ELSEVIER

Performance Evaluation 39 (2000) 5–35

PERFORMANCE EVALUATION
An International Journal

www.elsevier.com/locate/peva

Compositional performance modelling with the TIPPtool

H. Hermanns [*], U. Herzog, U. Klehmet, V. Mertsiotakis, M. Siegle

Lehrstuhl für Informatik VII, Friedrich-Alexander Universität Erlangen–Nürnberg, Martensstr. 3, 91058 Erlangen, Germany

Abstract

Stochastic process algebras have been proposed as compositional specification formalisms for performance models. In this paper, we describe a tool which aims at realising all beneficial aspects of compositional performance modelling, the TIPPtool. It incorporates methods for compositional specification as well as solution, based on state-of-the-art techniques, and wrapped in a user-friendly graphical front end. Apart from highlighting the general benefits of the tool, we also discuss some lessons learned during development and application of the TIPPtool. A non-trivial model of a real life communication system serves as a case study to illustrate benefits and limitations. ©2000 Published by Elsevier Science B.V. All rights reserved.

Keywords: Stochastic process algebra; Performance analysis; Markov chain; Bisimulation aggregation; Tool support

1. Introduction

Process algebras are an advanced concept for the design of distributed systems. From the beginning [39,49], their basic idea was to systematically construct complex systems from small building blocks. Standard operators allow highly modular and hierarchical specification. An algebraic framework supports the comparison of different system specifications, process verification and structured analysis. Classical process algebras such as LOTOS [8], CSP [40] or CCS [50] describe the functional behaviour of systems, but no temporal aspects.

Starting from [33], we developed an integrated design methodology by embedding stochastic features into process algebras, leading to the concept of *stochastic process algebras* (SPA). Since then, research on SPA has been a field of growing activity, motivated by the desire to carry out performance and dependability studies on the basis of an algebraic framework, exploiting the beneficial characteristics of process algebras for the purpose of stochastic modelling. SPAs allow one to specify and investigate both functional and temporal properties. The significant advantage of such an integrated approach is obvious: early consideration of all major design aspects, avoiding costly redesign. Research on SPA

[*] Corresponding author. Present address: Systems Validation Centre, Department of Computer Science, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands. Tel.: +31-53-4894661; fax: +31-53-4893247.
E-mail address: hermanns@cs.utwente.nl (H. Hermanns).

PII: S0166-5316(99)00056-5

has been presented in detail in several publications, e.g. [6,15,21,27,36,53]. The community of SPA researchers is still small, however, several European research groups work intensively in this exciting area and meet regularly at the successful series of *Workshops on Process Algebras and Performance Modelling* (PAPM) [1] .

This paper is about a modelling tool, the TIPPtool, which reflects the state-of-the-art of SPA research. Development of the tool started as early as 1992, the original aim being a prototype tool for demonstrating the feasibility of our ideas. Step by step, we added new features, allowing more general and more efficient specification and analysis, as well as a user-friendly graphical front end. Over the years, the tool has been extensively used in the TIPP project as a testbed for the semantics of different SPA languages and the corresponding algorithms. Meanwhile, the tool has reached a relatively high degree of maturity, supporting compositional modelling and analysis of complex distributed systems.

The core of this tool is an SPA language where actions either happen immediately, or are delayed in time, the delay satisfying a Markovian assumption [27]. Beside some support for analysis of functional aspects, the tool offers algorithms for the numerical analysis of the underlying stochastic process which, under certain restrictions, turns out to be a Markov chain. Exact and approximate evaluation techniques are provided for stationary as well as transient analysis. The tool is capable of handling large state spaces, and it incorporates some very advanced features, such as the semi-automatic compositional aggregation of complex models.

Among related work, the PEPA Workbench, developed by Hillston et al. in Edinburgh [18], is another tool for performance evaluation, where Markov chain models are also specified by means of a process algebra. The tool TwoTowers [5], based on two existing tools (one for functional analysis and one for performance analysis), also employs a stochastic process algebra as its specification formalism.

The paper is organised as follows: in Section 2, we summarise the theoretical background of stochastic process algebras. Section 3 gives an overview of the components of the tool and their inter-operation. All aspects of model specification are discussed in Section 4, and analysis algorithms are the subject of Section 5. Section 6 briefly discusses some implementation considerations and suggestions for improvement. In Section 7 we demonstrate the use of the tool by means of a non-trivial case study. Section 8 concludes the paper.

2. Foundations of stochastic process algebras

2.1. Process algebras

Classical process algebras (e.g., LOTOS [8], CSP [40], CCS [50]) have been designed as formal description techniques for concurrent systems. They are well suited to describe reactive systems, such as operating systems, automation systems, communication protocols, etc. Basically, a process algebra provides a language for describing systems as a cooperation of smaller components, which themselves belong to the language. However, there are some distinguishing features, schematically visualised in Fig. 1. The basic constructs from which all specifications are built are *actions* and *processes*, where processes may perform actions. In Fig. 1, processes are represented as blocks of different shape, and actions appear as labels ($a, b, \ldots$) of bidirectional arrows. The description formalism is *compositional*, which means that it allows to build highly modular and hierarchical system descriptions using composition operators.

[1] See, for further details [9,19,35,38,54,56].

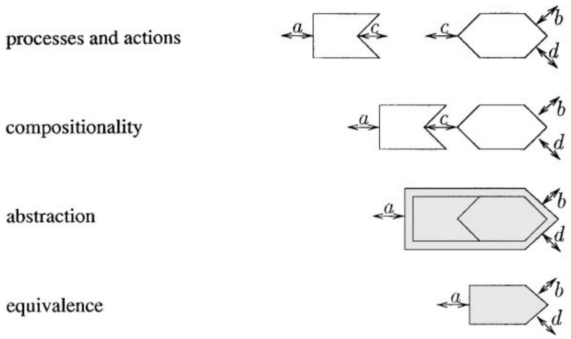


Fig. 1. Basic principles of process algebras.

These operators are provided by the language to construct processes out of smaller processes. For instance, a parallel composition operator is used to express concurrent execution (of, say, action c) and possible synchronisation of processes. Another important operator realises *abstraction*. Details of a specification which are internal details at a certain level of system description can be internalised by hiding them from the environment. Several notions of *equivalence* make it possible to reason about the behaviour of a system, e.g., to decide whether two systems are equivalent. Apart from a formal means for verification and validation purposes, equivalence-preserving transformations can be profitably employed in order to reduce the complexity of the system. This can also be performed in a compositional way, i.e., system *parts* can be replaced by behaviourally equivalent but aggregated representations. A formal semantics and an algebraic framework ease the handling and comparison of specifications.

Let us exemplify the basic constructs of a process algebraic specification by means of a simple queuing system. It consists of an arrival process *Arrival*, a queue with finite capacity, and a *Server*. First, we model an arrival process as an infinite sequence of incoming arrivals (*arrive*), each followed by an enqueue action (*enq*). This is specified using the *prefix* operator ‘;’.

$$\mathit{Arrival} := \mathit{arrive}; \mathit{enq};\ \mathit{Arrival}$$

The behaviour of a finite queue can be described by a family of processes, one for each value of the current queue population. Depending on the population, the queue may permit to enqueue a job (*enq*), dequeue a job (*deq*) or both. The latter possibility is described by a *choice* operator [] between two alternatives.

$$\begin{aligned}&\mathit{Queue}_0 := \mathit{enq}; \mathit{Queue}_1\\&\mathit{Queue}_i := \mathit{enq}; \mathit{Queue}_{i+1}\ [\,]\ \mathit{deq}; \mathit{Queue}_{i-1} \qquad 1 \le i < \mathit{max}\\&\mathit{Queue}_{\mathit{max}} := \mathit{deq}; \mathit{Queue}_{\mathit{max}-1}\end{aligned}$$

Next, we need to define a server process, as follows:

$$\mathit{Server} := \mathit{deq}; \mathit{serve}; \mathit{Server}$$

These separate processes can now be combined by the *parallel composition* operator |[. . .]| in order to describe the whole queuing system. This operator is parametrised with a list . . . of actions on which the partners are required to synchronise

$$\mathit{System} := \mathit{Arrival} \quad |[\mathit{enq}]| \quad \mathit{Queue}_0 \quad |[\mathit{deq}]| \quad \mathit{Server}$$

