentially distributed with a rate that equals the sum of the rates of the time-prefixes.

In case of a competition between a time-prefixed and an action-prefixed process, the latter process will be selected if the offered action is not blocked by the environment (e.g. an internal action). This stems from the fact that the probability that an exponentially distributed duration finishes instantaneously, is zero. However, to achieve compositionality, the precedence of timed-prefixed over (non-blocked) action-prefixed processes is not reflected in the rules defining the transition relations $\xrightarrow{a}$ and $\overset{\lambda}{\mapsto}$. Instead, we introduce below a notion of equality, weak Markovian bisimulation, that (among others) takes care of the interplay of time and action-prefixes.

[6] Recall that the maximum of two statistically independent random variables amounts to the product of their corresponding distribution functions.

3.3. Bisimulation and its stochastic interpretation

For similar reasons as for the untimed case (cf. Section 2) we like to identify certain behaviour expressions with different interactive Markov chains. Therefore, we extend the notions of strong and weak bisimulation to the stochastic setting. The resulting equivalence notions are called *Markovian bisimulation* equivalences. They are defined in the same style as strong and weak bisimulation. Let $\mathscr{L}'$ denote the language defined according to the syntax of Table 1 extended with the prefix $(\lambda); P$ (and let '$\{\!|$','$|\!\}$' delimit a multiset).

Definition 4 (*Strong Markovian bisimulation*). An equivalence relation $\mathscr{S}$ on $\mathscr{L}'$ is a *strong Markovian bisimulation* iff for any pair $(P,Q)\in\mathscr{L}'\times\mathscr{L}'$ we have that $(P,Q)\in\mathscr{S}$ implies for all actions a and all equivalence classes $C\in\mathscr{L}'/_{\mathscr{S}}$:

(1) $\gamma_{\mathrm{s}}(P,a,C)=\gamma_{\mathrm{s}}(Q,a,C)$, and

(2) $\gamma_{\mathrm{m}}(P,C)=\gamma_{\mathrm{m}}(Q,C)$ with $\gamma_{\mathrm{m}}(R,C)=\sum_{\lambda}\{\!|\ \lambda\,|\,R\overset{\lambda}{\longmapsto}R', R'\in C\ |\!\}$

Processes P and Q are *strongly Markovian bisimilar*, denoted $P\sim_{\mathrm{m}}Q$, if $(P,Q)\in\mathscr{S}$ with $\mathscr{S}$ a strong Markovian bisimulation.

Strong Markovian bisimulation is defined in the same style as probabilistic bisimulation [32] and strong equivalence [28]. Intuitively, two processes are strongly Markovian bisimilar if they are strongly bisimilar, and if the cumulated rates of moving by timed transition to each equivalence class are equal. $\sim_{\mathrm{m}}$ is a congruence with respect to all language operators. Like for strong bisimulation, $\sim_{\mathrm{m}}$ treats internal actions like any other action. An important result [28] is that for timed transitions, strong Markovian bisimulation coincides with *lumpability*, a notion on continuous-time Markov chains that justifies the aggregation of Markov chains without affecting performance properties. Due to the fact that $\sim_{\mathrm{m}}$ is a congruence, lumping can be performed in a compositional way, i.e. component-wise. We will use this facility when generating the Markov chain for our example telephone system.

The notion of weak Markovian bisimulation [23] is obtained in the following way. For action-transitions we replace $\longrightarrow$ by $\Longrightarrow$ as for the untimed case. The treatment of timed transitions is a bit more involved. First, remark that the probability distribution of a sequence of exponential distributions is not exponential (but constitutes a phase-type distribution). Therefore, it is not possible to define a weak version of the transition relation $\longmapsto$. The solution is to demand that timed transitions have to be bisimulated in the strong sense, while they can be preceded and/or followed by arbitrary sequences of internal action-transitions. The treatment of sequences of internal actions preceding a timed transition can follow the usual style (cf. Section 2), but with a slight exception. (The treatment is similar to that of branching bisimulation [17].) Since the probability that a continuously distributed duration finishes immediately (i.e. at time zero) is zero, whereas the probability for an internal action-transition to take place immediately is one, there is no need to require equality of cumulated rates for states that have an outgoing **i**-transition. Stochastically speaking, these states have a zero sojourn time since they are immediately left using an internal move. Let $P\overset{\mathbf{i}}{\not\longrightarrow}$ be a predicate that is true if and only if P has no outgoing **i**-transitions. Then, we define

Definition 5 (*Weak Markovian bisimulation*)**.** An equivalence relation $\mathscr{S}$ on $\mathscr{L}'$ is a *weak Markovian bisimulation* iff for any pair $(P,Q) \in \mathscr{L}' \times \mathscr{L}'$ we have that $(P,Q) \in \mathscr{S}$ implies for all actions a and all equivalence classes $C \in \mathscr{L}'/_{\mathscr{S}}$:

(1) $\gamma_{\mathrm{w}}(P,a,C) = \gamma_{\mathrm{w}}(Q,a,C)$, and

(2) $P \overset{\mathbf{i}}{\Longrightarrow} P' \overset{\mathbf{i}}{\not\longrightarrow}$ implies that $\exists Q'.\, Q \overset{\mathbf{i}}{\Longrightarrow} Q' \overset{\mathbf{i}}{\not\longrightarrow}$ and $\gamma_{\mathrm{m}}(P',C) = \gamma_{\mathrm{m}}(Q',C)$

Processes P and Q are *weakly Markovian bisimilar*, denoted $P \approx_{\mathrm{m}} Q$, if $(P,Q) \in \mathscr{S}$ with $\mathscr{S}$ a weak Markovian bisimulation.

Weak Markovian bisimulation is a congruence with respect to our language operators (except for the choice operator, for which the usual refinement applies [34].). It inherits the correspondence to lumpability from strong Markovian bisimilarity.

3.4. Generation and aggregation of Markov chains using process algebra

Starting from a system specification in our stochastic process algebra we can generate and aggregate a (homogeneous, continuous-time) Markov chain in the following way. Assuming that the specification S contains all components that are relevant for the complete system we abstract from all interactions A, thus yielding **hide** A **in** S. The interactive Markov chain that is obtained from our semantics – that due to the abstraction only contains timed- and internal action-transitions – is subsequently aggregated using $\approx_{\mathrm{m}}$. Since $\approx_{\mathrm{m}}$ is a congruence we can perform this generation and aggregation of the interactive Markov chain component-wise. We will see that in our telephone example this property turns out to be crucial since the entire system is too large to handle as a whole.

If the resulting transition system is free from non-determinism (like for our example telephone system) we have obtained a (lumped) Markov chain that can be analysed using the traditional techniques to study its transient or steady-state behaviour [40]. If not, the non-determinism needs to be resolved and the procedure must be repeated. We like to stress that the presence of non-determinism should not be considered as an artifact, it reflects that the system specification is not detailed enough (for analysing its performance). Stated otherwise, it indicates that the specification is too abstract – which for design purposes might be even desired – since it contains some under-specification. Technically speaking, non-determinism can for example arise from explicit non-deterministic choices (like $a;\, P\,[]\,a;\, Q$) or implicit ones (like $(a;\, P\,|||\,a;\, Q)\,|[a]|\,a;\, R$). There are several techniques to resolve (read: probabilise) non-determinism, like schedulers, weights and so on, but it is beyond the scope of this paper to further discuss these approaches.