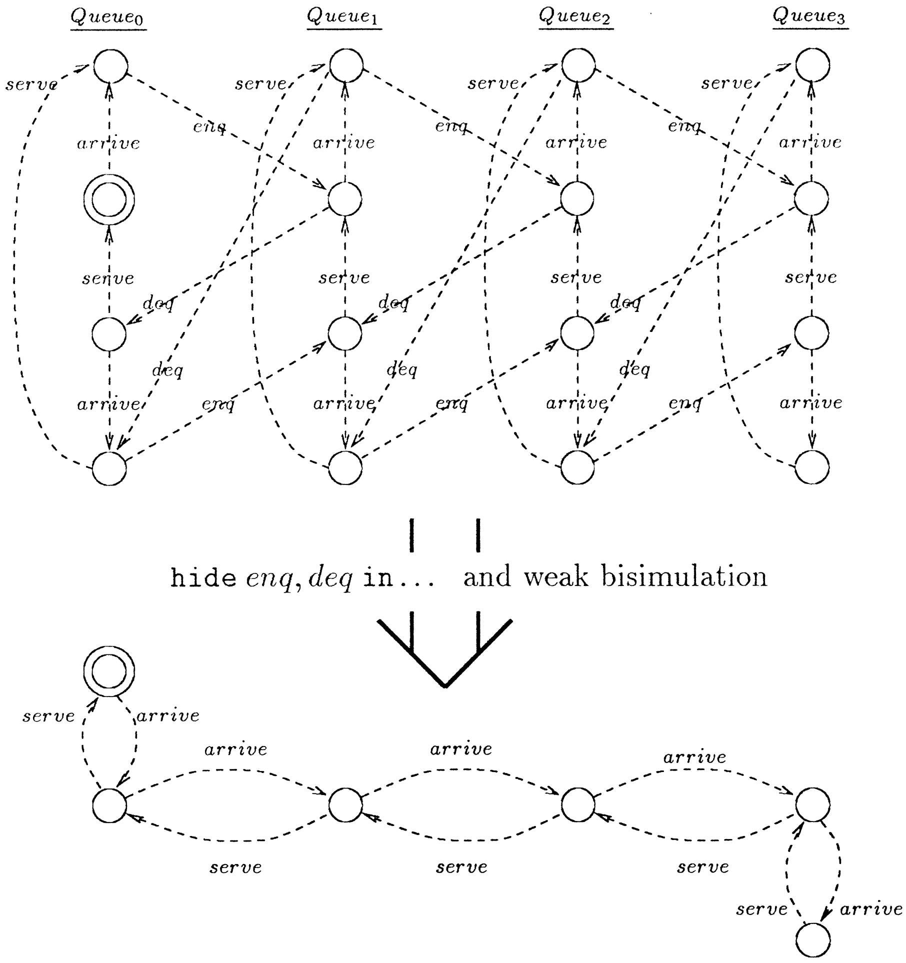


Fig. 2. Semantic model, hiding and aggregation.

A formal semantics associates each language expression with an unambiguous interpretation represented in terms of a variant of the well known state transition diagrams. This *labelled transition system* (LTS) is obtained by structural operational rules [52] which define for each language expression a specific LTS as the unique semantic model. Fig. 2 (top) shows the semantic model for our example queuing system, under the assumption that the maximal population of the queue is $max = 3$. There are 16 states, the initial state being indicated by a double circle. A transition between two states is represented by a dashed arrow and is labelled with an action which occurs when the system changes from one state to another. Since we can assume that we are not interested in the internal details of interaction between *Arrival* and *Queue*, respectively, *Queue* and *Server*, we may wish to only observe actions *arrive* and *serve*. This requires *abstraction* from internal details, and is achieved by employing the *hiding* operator:

hide *enq*, *deq* in *System*

As a result, actions *enq* and *deq* are now *internal* actions, i.e., they are not visible from the environment. Actions hidden from the environment become the distinguished internal action τ. In other words, the semantic model of the above expression is obtained by turning all *enq* or *deq* labels appearing in Fig. 2 (top) into τ.

Such τ-actions can be eliminated from the semantic model using an equivalence which is insensitive to internal details of a specification, such as *weak bisimulation*. Weak bisimulation is one of the central notions of equivalence in the general context of process algebras [50]. Fig. 2 (bottom) shows an LTS, which is weakly bisimilar to the one on top (where all *enq*-and *deq*-actions have been replaced by τ before applying weak bisimulation). It may be surprising that the resulting LTS has six and not four states (we assumed $max = 3$). This is due to the fact that the arrival of a customer and its enqueuing into the queue are separate actions, so that one more arrival is possible if the queue is already full. Likewise, dequeuing and serving are modelled as separate actions, such that at the moment the queue becomes empty, the server is still serving the last customer.

2.2. Stochastic process algebras

Parallel and distributed systems are usually fully designed and functionally tested before any attempt is made to determine their quantitative characteristics such as performance and dependability. As a consequence, costly redesign of both hardware and software is often needed. In order to contribute to the avoidance of such costs, the SPA modelling paradigm is aimed at the integration of qualitative-functional and quantitative-temporal aspects in a single specification and modelling approach [21].

In order to achieve this integration, temporal information is attached to actions, in the form of continuous random variables, representing activity durations. Models enhanced in this way are well-suited to capture the functional and temporal behaviour of a large range of applications which may be referred to as *shared resource systems*. These systems are characterised by randomly varying temporal behaviour, possibly due to data-dependent execution times, traffic-dependent communication delays, or runtimes which are highly dependent on unpredictable environmental conditions. Examples are communication networks and distributed systems, central server systems and parallel machines, or production lines and workflow systems.

The concept of stochastic process algebras follows the lines of classical process algebras: the system behaviour is described by an abstract language from which a LTS is generated, using structural operational rules [27]. The additional time information in the semantic model makes it possible to evaluate different system aspects:

- functional behaviour (e.g., liveness or deadlocks),
- temporal behaviour (e.g., throughput, waiting times, reliability),
- combined properties (e.g., probability of timeout, duration of certain event sequences).

Let us give a stochastic process algebra specification for the above queuing system, by attaching distributions to actions. We assume that the arrival process is a Poisson process with rate λ and the service time is exponentially distributed with rate μ. We are not forced to associate a duration with every action. Actions without durations happen as soon as possible, therefore, they are called *immediate* actions. In our example, enqueuing and dequeuing is assumed to happen without any relevant delay, thus *enq* and *deq* are immediate.

$$\begin{aligned}&\mathit{Arrival} := (\mathit{arrive}, \lambda);\ \mathit{enq};\mathit{Arrival}\\&\mathit{Server} := \mathit{deq};(\mathit{serve}, \mu);\ \mathit{Server}\end{aligned}$$

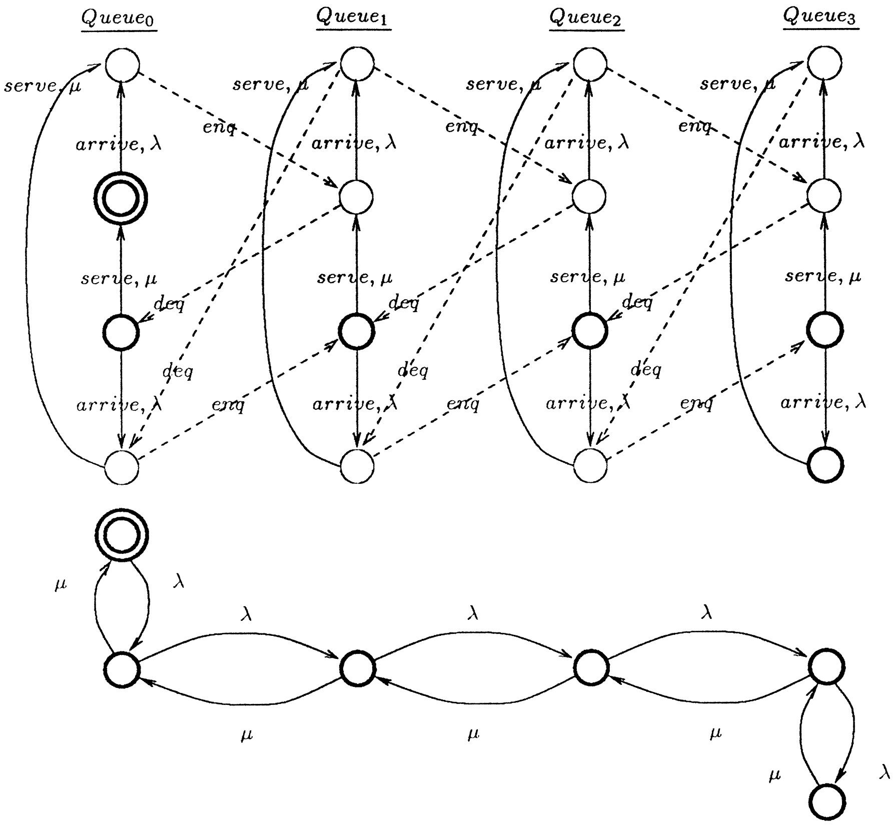


Fig. 3. (Top) the labelled transition system for the example queuing system; (bottom) the corresponding CTMC.

The queue is specified as before (it is only involved in *enq* and *deq*, therefore, its specification does not have to be changed) and the composed system is (also as above)

$$\mathit{System} := \mathit{Arrival} \quad |[\mathit{enq}]| \quad \mathit{Queue}_0 \quad |[\mathit{deq}]| \quad \mathit{Server}$$

Fig. 3 depicts the labelled transition system associated with this model, again assuming a maximal queue size of $max = 3$. Note that, there are two kinds of transitions between states: timed transitions (drawn by solid lines) which are associated with an exponential delay, and immediate transitions which happen as soon as the respective action is enabled.

States without outgoing immediate transition are shown emphasised in the figure. Under the assumption, that *enq* and *deq* happen without any delay, the emphasised states correspond to states of a continuous time Markov chain (CTMC). This chain is shown at the bottom of the figure. It is isomorphic to an LTS obtained by applying the notion of *weak Markovian bisimulation*, after hiding *enq* and *deq*. Weak Markovian bisimulation is an adaptation of weak bisimulation to the setting of timed and immediate actions [26]. Abstraction from the two immediate actions *enq* and *deq* turns out to be an essential prerequisite to unambiguously determine the Markov chain underlying this specification. If, say, *enq* is hidden, we can indeed be sure that our assumption that *enq* happens without any delay is justified. Otherwise, it may be the case that *System* is used as a component in further composition contexts, which require synchronisation on action *enq*. In this case, the Markov chain depends on additional timing constraints imposed on *enq*. Therefore, it is not possible to remove *enq*, as long as further synchronisation on *enq* is still possible. [2]

This particularity highlights an important difference in the semantics of immediate actions compared to the semantics of immediate transitions in GSPN [2]. In contrast to the priority levels defined in GSPN, immediate actions do not always have priority over timed actions. Under the notion of weak Markovian bisimulation only *internal* immediate actions preempt timed actions. Such a distinction is necessary to ensure that equivalent components remain equivalent in the context of arbitrary further compositions, see below (and [27] for an exhaustive discussion).

2.3. Bisimulation and compositional analysis

As illustrated in the running example, the notion of bisimulation is of high importance. Bisimulation manifests itself in the following way: two states of a process are bisimilar if they have the same possibilities to interact (with a third party) and reach pairwise bisimilar states after any of these interactions [50]. This definition only accounts for immediate actions. On the level of Markov chains, a corresponding definition is provided by the notion of *lumpability*. Two states of a Markov chain are lumpable if they have the same cumulative rate of reaching pairwise lumpable states [44]. *Markovian bisimulation* reflects lumpability and bisimulation on timed transitions, by imposing constraints on actions and rates, see [30] or [11,36] for details. *Weak* Markovian bisimulation additionally allows abstraction from internal immediate actions, in analogy to ordinary weak bisimulation [31].

Such equivalences are defined in terms of states and transitions, i.e., on the level of the LTS. In order to get insight into their particularities, it is highly valuable to characterise their distinguishing power on the level of the language by means of *equational laws*. Some important laws for weak Markovian bisimulation are given below [29]:

$$(a,\lambda);\ \tau;\ P = (a,\lambda);\ P \tag{1}$$

$$\tau;\ P \quad [\,] \quad (a,\lambda);\ Q = \tau;\ P \tag{2}$$

$$(a,\lambda);\ P \quad [\,] \quad (a,\mu);\ P = (a,\lambda+\mu);\ P \tag{3}$$

They reflect the following characteristics of weak Markovian bisimulation. According to law (1), an immediate internal action following a timed action has no effect and can therefore be eliminated. Law (2) states that the internal immediate action τ has priority over a Markovian timed action, since the former will happen without any delay. Recall that this priority only holds for *internal* immediate actions. Law (3) says that the rate of two timed transitions (with the same action) can be cumulated. This law reflects lumpability, and is also valid for (non-weak) Markovian bisimulation.

In the presence of composition operators, such as hiding and parallel composition, it is highly desirable that equivalences are *substitutive*. Intuitively, substitutivity allows to replace components by equivalent ones within a large specification, without changing the overall behaviour. Substitutive equivalences are also called *congruences*. Indeed, Markovian and weak Markovian bisimulation are congruences, except for the choice operator, where weak bisimulations generally require a slight refinement [29]. Practically important, such equivalences allow *compositional aggregation* techniques, where the size of a component's state space may be reduced, without affecting any significant property of the whole model.

[2] Indeed, abstraction rules out any further synchronisation, since τ is not allowed to appear in the list '. . . ' of synchronising actions of a parallel composition operator |[. . .]|.

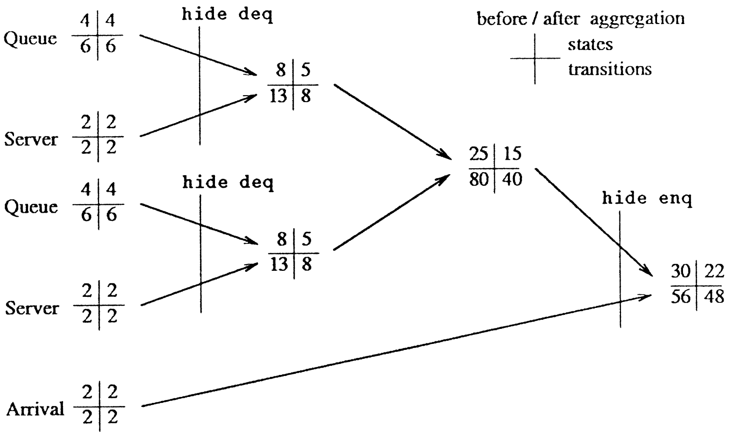


Fig. 4. Compositional aggregation of the example queuing system.

Compositional aggregation has successfully been applied to a variety of systems, see e.g. [14], for an impressive industrial case study.

Let us return to our queuing example in order to illustrate compositional aggregation. We will now model a queuing system with one Poisson arrival process, two queues and two servers. We can build this system from the same components, i.e., processes *Arrival*, *Queue* and *Server* are defined as above. The system is now

$$\mathit{System} := \mathit{Arrival} \quad |[\mathit{enq}]| \quad ((\mathit{Queue}_0 \quad |[\mathit{deq}]| \quad \mathit{Server}) \quad ||| \quad (\mathit{Queue}_0 \quad |[\mathit{deq}]| \quad \mathit{Server}))$$

If the queue sizes are given by $max = 3$, the model has 128 states and 384 transitions. By hiding actions *enq* and *deq* and applying weak Markovian bisimulation to the complete system, the state space can be aggregated to 22 states and 48 transitions. However, aggregation can also be performed in a compositional fashion such that the 128 state system never has to be generated explicitly: the subsystem consisting of one queue–server pair has eight states, which can be aggregated to five states. Combining both (aggregated) queue–server pairs, we obtain 25 states which can be aggregated to 15 states (this aggregation step mainly exploits symmetry of the model). If this aggregated system is combined with the arrival process, we get 30 states which can again be aggregated to 22 states. This concept of compositional aggregation is illustrated in Fig. 4, where the size of the state space and the number of transitions are given for each aggregation step.

It is interesting to observe that this system exhibits non-deterministic behaviour: after the completion of a Markovian timed action *arrive*, it is left unspecified which of the two queues synchronises with the arrival process on immediate action *enq* (provided, of course, neither queue is full, in which case the behaviour is deterministic). As a consequence, the Markov chain underlying this specification is formally not completely specified. One may assume that both alternatives occur with the same probability. Alternatively, one may explicitly add information (such as a scheduling strategy) in order to resolve non-determinism. In Section 4, we will follow the latter path.

The reduction obtained by compositional application of weak Markovian bisimulation relies on two phenomena.

1. Since the equivalence notion incorporates the idea of lumpability, symmetries within the specification can be exploited, as in the above example, where the two queue–server pairs are replicas of each other.