3.5. *Algorithmic and tool support*

The computation of strong and weak Markovian bisimulation can be performed using adaptations of existing algorithms for computing (ordinary) weak and strong bisimulation without increasing their worst-case time and space complexity [22]. Strong Markovian bisimulation is computed using a modification of the partition refinement algorithm of [37] and has a time complexity of $\mathcal{O}(m \log n)$, where n denotes the number of states and m the cardinality of the transition relation. Weak Markovian bisimulation is determined using an adaptation of the algorithm of [5]. Its time complexity is $\mathcal{O}(n^3)$. For the telephone system example we use the implementation of these algorithms in the TIPPTOOL [25].

4. The plain-old telephone system

In this section we informally describe the functional behaviour of the plain-old telephone system (POTS) and provide fragments of its formal specification in Basic LOTOS. Telephone systems have traditionally been a standard specification exercise in the context of LOTOS, see for instance [11, 10]. It is not the intention of our case study to develop yet another specification of the POTS. Instead, we start our study from an existing specification of the POTS that has been developed by Ernberg at the Swedish Institute of Computer Science, and tailor it to our purposes. This specification only describes the functional aspects; in the next section we exemplify how timing constraints are imposed on this specification in a modular way.

4.1. *Informal problem description*

The POTS is a model of the behaviour of classical telephony systems, often used as a basis for studying the consequences of the introduction of new telephony services. It describes the provision of support for physical analogue telephone, fax and modem devices over a telephone line. The system structure is simple, as depicted in Fig. 1. Several users are connected to a telephony network and they can phone each other using the service offered by this network.

The controlling unit *Provider* is responsible for establishing a connection between the originator and the recipient of a call. It handles various signals, like:

- ringing the *bell* on the arrival of a call,
- indicating a free line by means of a 'dial tone' (*dialT*) when a (registered) user has picked up the phone,
- indicating the originator by means of a 'busy tone' (*busyT*) that the recipient's phone is currently off hook,
- indicating the originator by means of a 'ring tone' (*ringT*) that the bell is ringing at the recipient's side,

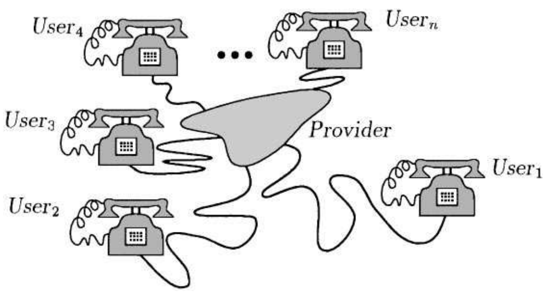

Fig. 1. A plain-old telephone system.

- indicating the originator by means of an ‘error tone’ ($errorT$) that it is necessary to hang up the phone because either an unregistered number has been dialled, or the connection has been interrupted by the partner.

4.2. Structure of the specification

The structure of a formal specification of the POTS reflects the entities shown in Fig. 1. The users $User_1, \ldots, User_n$ act mutually independent from each other, while interaction between them is coordinated by the *Provider*. Each $User_i$ has a private set of communication signals A_i to interact with the provider, such as $dialT\_on_i$ and $dialT\_off_i$ to control the status of the dial tone in the user’s phone. Among others, this set comprises $\_on_i$ and $\_off_i$ signals for the *bell*, for *dialT*one, *ringT*one, *errorT*one, and *busyT*one. For a fixed number n of subscribers, the POTS is specified by

$$POTS := (User_1 \,|||\, \ldots \,|||\, User_n)\,|[A_1, \ldots, A_n]|\, Provider$$

4.3. Provider specification

The main complexity of the POTS example lies in its controlling unit *Provider*. This component has to keep track of the current status of each of the subscribers, and has to establish and release connections. To ensure that all these duties are properly managed is quite challenging. Ernberg has given a specification of *Provider* consisting of more than thousand lines of full LOTOS. This specification has been used as a common example during the Eucalyptus project, a European/Canadian project that focussed on the elaboration of a toolset for LOTOS. During this project, 17 requirements have been formulated to be fulfilled by Ernberg’s specification. Garavel and Mounier have shown that these requirements are indeed satisfied by the specification, using different techniques such as equivalence checking and model checking, that are implemented in the Eucalyptus/CADP toolset [12, 6].

The details of this large specification are beyond the scope of this paper. For the purpose of compositional Markov chain generation, it is sufficient to know that the full LOTOS specification has been extensively verified, and that it can be mapped to a

Basic LOTOS description (where data types are absent) if the number of subscribers is kept fixed. In the sequel we will consider a system of two subscribers, that are aiming to phone each other. This restriction is sufficient to illustrate the key ideas of compositional Markov chain generation.

4.4. User specification

The basic model of user interaction we consider is depicted in Fig. 2 for $User_i$ (the index i is omitted to enhance readability). The state marked ⊙ denotes the initial state of the transition system. For convenience, some states are labelled with names for reference purposes.

We describe some details of the specification here that ease the understanding of the example. Initially, the user's phone is on hook. There are three possibilities to proceed from this initial state *User*.

- the user may either notice that the bell starts ringing (*bell_on*), or
- may decide to pick up the phone (*offH_out*) in order to initiate an outgoing call, or

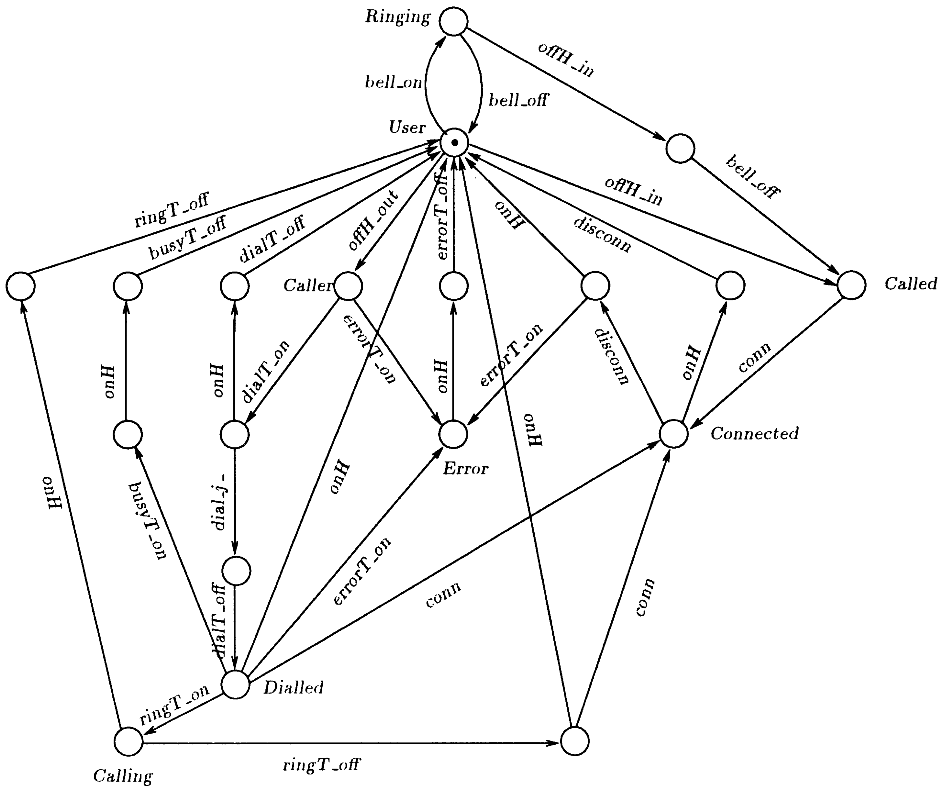

Fig. 2. Specification of the user's interaction possibilities.

- may pick up the phone exactly at the time instant at which an incoming call reaches the phone (*offH_in*), but just before the bell starts ringing.