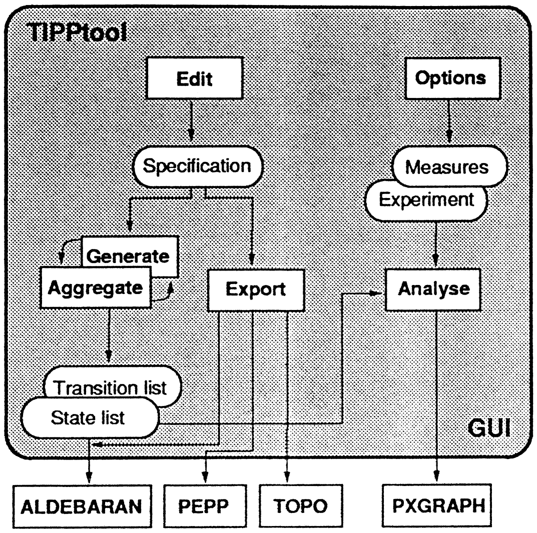


Fig. 5. Structure of the tool.

(Note, however, that in general the lumping effect of (weak and strong) Markovian bisimulation goes beyond simple symmetry exploitation [16,25].)

2. Abstraction of sequences of internal immediate actions can be exploited in a component wise fashion. Immediate transitions contribute to the state space only to the extent in which they are required for further composition. Therefore, it is possible to abstract from *internal* immediate actions (over which synchronisation is not allowed). This effect is also present in (the first and the final step of) the aggregation example depicted in Fig. 4.

3. Tool overview

The TIPPtool consists of several components. It includes a parser which checks specifications for syntactic correctness. The language accepted by the parser is a superset of Basic LOTOS [8], and will be explained in detail in Section 4. If a specification is syntactically correct, the tool applies the structural operational rules automatically and generates the underlying semantic model and its corresponding Markov chain. It provides several numerical algorithms for the solution of the Markov chain and the computation of measures. Algorithms are provided to aggregate the LTS according to different bisimulation equivalences. The interaction among different components of the tool is shown in Fig. 5.

Specifications can be created with an editor which is provided by the tool (`Edit` component). The `Generate/Aggregate` component is responsible for the generation of the semantic model and for the aggregation of the LTS according to a bisimulation equivalence. The user may currently choose between four bisimulation equivalences.

Via the `Options`, the user can specify various measures to be calculated, such as the probability of the system being in a certain subset of states, or the throughput (i.e., the mean frequency of occurrence) of some action. An experiment description contains information about activity rates which may vary, i.e., which are represented by a textual string to be replaced during analysis by a concrete value. A series of experiments can be carried out automatically in an efficient manner, generating numerical results for different values of a certain model parameter, while the state space only needs to be generated once.

Models can be analysed with the Analyze module. It provides several numerical solution methods for the steady state analysis as well as for the transient analysis of Markov chains. If an experiment series has been carried out, the results are presented graphically with the tool PXGRAPH from UC Berkeley.

The Export module of the tool provides interfaces to three other tools, ALDEBARAN [17], PEPP [24], and TOPO [45]. The former interface is based on a special semantics for SPAs which generates stochastic task graphs [34,46], for which the tool PEPP offers a wide range of both exact and approximate analysis algorithms, some of which work even for general distributions. The second interface provides support for the translation of SPA specifications into a format suitable for the LOTOS tool TOPO. Among other functionalities, this tool is capable of building C-programs from LOTOS specifications. The third interface can be used to exploit the powerful bisimulation equivalence algorithms of the tool ALDEBARAN. Here, the interface is at the level of the state space.

4. Model specification

In this section, we explain the details of the specification language supported by the TIPPtool. In particular, we highlight how *parametric processes* and *inter-process communication* can be used to model complex dependences conveniently. The specification language of the TIPPtool is closely related to LOTOS [41], the ISO standardised specification language. To reflect the passing of time in a specification, randomly varying delays may be attached to actions. At the moment, for reasons of analytical tractability, only exponential distributions are supported. Thus, our language can be seen as a means for the compositional high-level description of a CTMC.

4.1. Basic operators

The available operators are listed in Table 1; the upper half has already been used in Section 2, namely (timed and immediate) action prefix, choice, hiding and parallel composition (with synchronisation). Note

Table 1
Basic syntax of the TIPPtool language (P and Q are behaviour expressions, $a, a_1, \dots, a_n$ are action names)

Name		Syntax
Timed action prefix	Observable	$(a, r);\ \ P$
	Internal	$(\texttt{tau}, r); P$
Immediate action prefix	Observable	$a;\ \ P$
	Internal	$\texttt{tau};\ P$
Choice		$P\,[\,]\,Q$
Parallel composition	With synchronisation	$P\ \vert[\,a_1, \dots, a_n\,]\vert\ Q$
	Pure interleaving	$P\vert\vert\vert Q$
Hiding		$\texttt{hide}\,a_1, \dots, a_n\,\texttt{in}\,P$
Inaction		$\texttt{stop}$
Successful termination		$\texttt{exit}$
Enabling		$P \gg Q$
Disruption		$P\,[>Q$
Process instantiation		$P\,[a_1, \dots, a_n]$

that the internal action τ is denoted `tau`. If no synchronisation between two processes is required, the pure interleaving operator ||| models independent parallelism. Synchronisation is possible both between immediate or between timed actions. Synchronising a timed with an immediate action is not allowed. When synchronising on timed actions, we define the resulting rate to be the *product* of the two partner rates. This definition preserves compositionality [27,30].

The intuition of the remaining operators is as follows: `stop` represents an inactive process, i.e., a process which cannot perform any action. `exit` behaves like `stop` after issuing a distinguished signal δ indicating that the process has successfully terminated. This signal is used in combination with the enabling operator » to model sequential execution of two processes. Disruption with `[>` is useful to model the interruption of one process by another. As soon as Q executes any action, P is preempted and control is handed over to Q. There is one exception to this general rule: in order to allow the passing of time until preemption takes place, internal, timed actions of Q do not preempt P. Process instantiations $P[a_1, \dots , a_n]$ resemble the invocation of procedures in procedural programming languages such as PASCAL. We will see examples in the sequel.

4.2. Value passing and inter-process communication

The concept of process instantiation makes it possible to parameterise processes over action names. In addition, it is often convenient to parameterise a specification with some data values, such as a rate, or the length of a queue. Indeed, the above specification of a queue can be seen as a simple example for a data-dependent specification, since the parameter i governs the synchronisation capabilities of $Queue_i$.

We have incorporated the possibility to describe data dependencies explicitly in the TIPPtool. In addition, data can also be attached as parameters to actions, and therefore, be exchanged between processes, using the concept of inter-process communication [8]. This is highly beneficial, in order to conveniently describe complex dependencies.

As a prerequisite for inter-process communication and data parametrisation, it is necessary to support at least basic data types. In the current version of the TIPPtool, the type integer may be used for inter-process communication, and both integer and (positive) real for process parametrisation. In order to be used for inter-process communication, data values have to be declared. A value declaration has the form `!`*value* and is usually attached to an action, as in a`!2;stop`. *value* may be a specific value, a variable or an arithmetic expression. Variable declarations are the counterpart of value declarations. They have the form `?`*variable*`:`*type* where *variable* is the name of the variable. An example is a`?x:int;`P.

These basic ingredients can be combined to form three different types of inter-process communication supported by the TIPPtool.

- *Value passing*. If value declaration and variable declaration are combined in a synchronisation, the value is transmitted from one process to the other and the variable is instantiated by the transmitted value. An example is

  a`!2; stop` $|[a]|$ a`?x:int;` $b!(x+1);\ \dots$

- *Value matching*. If synchronisation on actions is specified where both actions involve value declarations, this synchronisation is only possible if the values turn out to be equal, as in the example given below.

  a`!2; stop` $|[a]|$ a`!(1+1);` $\dots$

Table 2
Language constructs for inter-process communication and data parametrisation (*bool-expr* is a Boolean expression, possibly containing ‘<’, ‘=’, and ‘>’; parameters $r, r_1, \ldots, r_m$ can be instantiated by arbitrary positive real numbers, integers or by arithmetic expressions of such numbers)

Name	Syntax
Value declarations (send)	$a\,!\,r$
Variable declarations (receive)	a?x:int
Return values	exit (n)
Accept values	accept x:int in P
Conditional constructs	[*bool-expr*]→ P
Parametric processes	$P[a_1, \ldots, a_n](r_1, \ldots, r_m)$

- *Value generation.* If several actions are synchronised, each with a variable declaration of the same type, a synchronisation with another process which offers a value of the required type yields a form of multicast communication.

$$a!2;\ \texttt{stop}\ |[a]|\ a?\texttt{x:int};\ P\ |[a]|\ a?\texttt{y:int};\ s\ldots$$

With the inclusion of values, further extensions to the basic syntax are convenient. When the enabling operator is used, it is sometimes desirable to receive a value from the exiting process, as for instance in a?x:int; exit(x) » accept v:int in P. Furthermore, it is convenient to describe behaviours which depend on conditions. For instance, the queue with three places can be described as follows.

```
Queue(i)  :=  [i < 3] → (enq; Queue(i + 1))  [ ]
              [i > 0] → (deq; Queue(i − 1))
```

The operators currently supported for inter-process communication and parametric processes are summarised in Table 2. Note that inter-process communication is currently only implemented for immediate actions.

4.3. The queuing example revisited

In order to illustrate the power of these language elements, we return to our running example of a queuing system. We modify the model in order to represent the join-shortest-queue (JSQ) service strategy. The idea is to insert a new process, Scheduler, between arrival and queue, whose task it is to insert an arriving job into the shortest queue, i.e., the queue with smallest current population. For this purpose, Scheduler scans all queues in order to determine the shortest queue, whenever an arrival has occurred. Process Server is defined as before. The arrival and queue processes, on the other hand, do not communicate directly via action enq any more, but via the Scheduler. Therefore, we simplify the arrival process as follows (‘process’ and ‘endproc’ are keywords enclosing a process specification):

```
process Arrival := (arrive, lambda); Arrival endproc
```

i.e., Arrival and Scheduler now synchronise on the timed action arrive. The top-level specification is as follows:

```
(Arrival |[arrive]|Scheduler(2,1,1,100,100))
  |[ask,repl,enq]|
 ((Queue(1,0)|[deq] |Server)||| (Queue(2,0)|[deq]|Server))
```

The Scheduler is a parametric process, which can be used for an arbitrary number noq of queues. After an arrival (action arrive with the “passive” rate 1), the scheduler polls all noq queues in order to identify the queue with the smallest population (actions ask and repl). Each queue sends as a reply its current population. After polling, Scheduler has identified the shortest queue. It then enqueues the job into that queue (action enq). Parameters c, b, nc and nb are needed to store the current queue, the queue with (currently) smallest population, the current population and the (currently) smallest population. In the example, nc and nb are (re-)initialised with the value 100, a value larger than any real queue population.

```
process Scheduler(noq,c,b,nc,nb) :=
  (arrive,1);  AskQueue(noq,c,b,nc,nb)
where
  process AskQueue(noq,c,b,nc,nb) :=
    ask!c; repl?x:int; Decide  (noq,c,b,x,nb)
  endproc
  process Decide (noq,c,b,nc,nb):=
    [c<noq and nc<nb]
        → AskQueue(noq,c+1,c,nc,nc)              []
    [c<noq and  (nc>nb or nc=nb)]
        AskQueue(noq,c+1,b,nc,nb)                []
    [c=noq and nc<nb]
        → (enq!c; Scheduler(noq,1,1,100,100))  []
    [c=noq and  (nc>nb or nc=nb)]
        → (enq!b; Scheduler(noq,1,1,100,100))
  endproc
endproc
```

The Queue process has to be modified as well: it now has a parameter s which denotes the identity of the queue. In addition, it can now perform actions ask and repl in order to supply information on the current queue size to the scheduler. Note how value matching is used with actions ask and enq, and value passing is used with action repl.

```
process Queue(s,i)  :=
  [i < 3] → enq!s; Queue(s,i + 1)  []
  [i > 0] → deq; Queue(s,i − 1)
  ask!s; repl!i; Queue(s,i)        []
endproc
```

5. Analysing a specification

The semantic model serves as a basis for functional analysis and performance analysis. We will informally explain how the semantic model is constructed by the TIPPtool and how it is used later on. Details can be found in [27].

5.1. *Generating and analysing the semantic model*

The formal semantics of SPA provides an unambiguous description of how to construct the semantic model in a mechanised way. The structural operational rules can be implemented in a straightforward fashion. One such rule, e.g., is

$$\text{if} \quad P \xrightarrow{a} P' \quad \text{then} \quad P\,[\,]\,Q \xrightarrow{a} P'$$

i.e., if the process P is able to perform the action a and switch to P', then the process $P\ [\,]\ Q$ can do the same action, leading to P', thus preempting Q. Of course, a symmetric rule would allow Q to preempt P. The semantic model contains all possible states to which the specified system may evolve.

The semantic model is a directed graph whose nodes denote states and whose arcs represent transitions between states. According to the semantics, states are labelled by terms of the SPA language (in encoded notation), while the arcs contain an action name and optionally a transition rate and some auxiliary labels. The state space is either saved directly to files (while a hash-table of all states is maintained in memory) or it is temporarily stored in main memory as an adjacency list (a common data structure for graphs), depending on whether equivalence checking algorithms are enabled or not (see Section 5.3).

Once the semantic model is generated, it can be used for some elementary functional analysis. Our tool provides the capabilities of checking for deadlocks and tracing through the states, i.e., showing a path of actions leading from the initial state to a user-specified target state. Apart from that, equivalence checking algorithms can be used for deciding equivalence of two models. In this way it can be checked, for instance, whether a model meets the requirements of a high-level specification.

5.2. *Performance evaluation*

Transforming the semantic model into CTMC and then analysing it by means of numerical solution algorithms for Markov chains, we can obtain performance and reliability measures for a given specification. For didactic reasons, let us first assume that the model contains timed actions only, and later show how to extend the procedure for immediate actions.

5.2.1. *Models without immediate actions*

For any SPA model with timed actions only and finite state space, the underlying CTMC can be derived directly by associating a Markov chain state with each node of the labelled transition system [20,36]. The transitions of the CTMC are given by the union of all the arcs joining the LTS nodes, and the transition rate is the sum of the individual rates (see Fig. 6). This is justified by the properties of exponential distribution, in particular the fact that the minimum of two exponentially distributed random variables with rates λ_1, λ_2 is again exponentially distributed with rate $\lambda_1 + \lambda_2$. Transitions leading back to the same node (loops) can be neglected, since they would have no effect on the balance equations of the CTMC. The action names are only taken into account later on, when high-level performance measures are to be computed.

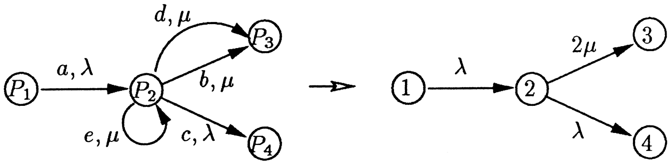


Fig. 6. Deriving a Markov chain.

In the TIPPtool, standard numerical solution algorithms [61] (Gauß–Seidel, Power method, LU factorisation, refined randomisation) are employed for steady state analysis as well as transient analysis of the CTMC. Apart from these, prototypical implementations of efficient approximation methods are realised (see Section 5.4).

5.2.2. Models with both timed and immediate actions

As discussed in Section 2, immediate actions happen as soon as they become enabled. In order to ensure that this enabling cannot be delayed by further composition, abstraction of immediate actions is mandatory. In the stochastic process, these immediate actions correspond to immediate transitions. The presence of immediate transitions leads to two kinds of states in this process: states with outgoing immediate transitions and states without such transitions. We adopt the usual terminology for the former kind of states and refer to them as *vanishing* states. All other states are called *tangible* states [2]. If several immediate transitions emanate from a single state, the decision among these alternatives is non-deterministic, and it may depend on which action is offered by the environment. If we consider the system as a closed system (which is made explicit by hiding all immediate actions) the decision among several (now internalised) immediate transitions still has to be taken. One possible solution is to weigh all alternatives with equal probabilities. Under this assumption of equi-probability, the underlying stochastic process is not a Markov chain, but a special type of semi-Markov process which has both Markovian and immediate transitions. Although, the solution of such semi-Markov processes represents no conceptual problem, the solution effort is usually reduced by eliminating vanishing states, thus yielding a CTMC with fewer states.

Several methods exist for eliminating immediate transitions. The method used in most tools is to incorporate transitions into the CTMC which are due to the traversal of some vanishing states between two tangible states. This is done until all vanishing states are bypassed [2]. The rate of these arcs is computed by multiplying the rate of the Markovian transitions leaving the source tangible state with the probability of reaching the target tangible state (see Fig. 7). This is quite a general and efficient technique. However, [55] showed that it should be applied with care in the SPA context, essentially because a non-deterministic decision is conceptually different from an equi-probable decision. Therefore, in order to remove immediate transitions, it is more appropriate for SPAs to eliminate them on the basis of bisimulation equivalences, as it has been done in Fig. 3. If non-deterministic alternatives only lead

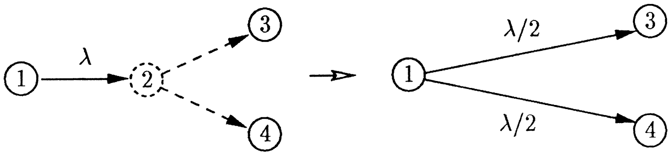


Fig. 7. Eliminating immediate transitions.