This behaviour is formally specified by

$$User := offH\_out; Caller \, [] \, bell\_on; Ringing \, [] \, offH\_in; Called$$

Let us first discuss what happens if an incoming call arrives. When the bell starts ringing, the phone enters a state *Ringing*. Subsequently, the bell turns off (*bell_off*) either after the user picks up the phone (*offH_in*), or immediately because no reaction of the user occurs in time.

$$Ringing := bell\_off; User \, [] \, offH\_in; bell\_off; Called$$

In the former case (state *Called*), the connection to the caller is established (*conn*) and the conversation can start, leading to the state *Connected*.

$$Called := conn; Connected$$

Conversation may go on until either the user hangs up (*onH*) and the connection is released (*disconn*), or a disconnection is caused by the opposite side (or by the service provider). In this case, the user may either notice an error tone (*errorT_on*), or he may simply hang up (*onH*).

$$\begin{aligned} Connected := {} & onH; disconn; User \, [] \\ & disconn; (onH; User \, [] \, errorT\_on; Error) \end{aligned}$$

If initially the user acts as a caller, he picks up the phone (*offH_out*), reaching a state *Caller*. Under the assumption that no error occurs, the dial tone is received (*dialT_on*), after which the user may decide to hang up (*onH*) and subsequently return to its initial state (after *dialT_off*). In the successful scenario, the user dials a number j (*dial_j_*), waits until the dial tone ceases (*dialT_off*), and reaches the state *Dialled*.

$$\begin{aligned} Caller := {} & errorT\_on; Error \, [] \\ & dialT\_on; (dial\_j\_; dialT\_off; Dialled \, [] \\ & \qquad onH; dialT\_off; User) \end{aligned}$$

If a ring tone now occurs, the state *Calling* is entered. We refer to Fig. 2 for further details of any subsequent behaviour.

5. Constraint-oriented timing information

In this section we illustrate how stochastic timing information can be added to the POTS specification in a modular, constraint-oriented style. To enhance specification convenience, we introduce an auxiliary operator, the *elapse* operator, that is used to impose phase-type distributed time constraints on specific occurrences of actions.

Phase-type distributions can be considered as matrix generalisations of exponential distributions, they include frequently used distributions such as Erlang, Cox, hyper- and hypo-exponential distributions. The elapse operator facilitates the description of such time constraints in a *modular* way, that is, as separated processes that are composed in parallel with the untimed specification.

After introducing phase-type distributions (itself requiring some basic material on continuous-time Markov chains) we provide the syntax and semantics of the elapse operator and show how we extend the LOTOS specification of the POTS with phase-type time constraints in a modular way.

5.1. Continuous-time Markov chains

A (discrete space) stochastic process is a collection of random variables $\{X(t) \mid t \in T\}$ where $X(t)$ assigns probabilities to elements of a discrete set S of states, the state space. If the set T (usually called the *time range*) is a continuous domain, the stochastic process is referred to as continuous-time. A continuous-time Markov chain is a continuous-time stochastic process that satisfies the Markov property: for each sequence of time instances $t_{n+1} > t_n > t_{n-1} > \cdots > t_0$ (of arbitrary length n), we have

$$\begin{aligned} &Prob\{X(t_{n+1}) = P_{n+1} \mid X(t_n) = P_n,\ X(t_{n-1}) = P_{n-1}, \ldots, X(t_0) = P_0\} \\ &\quad = Prob\{X(t_{n+1}) = P_{n+1} \mid X(t_n) = P_n\}. \end{aligned}$$

Thus, the fact that the chain was in state P_{n-1} at time t_{n-1}, in state P_{n-2} at time t_{n-2}, and so on, up to the fact that it was in state P_0 at time t_0 is completely irrelevant. The probability distribution on states at time t_n, given by $X(t_n)$, contains all relevant history information to determine the distribution on S at time t_{n+1}.

For each state of a continuous-time Markov chain there is some rate λ representing the distribution of the sojourn time for this state, which, in fact, turns out to be an exponential distribution. Furthermore, a continuous-time Markov chain is completely characterised by its generator matrix and its initial distribution. The entries of the generator matrix specify the rates of moving from a certain state in the chain to another state. The initial distribution specifies the probability of starting in a certain state. More precisely,

Definition 6 (*Generator matrix*). A square matrix $\mathbf{Q}$ is the (infinitesimal) generator matrix of a continuous-time Markov chain iff, for all i, $\mathbf{Q}(i,j) \geqslant 0$ $(j \neq i)$, and $\mathbf{Q}(i,i) = -\sum_{j \neq i} \mathbf{Q}(i,j)$.

The states of a continuous-time Markov chain can be classified into recurrent, transient and absorbing states. A state P_i is said to be transient if there is a positive probability of never returning to that state after leaving it. A recurrent state is revisited (in a finite amount of time) with probability 1. An absorbing state is a state without any outgoing transition, i.e. $\mathbf{Q}(i,j) = 0$, for all j.

5.2. Phase-type distributions

A phase-type distribution is defined by means of a continuous-time Markov chain with an absorbing state. The time until absorption determines the phase-type distribution [36].

Definition 7 (*Phase-type distribution*). Let

$$\mathbf{Q} = \begin{pmatrix} \mathbf{T} & \underline{T^0} \\ \underline{0} & 0 \end{pmatrix}$$

be the (infinitesimal) generator matrix of a continuous-time Markov chain with transient states $\{1,\dots,m\}$ and absorbing state $m{+}1$, and initial probability vector $(\underline{\alpha}, \alpha_{m+1})$ with $\underline{\alpha}\underline{1} + \alpha_{m+1} = 1$ such that the row sums of $\mathbf{Q}$ equal 0, i.e. $\mathbf{T}\underline{1} + \underline{T^0} = \underline{0}$. The phase-type distribution defined by this Markov chain is defined by[7]

$$F(x) = 1 - \underline{\alpha}\mathrm{e}^{\mathbf{T}x}\underline{1}$$

for $x \geqslant 0$, and $F(x) = 0$, for $x < 0$.

$\mathbf{T}$ is a square matrix of order m such that $\mathbf{T}(i,i) < 0$ and $\mathbf{T}(i,j) \geqslant 0$ $(i \neq j)$. The row sums of $\mathbf{Q}$ equal zero, i.e. $\mathbf{T}\underline{1} + \underline{T^0} = \underline{0}$. $\mathbf{T}(i,j)$ $(i \neq j)$ can be interpreted as the rate at which the chain changes from transient state i to transient state j. Stated otherwise, starting from state i it takes an exponentially distributed time with mean $1/\mathbf{T}(i,j)$ to reach state j. $\underline{T^0}(i)$ is the rate at which the system can move from transient state i to the absorbing state, state $m{+}1$. $-\mathbf{T}(i,i)$ is the total rate of departure from state i. Notice that for $m = 1$ the above definition boils down to an exponential distribution, highlighting that phase-type distributions can be considered as matrix generalisations of exponential distributions.

The moments μ_i of phase-type distribution $F(x)$ are finite and given by

$$\mu_i = (-1)^i i!(\underline{\alpha}\mathbf{T}^{-i}\underline{1}) \quad \text{for } i = 1,2,\dots$$

μ_1 is the expectation, $\mu_2 - \mu_1^2$ the variance, and the fraction of the expectation over the standard deviation (the square root of the variance) is called the coefficient of variation, CV.

5.3. Algebraic specification of phase-types

In terms of our language, phase-type distributions can be described by successfully terminating processes that do not contain any action-prefix. The occurrence of successful termination action δ signals the time instant of finishing. For example, the process

$$PH := (\lambda);\ ((\mu);\ \mathbf{exit}\,[]\,(\kappa);\ (\nu);\ (\nu);\ (\nu);\ \mathbf{exit})$$

[7] For square matrix $\mathbf{T}$ of order m, $\mathrm{e}^{\mathbf{T}x}$ is defined by $\mathrm{e}^{\mathbf{T}x} = \sum_{k=0}^{\infty} \mathbf{T}^k x^k / k!$.