(via some internal, immediate steps) into equivalent states, equivalence-preserving transformations allow to remove this non-determinism, see Section 5.3. The TIPPtool proposes to follow this way, whenever a critical non-deterministic decision is encountered, by issuing a warning. Depending on the user's advice, it is able to proceed with performance analysis, by applying the usual elimination of vanishing states.

5.3. Compositional aggregation

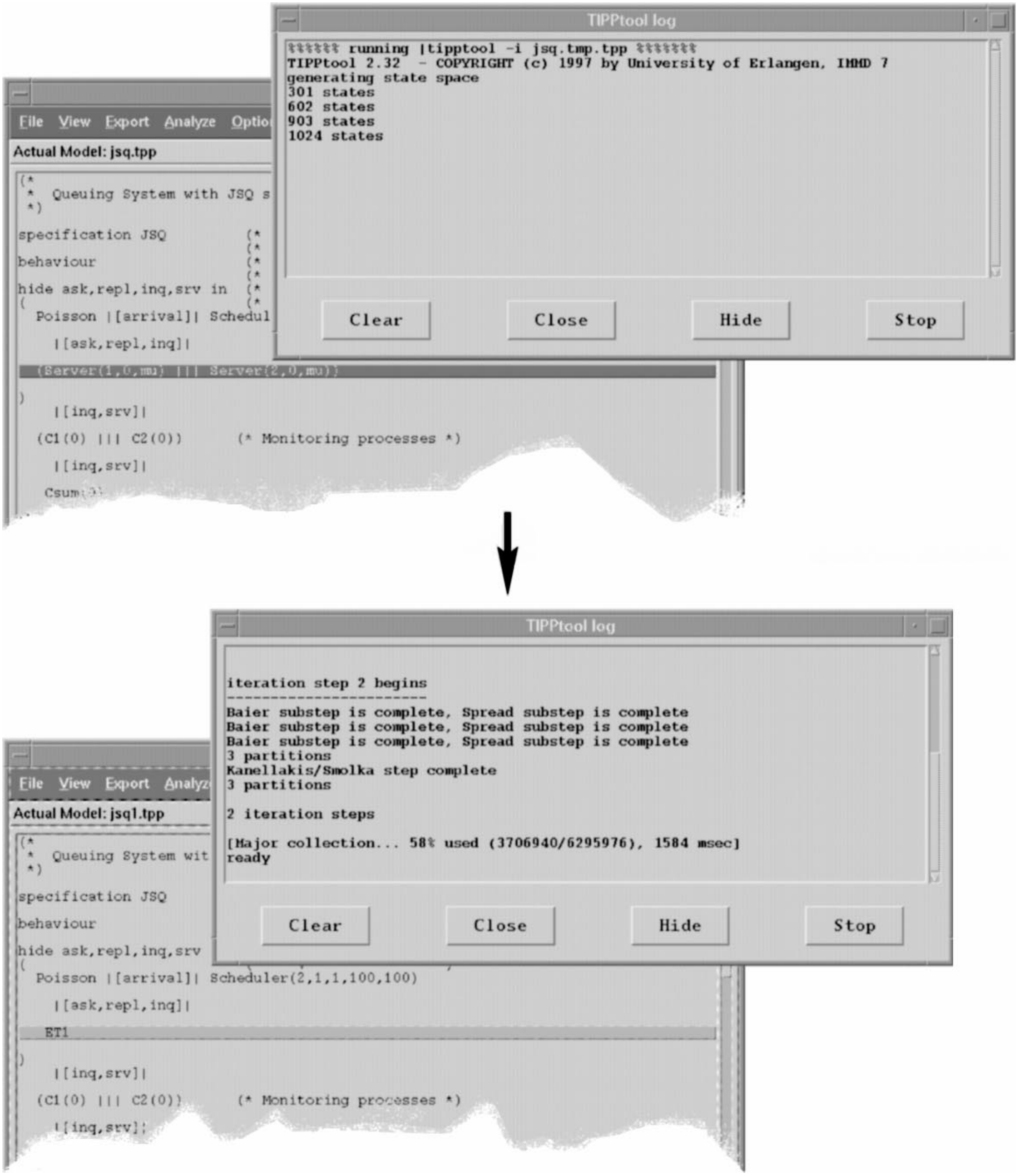
Equivalence relations such as (weak) Markovian bisimulation, introduced in Section 2.3, are beneficial both for eliminating immediate transitions, and for alleviating the state space explosion problem by means of lumping. Both effects can be achieved by means of the same strategy. For a given specification, say *System*, the key idea is to compute a specification, $System'$, which is minimal (with respect to the number of states) among all those specifications which are equivalent to *System*. Performance analysis can then be based on the minimised specification. In principle, it is possible to produce $System'$ by term rewriting on the level of the syntax, using equational laws, see e.g. [26]. A different approach works on the level of the transition system, factorising the whole state space into equivalence classes of states. A minimal representation is obtained afterwards, representing each class by a single state.

The general strategy for factorising the state space is known as *partition refinement*. A partition is a representation of a set as a disjoint union of subsets. The bisimulation algorithm should obviously compute a partition of the state space, such that the subsets correspond to the bisimulation equivalence classes. This is achieved by a successive refinement of an initial partition which consists of a single subset containing all states. The partition becomes finer and finer until no further refinement is needed, or, in technical terms, a fixed-point is reached. This fixed-point is the desired result.

This general strategy can be realised by means of efficient algorithms [42,51]. Therefore, our bisimulation algorithm prototypes implemented in the TIPPtool follow the partition refinement approach [32]. For specifications which do not contain timed transitions, we implemented Kanelakis and Smolka's algorithm to compute strong and weak bisimulation. For the converse case (only timed transitions), we implemented an algorithm which is due to Baier [4] for factorising specifications with respect to Markovian bisimulation. These two implementations form the basis of the general case, where timed and immediate transitions coexist: weak Markovian bisimulation is computed by alternating the algorithms for weak bisimulation (for immediate transitions) and Markovian bisimulation (for timed transitions) until a fixed-point is reached. Since weak Markovian bisimulation abstracts from internal, immediate transitions, this opens a way to eliminate immediate transitions from a specification, as long as they are internal. So hiding of immediate transitions is necessary for an elimination, but it is, in some cases, not sufficient, because non-deterministic internal decisions may remain after factorisation. In this case the system is underspecified, and the TIPPtool produces a warning message to the user.

Bisimulation-based minimisation is particularly beneficial if it is applied to components of a larger specification in a stepwise fashion. Since all implemented bisimulations have the algebraic property of *substitutivity*, minimisation can be applied compositionally, as illustrated in Fig. 4. Minimising an arbitrary component of a specification does not alter the behaviour of the whole specification. In this way, specifications with very large state spaces become tractable, as outlined in [28].

In the TIPPtool, compositional minimisation is supported in an elegant way. By dragging the mouse inside the edit window, it is possible to highlight a certain component of the specification and to invoke compositional minimisation of this component. When the minimised representation is computed, a new



Actual Model: jsq.tpp

```
(*
 *  Queuing System with JSQ s
 *)
specification JSQ          (*
                           (*
behaviour                  (*
                           (*
hide ask,repl,inq,srv in   (*
(                          (*
  Poisson |[arrival]| Schedul
    |[ask,repl,inq]|
  (Server(1,0,mu) ||| Server(2,0,mu))
)
    |[inq,srv]|
  (C1(0) ||| C2(0))       (* Monitoring processes *)
    |[inq,srv]|
   Csum(0)
```

```
%%%%%% running |tipptool -i jsq.tmp.tpp %%%%%%%
TIPPtool 2.32  - COPYRIGHT (c) 1997 by University of Erlangen, IMMD 7
generating state space
301 states
602 states
903 states
1024 states
```

Actual Model: jsq1.tpp

```
(*
 *  Queuing System wit
 *)
specification JSQ
behaviour
hide ask,repl,inq,srv
(
  Poisson |[arrival]| Scheduler(2,1,1,100,100)
    |[ask,repl,inq]|
   ET1
)
    |[inq,srv]|
  (C1(0) ||| C2(0))       (* Monitoring processes *)
    |[inq,srv]|
```

```
iteration step 2 begins
-----------------------
Baier substep is complete, Spread substep is complete
Baier substep is complete, Spread substep is complete
Baier substep is complete, Spread substep is complete
3 partitions
Kanellakis/Smolka step complete
3 partitions

2 iteration steps

[Major collection... 58% used (3706940/6295976), 1584 msec]
ready
```

Fig. 8. Compositional aggregation by selecting parts of the specification with the mouse.

specification is generated automatically, where the selected component has been replaced by the minimised representation. This new specification is displayed in a new, distinguished window, see Fig. 8. We used this feature for compositional aggregation of our queuing example. The resulting sizes of the component's state spaces and their aggregated versions are depicted in Fig. 4.