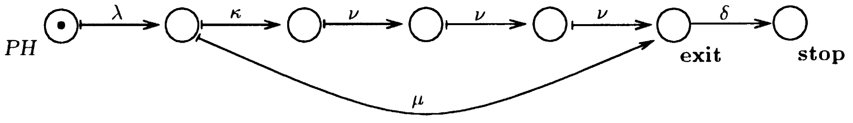

Fig. 3. Transition system of an absorbing Markov chain.

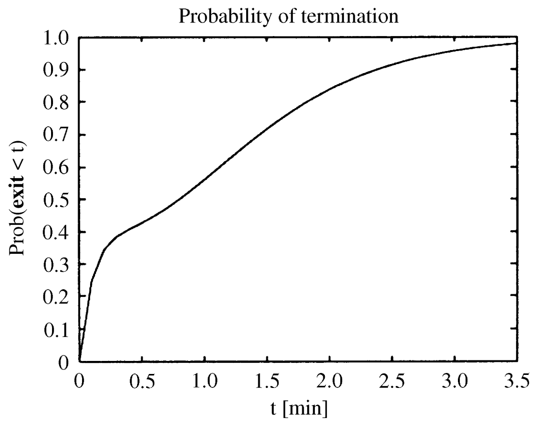

Fig. 4. Probability distribution described by PH $(\lambda = 10, \mu = 100, \kappa = 150, \nu = 2)$.

describes the absorbing Markov chain of six states that is depicted in Fig. 3. The distribution of the time until absorption is plotted in Fig. 4, for a specific choice of the parameters λ, μ, κ, and ν. In the sequel we will refer to a process that has the potential of successful termination, and where action-prefix does not occur, as an absorbing Markov chain. The possibility of specifying phase-type distributions is of significant interest, since phase-type distributions can approximate arbitrary distributions arbitrarily closely [36]. In other words, we can impose an arbitrarily distributed time constraint, by choosing the appropriate absorbing Markov chain.

5.4. Syntax and informal semantics of elapse

To fix terminology, we will refer to a time constraint as a delay that necessarily has to elapse between two kinds of interactions, unless some interaction of a third kind occurs in the meanwhile. In order to facilitate the definition of such time constraints the elapse operator, syntactically denoted by [**on** s **delay** d **by** $Q \leftarrow\!\!\!\sim b$], is an operator with four parameters:

- an absorbing Markov chain Q that determines the duration of the time constraint,
- an action s (start) that determines when the delay (governed by Q) starts,
- an action d (delay) that has to be delayed, and
- an action b (break) that may interrupt the delay.

The intuition behind this operator is that it enriches the chain Q with some synchronisation potential that is used to initialise and reset the time constraint in an appropriate way. The time constraint is imposed on a process P by means of parallel composition, such as

$$P\,|[s,d,b]|\,[\mathbf{on}\ s\ \mathbf{delay}\ d\ \mathbf{by}\ Q \leftsquigarrow b]$$

where the action d is delayed by Q inside P.

5.5. Semantics of elapse

The elapse operator is an auxiliary operator. Hence, we can provide a semantics by means of a translation into the basic operators.

Definition 8 (*Elapse operator*). For a given absorbing Markov chain Q $[\mathbf{on}\ s\ \mathbf{delay}\ d\ \mathbf{by}\ Q \leftsquigarrow b]$ is defined as the process $\hat{Q}$, where

$$\hat{Q} := (d;\ \mathbf{exit}\,[]\,s;\ (Q \gg d;\ \mathbf{exit})\ [> b;\ \mathbf{exit}) \gg \hat{Q}$$

For the above example, we have depicted in Fig. 5 the interactive Markov chain that corresponds to $[\mathbf{on}\ s\ \mathbf{delay}\ d\ \mathbf{by}\ PH \leftsquigarrow b]$. For illustration purposes, the transition system is factorised with respect to weak Markovian bisimulation ($\approx_{\mathrm{m}}$). In the initial state, actions s, d, and b are possible, but only s induces a change of state, starting the phase-type distributed delay. Once the delay started, action d is no longer possible, until either the delay has elapsed completely, or an interrupting action b has occured, resetting the delay.

We use the elapse operator in a constraint-oriented fashion [41], by synchronising it with the system under investigation. Let us, for example, consider the POTS specification and assume $User_1$ has successfully dialled a number (i.e. consider state *Dialled*). Suppose we want to constrain the time until an impatient caller may put down the phone in case the called party is not answering the call. If we assume this phase-type

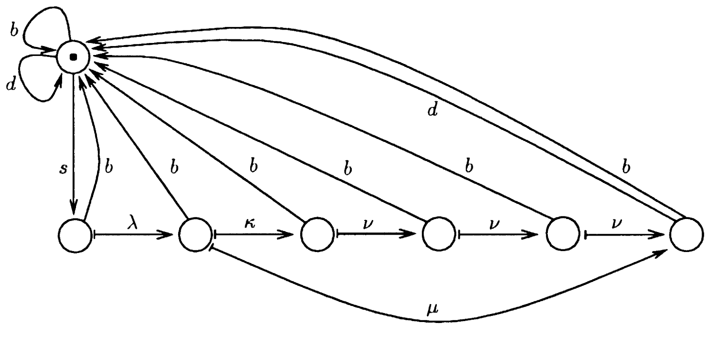

Fig. 5. Effect of the elapse operator on PH (modulo weak Markovian bisimulation).

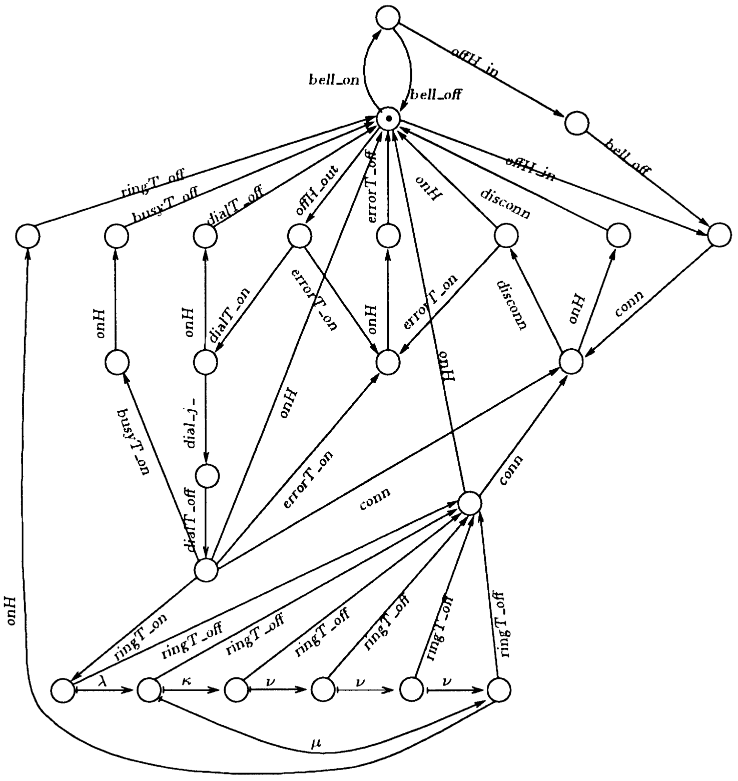

Fig. 6. A time constraint for $User_1$, governed by PH (modulo weak Markovian bisimulation).

distribution to be governed by PH, we can specify this time constraint by

$$User_1 \,|[A]|\, [\textbf{on } ringT\_on_1 \textbf{ delay } onH_1 \textbf{ by } PH \twoheadleftarrow ringT\_off_1]$$

where $A = \{ringT\_on_1, ringT\_off_1, onH_l\}$. Starting from $ringT\_on_1$, the action onH_1 is prohibited for a time governed by PH, with the exception that the delay is interrupted if the ring tone ceases ($ringT\_off_1$), indicating a timely reaction of the called party. The corresponding interactive Markov chain for the user is depicted in Fig. 6.

The elapse operator can be easily adapted to situations where the starting, delayed and interrupting actions are sets rather than singletons, basically by replacing the action-prefixes in the defining equation of $\hat{Q}$ by sums of prefixes [22].

5.6. Time constraints for the POTS

In order to incorporate stochastic time into the POTS model, we incorporate several delays that have to elapse between certain interactions. There is, of course, a variety of different choices where to introduce time constraints. We add the following 14 time constraints to the specification.

- Whenever an existing connection is released, the error tone is raised after a while, except if the respective user hangs up the phone in the meanwhile. The delay is governed by *ErrorDelay*, and is imposed for $User_1$, as well as $User_2$.
- After a user has picked up the phone in order to make a call, it requires some time (to check whether the user is registered, for instance) before some tone is raised. The corresponding time constraint is governed by *InitDelay*, for both $User_1$ and $User_2$.
- Establishing a connection (or detecting that no connection can be established) requires some time. Thus we impose a delay *DialDelay* between dialling a number and getting a reaction from the provider's side, symmetrically for both users.
- When noticing a ring tone, $User_1$ waits for a connection. He may hang up the phone after a while, except if a connection has been established in the meanwhile. The time until hanging up the phone is determined by $WaitDelay_1$. For $User_2$, a time constraint governed by $WaitDelay_2$ is imposed in the same manner.
- Between two phone-calls $User_1$ takes a rest, but if the phone rings, he decides to go and pick up the phone. We let $IdleDelay_1$ govern the time that elapses between two phone-calls originated by $User_1$. For $User_2$, the situation is symmetric, but the delay is given by $IdleDelay_2$. Since both phones are initially on hook, we initialise the corresponding time constraint such that both users have to wait before they may pick up the phone.
- If the phone bell starts ringing, $User_1$ needs a certain time to reach the phone, given by $PickupDelay_1$. For $User_2$, the corresponding constraint is governed by $PickupDelay_2$.
- Once a connection is established, $User_1$ contributes to the conversation a certain amount of time, but a disconnection may occur anyway in the meanwhile. The corresponding delay is imposed by a time constraint governed by $SpeakDelay_1$ for $User_1$, and similar for $User_2$.

This summarises the major time constraints we assume to exist. Note that the time constraints that are due to the provider (*InitDelay*, *DialDelay*, and *ErrorDelay*) are identical to both users. So, the provider is assumed to be fair and not to prefer either of the users. On the other hand, the time constraints that are caused by the users ($SpeakDelay_i$, $IdleDelay_i$, $PickupDelay_i$, and $WaitDelay_i$) are parametric in the identity of the user. This makes it possible to incorporate different user profiles.

The complete specification of the time-constrained POTS is given in the appendix. For our purpose, it is sufficient to sketch the structure of the specification, where *TC* summarises the constraints: [8]

$$TimedPOTS := ((User_1 \,|||\, User_2)\,|[A_1, A2]|\, Provider)\,|[\ldots]|\, TC$$

[8] Owing to some specific associativity laws enjoyed by parallel composition [3], the time constraints can appear at the outermost level. Note, however, that parallel composition, in general, is not associative.

Table 4
Stochastic time delays imposed on the POTS

	$User_1$	$User_2$	CV	Type of distribution
ErrorDelay	15 s	15 s	1	Exponential
InitDelay	2 s	2 s	1	Exponential
DialDelay	6 s	6 s	0.408	Erlang_6
WaitDelay	60 s	30 s	1	Exponential
SpeakDelay	15 min	5 min	1.5	Phase-type
IdleDelay	60 min	75 min	0.316	Erlang_{10}
PickupDelay	40 s	10 s	1	Exponential

5.7. Time values

Table 4 shows the time values we have used for the numerical analysis of the telephony system specification. The table lists the mean durations (first two columns), the coefficient of variation of the distribution (third column) and the distribution function (last column).

The profiles of $User_1$ and $User_2$ differ with respect to their time values in a particular manner. We assume that $User_2$ is generally reacting quicker than $User_1$, while $User_1$ likes to phone more often and longer than $User_2$. The majority of delays simply consist of a single exponential phase. *DialDelay* describes an Erlang distribution (a sequence of equally rated exponential phases), because we assume that the time required to establish a connection should not exhibit too much variance. On the other hand, the time of conversation may vary a lot, and the variance of the respective distribution, given by $SpeakDelay_i$, should therefore be rather high. We use a phase-type distribution similar to PH for this purpose. The delays that are imposed between two successive phone-calls, $IdleDelay_i$, are assumed to be governed by an Erlang distribution with fairly low variances, consisting of ten exponential phases each.

6. Analysing the POTS

In this section we describe how we manage to analyse the POTS by means of compositional generation of the Markov chain and present the numerical results from the transient analysis of the behaviour.

6.1. State space of the system

The specification we want to analyse is quite complex. Therefore, the state space, obtained by means of the operational semantics, is rather large. It consists of more than 10 million states. Indeed we are not even able to generate the complete state space of the specification at all. The tools we are employing, TIPPTOOL and CÆSAR ran out of memory after generating 0.3, respectively 10.2 million states (on a SUN Ultra-1 with 512 MB main memory).

6.2. Compositional generation

In order to circumvent the state space explosion problem, we generate the state space in a compositional way. We produce the state space component-wise, and perform an aggregation on components. The aggregation is based on building the quotient (i.e. the equivalence classes) with respect to a bisimulation. The congruence property of (strong and weak) Markovian bisimulation justifies to use aggregated components instead of the original ones in the context of further composition.

In order to perform this aggregation efficiently, we apply the operational semantics rules to the non-constrained specification, *POTS*. Since this specification is a Basic LOTOS expression, strong and weak Markovian bisimulation ($\sim_m$ respectively $\approx_m$) boil down to ordinary strong and weak bisimulation ($\sim$ respectively $\approx$). We therefore rely on the efficient algorithms provided by the CADP tool-set to generate and aggregate the transition system. The transition system generated from this specification turns out to be smaller by one order of magnitude, compared to the time-constrained specification. It consists of 1 040 529 states and 2 527 346 transitions. Using ALDÉBARAN we compute the quotient of this transition system with respect to $\sim$ (in about four minutes), resulting in 327 states only. Note that we can rely on the algorithms for ordinary bisimulation, since neither $User_i$ nor *Provider* gives rise to any timed transitions. This is due to the use of a constraint-oriented style, where time delays are specified as separate processes.

The reason for the enormous reduction of state space essentially originates from the fact that Ernberg's LOTOS specification of *Provider* uses a variety of data variables to keep track of the current status of each individual subscriber. The LOTOS semantics produces different states for every (reachable) combination of actual variable assignments, as it is the case in a real implementation. However, from a behavioural point of view, many states are equivalent, since they exhibit the same behaviour. These states are therefore equated by $\sim$.