5.4. Approximate analysis

In addition to the exact analysis methods discussed above, prototypical implementations of two efficient approximation algorithms are integrated into the TIPPtool. Both approaches are based on decomposition. The theoretical foundations of SPA were of high importance for both approaches, since they were needed to show the correctness of the transformations imposed on the model during decomposition/aggregation [47].

5.4.1. Time scale decomposition

Time scale decomposition (TSD) is a decomposition method which tries to exploit the *near complete decomposability* (NCD) property of many Markov chains. In particular, CTMCs resulting from models which contain reliability aspects lead to NCD Markov chains. Such models tend to lead to so-called *stiff* Markov chains, which increases the solution effort immensely. TSD partitions the state space into fast and slow components, based on a distinction between fast and slow actions, according to a threshold value for the rate [37]. The generation of the whole state space at once is avoided. Only one partition at a time is held in memory. The accuracy of the results is excellent for systems with NCD structure. The algorithm is based on existing work on SPNs [7] and goes back to the decomposition/aggregation scheme of Simon and Ando [60]. Of course, there are some drawbacks, in particular due to the restriction that the partitions need to be solvable by steady state analysis, i.e., they have to be irreducible. If this is not the case, an additional error is introduced.

5.4.2. Response time approximation

Response time approximation (RTA) works on the specification level rather than on the CTMC level [47,48]. The basic principle goes back to early work on queuing networks [1,13] and more recent derivatives in the SPN context, e.g. [12]. Here, the state space is not generated for the whole model, but only for a small part of the model. The RTA algorithm for a special class of SPNs, called marked graphs, has been adapted to so-called decision free processes (DFP), and implemented in the TIPPtool, in order to derive substitute aggregates which approximate the response time of the original aggregates. Several equivalence-preserving transformations are applied to the model prior to decomposition. If the decomposed model components are still too big, they can be decomposed again recursively in a divide and conquer fashion. Thus, the state complexity is reduced by several orders of magnitude. The main limitation is that this method is restricted to DFP, a very specific class of models [47].

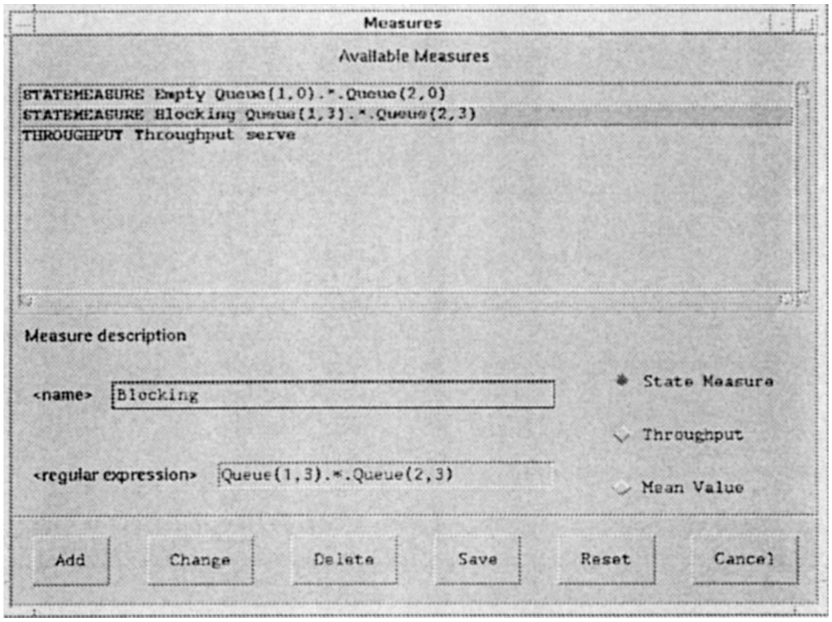
For example, a DFP whose specification is given as

$$\text{System} := P_1|[\dots]|P_2|[\dots]|P_3|[\dots]|P_4|[\dots]|P_5|[\dots]|P_6|[\dots]|P_7|[\dots]|P_8$$

is aggregated into a smaller aggregated system

$$\text{Aggregated system} := P_1|[\dots]|P_2|[\dots]|P_3|[\dots]|AP$$

where the substitute aggregate AP amalgamates the whole behaviour of the replaced components P_5-P_8 into a few actions only, thus reducing the total state space complexity. The substitute aggregate is obtained automatically, by weak bisimulation preserving transformations. Its temporal behaviour is estimated by an adaptation of the RTA algorithm for marked graphs.



```
STATEMEASURE Empty Queue(1,0).*.Queue(2,0)
STATEMEASURE Blocking Queue(1,3).*.Queue(2,3)
THROUGHPUT Throughput serve
```

<name> Blocking

<regular expression> Queue(1,3).*.Queue(2,3)

● State Measure
◇ Throughput
◇ Mean Value

Fig. 9. Measure definition dialog box.

5.5. Definition and computation of characteristic performance measures

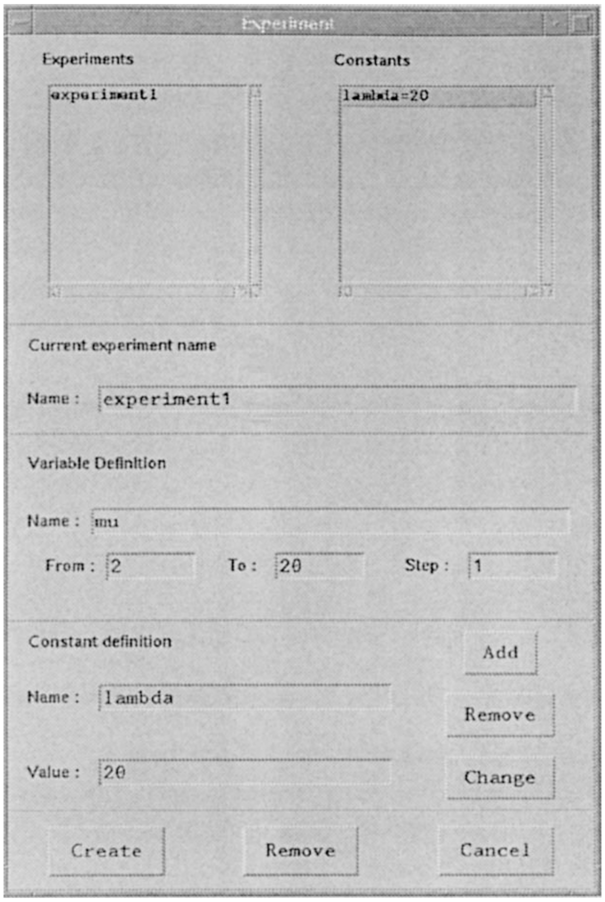
The result of steady state analysis as well as transient analysis is a vector of probabilities. This vector can be used by the TIPPtool in order to derive more sophisticated measures. Currently, three types of measures are supported:

- *State measure*. This measure represents the probability that the system is in a certain state or in a group of states. The user may specify such a set of states via regular expressions. After analysis, the tool collects all states from the state space which are matched by this expression and sums up the corresponding probabilities. Typical measures which can be obtained in this way are resource utilisation, availability, or probability of deadlock.
- *Throughput*. Here the result is not a probability, but a frequency. If the name of a timed action is specified, its throughput will be computed, i.e., the average number of occurrences of this action per time unit.
- *Mean value*. In the presence of parametric processes where one parameter represents a counter (e.g., a queue length), this measure type returns the mean value of this counter.

Fig. 9 shows how three measures for the running example are defined via a dialog box. The state measures ‘Empty’ and ‘Blocking’ correspond to the probability of finding both queues empty (full). ‘Throughput’ is the number of `serve`-actions per time unit.

5.6. Experiment definition and visualisation

The TIPPtool provides a dialog box for the definition of experiments (Fig. 10). Here, the user specifies the numerical values of symbolic model parameters. Values may either be constant or variable. In the latter case, a smallest and a largest value, as well as a stepsize have to be specified. The tool will then



experiment1

lambda=20

Name : experiment1

Name : mu

From : 2 To : 20 Step : 1

Name : lambda

Value : 20

Fig. 10. Definition of experiments.

automatically replace the symbolic parameters by the actual values, and evaluate the model for each value combination. The state space, however, only has to be generated once.