Let $POTS_{\mathrm{min}}$ denote (an expression corresponding to) the quotient transition system that we obtained with ALDÉBARAN:

$$POTS \sim POTS_{\mathrm{min}}$$

Due to the congruence property (and the fact that $\sim \;\subseteq\; \sim_m$), our time-constrained specification *TimedPOTS* is bisimilar to the specification obtained by imposing the same time constraints on $POTS_{\mathrm{min}}$, i.e.

$$TimedPOTS \sim_m POTS_{\mathrm{min}} \;|[\ldots]|\; TC$$

We continue with investigating the latter specification. Since *TC* consists of various parallel components, we may compositionally construct some TC_{min} in a similar modular way (using $\sim_m$). We do not work out the details. The interactive Markov chain of

$$POTS_{\mathrm{min}} \,|[\ldots]|\, TC_{\mathrm{min}}$$

has 3215 states and 10 421 transitions, instead of more than 10^7 states, as for the original, yet bisimilar, specification *TimedPOTS*. We base the analysis of the specification on this interactive Markov chain. In order to analyse the timing dependent behaviour, we aim to transform the system into a Markov chain. As outlined in Section 3, this requires abstraction from all interactions, yielding

$$\mathbf{hide}\ A_1, A_2\ \mathbf{in}\ POTS_{\mathrm{min}} \,|[\ldots]|\, TC_{\mathrm{min}}$$

which is essentially the above interactive Markov chain, with all action labels renamed into **i**. Its state space, in turn, can be aggregated once again using $\approx_{\mathrm{m}}$ to a minimal representation consisting of 720 states, say $TimedPOTS_{\mathrm{min}}$. This computation using TIPPTOOL lasts about 30 min.

So, in order to analyse the behaviour of the time-constrained POTS specification, we have to analyse an interactive Markov chain with 720 states. This chain does not contain any non-determinism. As a consequence, we have obtained a (lumped) Markov chain. It is worth to remark that this Markov chain has a highly irregular shape.

6.3. Transient analysis of the POTS

Analysis of the lumped Markov chain relies on the computation of state probabilities of each of the 720 states. We can compute transient state probabilities, in order to obtain the time-dependent probabilities $\pi_P(t)$ for each of the states P of the Markov chain at a given time instant t. This analysis requires the solution of an ordinary differential equation system. Alternatively, we can compute the steady state probability distribution, since the lumped Markov chain is ergodic [40]. As a result, we obtain the probabilities π_P that the system is in a certain state P, assuming that an equilibrium has been reached. Steady state analysis requires the solution of a linear equation system. Efficient algorithms are known for steady state as well as transient analysis [39].

State probabilities are the starting point to obtain more general insight into the behaviour of a system. They form the basis of a wide class of performance measures that appear (in the simplest cases) as weighted sums of such probabilities. To obtain a particular measure each state P is equipped with a *reward* $\Re(P)$ provided by a real-valued reward function $\Re$.

For instance, we can isolate the states where $User_1$ is actually using his phone by assigning a reward $\Re(P) = 1$ to those states P where $User_1$ is able to put down the phone (action onH_1) while all other states obtain reward 0. A measure of interest, the probability (at a certain time instant t) that $User_1$ has actually picked up the phone, then arises as the weighted sum over all states P.

$$\sum \pi_P(t)\, \Re(P).$$

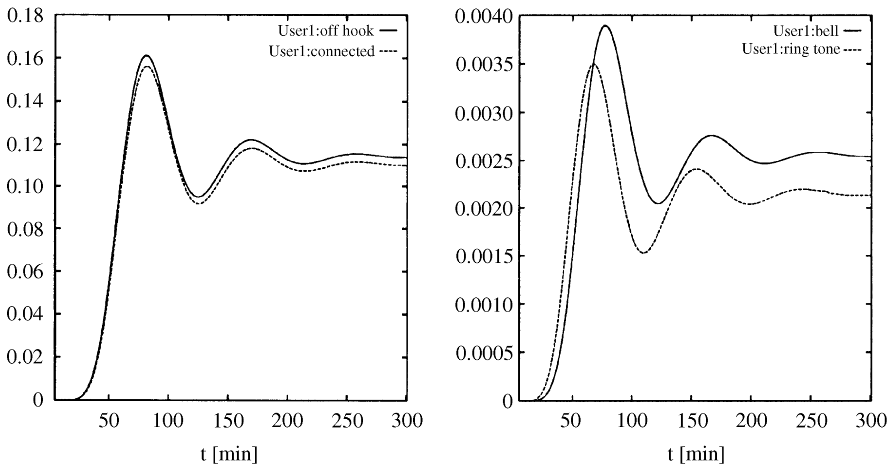

Fig. 7. Transient behaviour of the POTS. *Left:* Probability of being off hook and of being connected for $User_1$. *Right:* Probability of ringing bell and ring tone at $User_1$.

In the context of this case study we have computed measures of interest using similar reward functions. More complicated reward functions are known, that are applicable to describe advanced performance, performability and dependability aspects [21].

In Fig. 7 we have depicted some results obtained by means of transient analysis. The figure shows how the system behaviour converges to an equilibrium as time progresses. To achieve these plots we have iteratively calculated the state probabilities every 2 min of system life time, using the TIPPTOOL. The state probabilities are cumulated by means of appropriate reward functions.

The plot on the left indeed has been achieved using the reward function described above. It shows the probability that $User_1$ has currently picked up the phone, together with the probability that a speech connection is actually established, which is slightly less probable. The shape of these plots is mostly governed by our choice of *IdleDelay*s. The time constraints controlled by these delays initially prohibits that either of the users may pick up the phone, until some *IdleDelay* has elapsed. If this has occurred (given by a superposition of two $Erlang_{10}$ distributions) the probabilities depicted in Fig. 7 raise. A first peak is reached after approximately 77 min. The probabilities oscillate for a while, because idle phases and connection phases alternate. An equilibrium is reached, due to the stochastic perturbation caused by the distributions, especially the phase-type distributed *SpeakDelay*s.

The right plot in Fig. 7 shows the probability that $User_1$ is noticing a ring tone, respectively a ringing bell. The probabilities are fairly small compared to the ones in Fig. 7. The probability of a ringing bell increases later than that of a ring tone. This is a consequence of the fact that $User_1$ (in the mean) is calling $User_2$ earlier than the other way around. The first attempt to call $User_2$ occurs after roughly 60 min. The opposite direction is delayed for about 75 min.

In summary, the influence of the time that elapses between two phone calls is decisive for the transient behaviour of the system. If we change the distribution of *IdleDelay* such that the variance is increased, the oscillation is flattened. On the other hand, equilibrium probabilities are not affected significantly. To study the effect of different distributions is quite easy, due to the constraint-oriented specification style used for time constraints. Of course, employing different distributions for a time constraint requires that the state spaces have to be generated again (except from $POTS_{\min}$), since different distributions lead to different state spaces. For instance, changing both *IdleDelay*s to single exponential phases leads to a lumped Markov chain of 351 states.

7. Concluding remarks

In this paper we presented a case study in which we analysed the performance of a plain-old telephone system (POTS) using a non-traditional performance technique. Starting from a process algebra description of the POTS in LOTOS which we extended with time constraints, a continuous-time Markov chain is generated and minimised using the formal apparatus – in particular, semantics and equivalence checking – of process algebra.

To specify the stochastic time constraints we introduced an extension of Basic LOTOS with the time-prefix $(\lambda); P$. The semantics of the resulting stochastic process algebra is an orthogonal extension of the standard LOTOS semantics, and of (homogeneous) continuous-time Markov chains. The explicit separation between phases in which actions and their state transitions occur, but no time passes, and phases in which time advances, but no actions appear, naturally supports an intuitively clear semantics for synchronisation (of rates). The auxiliary elapse operator eases the specification of phase-type durations in a modular way.

It has not been our intention to carry out an extensive performance analysis of the POTS. Although the generated Markov chain allows several (traditional) kinds of analyses, we just presented some transient analysis. Instead, we focussed on the use of a stochastic process algebra for the generation and minimisation of a Markov chain. We like to make the following remarks concerning our experience with respect to this issue. First, the resulting aggregated Markov chain (of 720 states) is highly irregular, whereas the formal specification of the timed POTS is rather well-structured – its compositional structure reflects the system structure and time constraints are added in a constraint-oriented way. We do believe that such irregular minimised Markov chains can hardly be obtained by applying any of the standard techniques for obtaining such models. Secondly, our case study has shown that an aggregated Markov chain can be obtained in an almost fully automated way, starting from an existing LOTOS specification. The only step in which we used ‘intelligence’ has been the decision on how to decompose the timed POTS specification. Due to its large size (more than 10^7 states) we were not able to apply the bisimulation algorithms on the specification as a whole. Instead, we exploited compositionality and applied these algorithms component-wise. This required to break down the specification into pieces of moderate size. The constraint-oriented character of the elapse operator has been of crucial importance here. For aggregation purposes we applied ordinary strong and weak bisimulation (using CÆSAR/ALDÉBARAN [12, 6]) and stochastic variants thereof (using TIPPTOOL [25]). Although theoretically the worst-case time and space complexity of these algorithms is the same [22], we experienced that the implementation of ALDÉBARAN outperforms that of TIPPTOOL to a significant extent. With respect to the state space generation, the same is true for CÆSAR compared to TIPPTOOL.

A challenging issue for the future research is to investigate the practical applicability of techniques that support general distributions symbolically, that is, not as encodings of phase-types. Interesting approaches in this direction are the generation of generalised semi-Markov processes [9] and the mapping onto stochastic task graphs [27] which can be formalised using the true concurrency semantics of [7].

Appendix A. Time-constrained POTS

In this appendix we list the detailed time constraints imposed on the POTS specification by means of our elapse operator. The time constraints labelled (†) require a straightforward extension of this operator to sets of actions to be delayed. The last two constraints, labelled (‡) require a similar extension of a variant of the elapse operator, $[\textbf{on'}\ s\ \textbf{delay}\ d\ \textbf{by}\ Q \leftsquigarrow b]$, defined by

$$((Q \gg d;\ \textbf{exit})\,[> b;\ \textbf{exit}) \gg [\textbf{on}\ s\ \textbf{delay}\ d\ \textbf{by}\ Q \leftsquigarrow b]$$

the difference being that the time constraint is initially running.

$TimedPOTS :=$
$(\ldots($
  $POTS$
$|[disconn_1, errorT\_on_1, onH_1]|$
  $[\textbf{on}\ disconn_1\ \textbf{delay}\ errorT\_on_1\ \textbf{by}\ ErrorDelay \leftsquigarrow onH_1])$
$|[disconn_2, errorT\_on_2, onH_2]|$
  $[\textbf{on}\ disconn_2\ \textbf{delay}\ errorT\_on_2\ \textbf{by}\ ErrorDelay \leftsquigarrow onH_2])$
$|[conn_1, onH_1, disconn_1]|$
  $[\textbf{on}\ conn_1\ \textbf{delay}\ onH_1\ \textbf{by}\ SpeakDelay \leftsquigarrow disconn_1])$
$|[conn_2, onH_2, disconn_2]|$
  $[\textbf{on}\ conn_2\ \textbf{delay}\ onH_2\ \textbf{by}\ SpeakDelay \leftsquigarrow disconn_2])$
$|[bell\_on_1, offH\_in_1, bell\_off_1]|$
  $[\textbf{on}\ bell\_on_1\ \textbf{delay}\ offH\_in_1\ \textbf{by}\ PickupDelay_1 \leftsquigarrow bell\_off_1])$
$|[bell\_on_2, offH\_in_2, bell\_off_2]|$
  $[\textbf{on}\ bell\_on_2\ \textbf{delay}\ offH\_in_2\ \textbf{by}\ PickupDelay_2 \leftsquigarrow bell\_off_2])$
$|[ringT\_on_1, onH_1, conn_1]|$
  $[\textbf{on}\ ringT\_on_1\ \textbf{delay}\ onH_1\ \textbf{by}\ WaitDelay_1 \leftsquigarrow conn_1])$

$|[\mathit{ringT\_on_2}, \mathit{onH_2}, \mathit{conn_2}]|$
 [**on** $\mathit{ringT\_on_2}$ **delay** $\mathit{onH_2}$ **by** $\mathit{WaitDelay_2} \leftsquigarrow \mathit{conn_2}$])
$|[\mathit{dial\_2\_{_1}}, A_1, \mathit{onH_1}]|$
 [**on** $\mathit{dial\_2\_{_1}}$ **delay** A_1 **by** $\mathit{DialDelay} \leftsquigarrow \mathit{onH_1}$]) (†)
$|[\mathit{dial\_1\_{_2}}, A_2, \mathit{onH_2}]|$
 [**on** $\mathit{dial\_1\_{_2}}$ **delay** A_2 **by** $\mathit{DialDelay} \leftsquigarrow \mathit{onH_2}$]) (†)
$|[\mathit{offH\_out_1}, B_1, \mathit{onH_1}]|$
 [**on** $\mathit{offH\_out_1}$ **delay** B_1 **by** $\mathit{InitDelay} \leftsquigarrow \mathit{onH_1}$]) (†)
$|[\mathit{offH\_out_2}, B_2, \mathit{onH_2}]|$
 [**on** $\mathit{offH\_out_2}$ **delay** B_2 **by** $\mathit{InitDelay} \leftsquigarrow \mathit{onH_2}$]) (†)
$|[\mathit{onH_1}, C_1, \mathit{bell\_on_1}]|$
 [**on'** $\mathit{onH_1}$ **delay** C_1 **by** $\mathit{IdleDelay_1} \leftsquigarrow \mathit{bell\_on_1}$]) (‡)
$|[\mathit{onH_2}, C_2, \mathit{bell\_on_2}]|$
 [**on'** $\mathit{onH_2}$ **delay** C_2 **by** $\mathit{IdleDelay_2} \leftsquigarrow \mathit{bell\_on_2}$]) (‡)
where

$$A_i = \{\mathit{ringT\_on_i}, \mathit{busyT\_on_i}, \mathit{errorT\_on_i}, \mathit{conn_i}\}$$

$$B_i = \{\mathit{dialT\_on_i}, \mathit{busyT\_on_i}, \mathit{errorT\_on_i}\}$$

$$C_i = \{\mathit{offH\_out_i}, \mathit{offH\_in_i}\}$$

References

[1] M. Bernardo, Using EMPA for the performance evaluation of an ATM switch, in: J.F. Groote, B. Luttik, J. van Wamel (Eds.), Proc. 3rd ERCIM Int. Workshop on Formal Methods for Industrial Critical Systems, SMC Press, 1998, pp. 43–58.

[2] M. Bernardo, R. Gorrieri, Extended Markovian process algebra, in: U. Montanari, V. Sassone (Eds.), Concur'96: Concurrency Theory, Lecture Notes in Computer Science, vol. 1119, Springer, Berlin, 1996, pp. 315–330.

[3] T. Bolognesi, Regrouping parallel processes, J. Formal Methods Systems Des. 9 (1996) 263–302.

[4] T. Bolognesi, E. Brinksma, Introduction to the ISO specification language LOTOS, Comput. Networks ISDN Systems 14 (1987) 25–59.

[5] A. Bouali, Weak and branching bisimulation in FCTOOL, Rapports de Recherche 1575, INRIA Sophia Antipolis, 1992.

[6] M. Bozga, J.-C. Fernandez, A. Kerbrat, L. Mounier, Protocol verification with the Aldébaran toolset, Int. J. Software Tools Techn. Transfer 1(1/2) (1997) 166–184.

[7] E. Brinksma, J.-P. Katoen, R. Langerak, D. Latella, A stochastic causality-based process algebra, Comput. J. 38(7) (1995) 552–565.

[8] G. Chehaibar, H. Garavel, L. Mounier, N. Tawbi, F. Zulian, Specification and verification of the Powerscale bus arbitration protocol: an industrial experiment with LOTOS, in: R. Gotzhein, J. Bredereke (Eds.), Formal Description Techniques IX, Chapman & Hall, London, 1996, pp. 435–451.

[9] P.R. D'Argenio, J.-P. Katoen, E. Brinksma, An algebraic approach to the specification of stochastic systems (extended abstract), in: D. Gries, W.-P. de Roever (Eds.), Programming Concepts and Methods, Chapman & Hall, London, 1998, pp. 126–148.

[10] P. Ernberg, T. Hovander, F. Monfort, Specification and implementation of an ISDN telephone system using LOTOS, in: M. Diaz R. Groz (Eds.), Formal Description Techniques V, North-Holland, Amsterdam, 1992.

[11] M. Faci, L. Logrippo, B. Stepien, Formal specification of telephone systems in LOTOS: the constraint-oriented approach, Comput. Networks ISDN Systems 21 (1991) 53–67.

[12] H. Garavel, Open/Cæsar: an open software architecture for verification, simulation, and testing, in: B. Steffen (Ed.), Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science, vol. 1384, Springer, Berlin, 1998, pp. 68–84.

[13] S. Gilmore, J. Hillston, R. Holton, M. Rettelbach, Specifications in stochastic process algebra for a robot control problem, Int. J. Production Res. 34(4) (1996) 1065–1080.

[14] R.J. van Glabbeek, The linear time — branching time spectrum, in: J.C.M. Baeten, J.-W. Klop (Eds.), Concur'90: Theories of Concurrency — Unification and Extension, Lecture Notes in Computer Science, vol. 458, Springer, Berlin, 1990, pp. 278–297.

[15] R.J. van Glabbeek, The linear time — branching time spectrum II (The semantics of sequential systems with silent moves), in: E. Best (Ed.), Concur'93: Concurrency Theory, Lecture Notes in Computer Science, vol. 715, Springer, Berlin, 1993, pp. 66–81.

[16] R.J. van Glabbeek, S.A. Smolka, B. Steffen, Reactive, generative, and stratified models of probabilistic processes, Inform. and Comput. 121 (1995) 59–80.

[17] R.J. van Glabbeek, W.P. Weijland, Branching time and abstraction in bisimulation semantics, J. ACM 43(3) (1996) 555–600.

[18] J.C. Godskesen, K.G. Larsen, Real-time calculi and expansion theorems, in: R.K. Shyamasundar (Ed.), Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Computer Science, vol. 652, Springer, Berlin, 1992, pp. 302–316.

[19] N. Götz, U. Herzog, M. Rettelbach, Multiprocessor and distributed system design: the integration of functional specification and performance analysis using stochastic process algebras, in: L. Donatiello, R. Nelson (Eds.), Performance Evaluation of Computer and Communication Systems, Lecture Notes in Computer Science, vol. 729, Springer, Berlin, 1993, pp. 121–146.

[20] P.G. Harrison, B. Strulo, Stochastic process algebra for discrete event simulation, in: F. Baccelli, A. Jean-Marie, I. Mitrani (Eds.), Quantitative Methods in Parallel Systems, Springer, Berlin, 1995, pp. 18–37.

[21] B.R. Haverkort, K.S. Trivedi, Specification techniques for Markov reward models, Discrete Ev. Systems: Theory and Appl. 3 (1993) 219–247.

[22] H. Hermanns, Interactive Markov chains, Ph.D. Thesis, University of Erlangen-Nürnberg, 1998.

[23] H. Hermanns, M. Rettelbach, T. Weiss, Formal characterisation of immediate actions in SPA with non-deterministic branching, Comput. J. 38(7) (1995) 530–542.

[24] H. Hermanns, U. Herzog, V. Mertsiotakis, Stochastic process algebras: between LOTOS and Markov chains, Comput. Networks ISDN Systems 9/10 (1998) 901–924.

[25] H. Hermanns, U. Herzog, U. Klehmet, V. Mertsiotakis, M. Siegle, Compositional performance modelling with the TIPPtool, in: R. Puignajer, N.N. Savio, B. Serra (Eds.), Computer Performance Evaluation, Lecture Notes in Computer Science, vol. 1469, Springer, Berlin, 1998, pp. 51–63.

[26] H. Hermanns, M. Rettelbach, Towards a superset of LOTOS for performance prediction, in: M. Ribaudo (Ed.), Proc. 4th Int. Workshop on Process Algebras and Performance Modelling, C.L.U.T. Press, 1996, pp. 77–94.

[27] U. Herzog, A concept for graph-based stochastic process algebras, generally distributed activity times, and hierarchical modelling, in: M. Ribaudo (Ed.), Proc. 4th Int. Workshop on Process Algebra and Performance Modelling, C.L.U.T. Press, 1996, pp. 1–20.

[28] J. Hillston, A Compositional Approach to Performance Modelling, Cambridge University Press, Cambridge, 1996.

[29] J. Hillston, A class of PEPA models exhibiting product form solution over submodels, Tech. Report ECS-LFCS-98-382, Univ. of Edinburgh, 1998.

[30] J. Hillston, V. Mertsiotakis, A simple time scale decomposition technique for stochastic process algebras, Comput. J. 38(7) (1995) 566–578.

[31] P. Inverardi, C. Priami, Automatic verification of distributed systems: the process algebra approach, J. Formal Methods Systems Des. 8 (1996) 1–38.

[32] K.G. Larsen, A. Skou, Bisimulation through probabilistic testing, Inform. and Comput. 94(1) (1992) 1–28.

[33] V. Mertsiotakis, M. Silva, Throughput approximation of decision-free processes using decomposition, in Proc. 7th Int. Workshop on Petri Nets and Performance Models, IEEE CS-Press, 1997, pp. 174–182.

[34] R. Milner, Communication and Concurrency, Prentice-Hall, Englewood Cliffs, NJ, 1989.

[35] F. Moller, C. Tofts, A temporal calculus of communicating systems, in: J.C.M. Baeten, J-W. Klop (Eds.), Concur'90: Theories of Concurrency — Unification and Extension, Lecture Notes in Computer Science, vol. 458, Springer, Berlin, 1990, pp. 401–415.
[36] M.F. Neuts, Matrix-geometric Solutions in Stochastic Models — An Algorithmic Approach, The Johns Hopkins University Press, Baltimore, MD, 1981.
[37] R. Paige, R. Tarjan, Three partition refinement algorithms, SIAM J. Comput. 16(6) (1987) 973–989.
[38] D. Park, Concurrency and automata on infinite sequences, in: P. Deussen (Ed.), Proc. 5th GI Conf., Lecture Notes in Computer Science, vol. 104, Springer, Berlin, 1981, pp. 167–183.
[39] W.J. Stewart, Introduction to the Numerical Solution of Markov Chains, Princeton University Press, Princeton, NJ, 1994.
[40] K.S. Trivedi, Probability and Statistics with Reliability, Queuing, and Computer Science Applications, Prentice-Hall, Englewood Cliffs, NJ, 1982.
[41] C.A. Vissers, G. Scollo, M. van Sinderen, E. Brinksma, On the use of specification styles in the design of distributed systems, Theoret. Comput. Sci. 89(1) (1991) 179–206.
[42] W. Yi, Real-time behaviour of asynchronous agents, in: J.C.M. Baeten, J.-W. Klop (Eds.), Concur'90: Theories of Concurrency — Unification and Extension, Lecture Notes in Computer Science, vol. 458, Springer, Berlin, 1990, pp. 501–520.