Fig. 11 contains a screen shot of some numerical results for our running example, calculated during an experiment where the service rate μ was varied, and displayed with the help of PXGRAPH.

6. Implementation considerations

In this section, we wish to give some insight into a few implementation aspects of the TIPPtool. Although, the tool is quite far advanced and provides a user-friendly interface, it still represents a prototype, and as such has shortcomings with regard to the efficiency of the implementation.

Our choice for the programming language Standard ML deserves special consideration. We used it for implementing the parser, the semantics, and the bisimulation algorithms. It was also used for the approximate solution methods. The main advantage of this language is that it is perfectly suited for implementing semantics of formal languages. Its type concept, memory management (garbage collector) and a rich library made the development of the tool a lot easier. Standard ML code is translated into

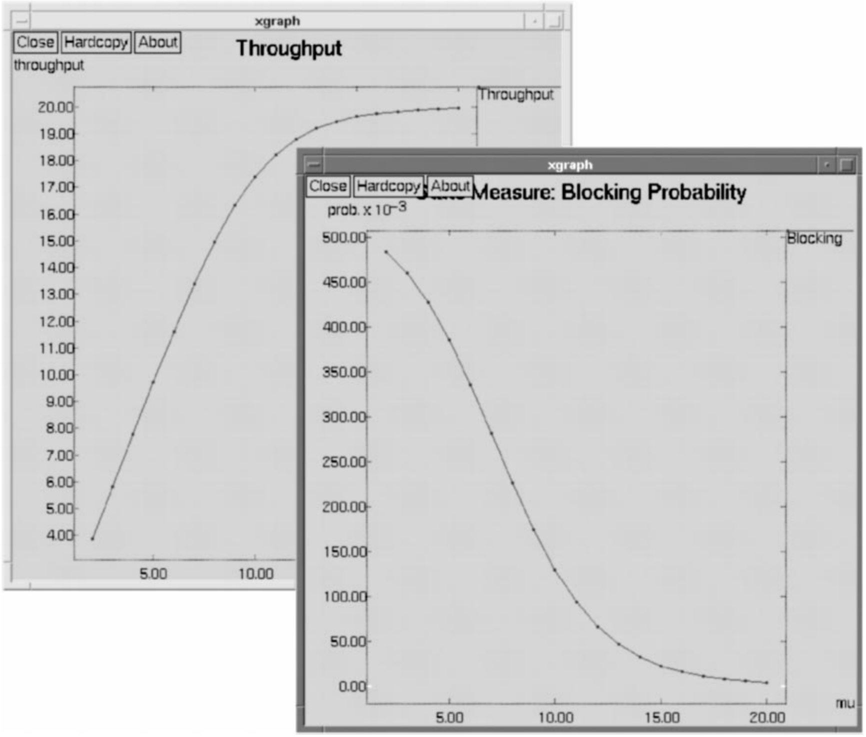


Fig. 11. Display of experimental results via PXGRAPH.

an architecture-dependent executable bytecode. Consequently, this part of the tool clearly represents a bottleneck. Furthermore, the implementations of partition refinement to compute (weak) Markovian bisimulation do not meet the best possible complexity results, their efficiency can therefore be improved a lot [25,32]. For instance, the weak bisimulation algorithm implemented in the TIPPtool currently fails for state spaces of more than 8000 states, while a far more advanced implementation of essentially the same algorithmic idea, realised in ALDEBARAN, is able to handle more than 10^6 states.

As for the numerical analysis part, we chose a ‘C’-library which provides data structures for sparse matrices, called SparseLib1.3 (by Kenneth Kundert, UC Berkeley). We extended this library by a few iterative solution methods for steady state analysis and transient analysis. The numerical solvers were implemented in ‘C’, and the communication with the state space generator is done via ASCII-files. The clear interface of the library makes it easy to integrate other solution methods into the tool.

For computing the measures, shell-scripts are used, which are based on standard UNIX-tools such as GREP, AWK and SED. Finally, the graphical user interface has been implemented using another scripting language, Tcl/Tk. The communication between the GUI and the other tools is done via UNIX-pipes. This turned out to be a good choice, since the use of Tcl/Tk makes it easy to customise the GUI of the tool.

7. Case study: a hospital communication system

In this section, we exemplify the use of the TIPPtool by means of a non-trivial case study. We describe the specification and analysis of the hospital communication system (HCS) operated by the University

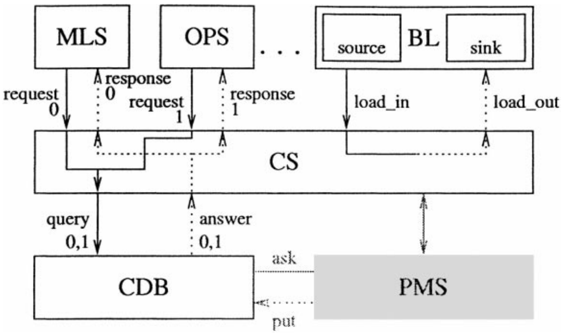


Fig. 12. The hospital communication system model.

of Erlangen. This study is part of an ongoing performance measurement and modelling project which is being conducted at the University of Erlangen [3,58,59].

7.1. *Global structure of the HCS*

The hospital communication system provides a communication infrastructure which is used by medical subsystems for exchanging information such as patient data, observation results, medical images and accounting data. Furthermore, the system consists of a huge number of interacting subsystems, among them the hospital's main laboratory, an observations processing system and the operations documentation system.

In a large clinic such as the Erlangen University hospital there exists a great variety of subsystems associated with different departments and institutions. Due to historical reasons, these decentralised information processing systems are mostly incompatible. In the past, communication between subsystems was based on proprietary one-to-one relations. Integration efforts have led to the use of standardised message formats (e.g., the Health Level 7 message standard developed for the health care sector) and the deployment of a central communication server. In Erlangen, the communication server DataGate from STC is used, whose tasks are the reception, checking, processing, routing and forwarding of (standardised) messages between medical subsystems. Among the subsystems in the Erlangen HCS, the patient management system, a SAP R/3 IS-H product, is the central business application. Beside the patient management system, there is a second large data base, the communication data base, which mirrors parts of the patient management system and contains additional medical information. The communication database serves as a fast data buffer which, from the point of view of the subsystems, provides data access about 10 times faster than the patient management system itself, and as a side effect also significantly reduces the load of the latter.

We only present a rudimentary model of the Erlangen hospital communication system which during our project has been extended in various directions. Fig. 12 shows the basic structure of the model. It consists of the communication server (CS), the communication data base (CDB) and two medical subsystems, the main laboratory system (MLS) and an observations processing system (OPS). Since almost all of the subsystems' demands for data can be satisfied by the CDB, we do not model the patient management system (PMS) at this stage (therefore the PMS is drawn grey in the figure). There is an "artificial" subsystem, representing an adjustable background load (BL), caused by those subsystems which are not explicitly modelled (BL actually consists of two subprocesses, a source and a sink which communicate with CS via actions `load_in` and `load_out`).

The top-level specification for the TIPPtool is as follows:

```
hide request, response in
  (MLS|||OPS)
  |[request, response]|
  (
  hide query, answer in
    (
    hide load_in, load_out in
      BL|[load_in, load_out]|CS
    )
  |[query,answer]|CDB
  )
```

7.2. Specification of components

We now describe a typical communication sequence in the system: the MLS, after some internal processing, needs a patient data record from the CDB. It sends a request message to the CS (action `request`) which is forwarded to the CDB (action `query`). After completing the data base lookup, the CDB generates an answer message which is sent back to the CS (action `answer`) and from there on to the MLS (action `response`). Queries initiated by the OPS subsystem do not request patient data records, they request observation results instead. Apart from that, they follow the same basic pattern as queries initiated by MLS. However, since the answer to a request for observations may consist of a number of different observations, an OPS query does not result in a single answer message, but in a random number of answer messages. Measurements on the real system have shown that this number follows a geometric distribution.

Subsystems MLS, OPS communicate with CS via actions `request` and `response`. In order for these communications to be distinguishable, we use inter-process communication, in particular value passing and value matching. All actions associated with communications initiated by subsystem MLS carry the value 0, while those originating from OPS carry the value 1. The following part of the specification illustrates how value passing and value matching are employed between MLS and CS. Note that MLS is capable of receiving a `response` at any time. This is used to ensure that CS may engage in `response` even though MLS has just decided to induce the next `request!0` (equivalently a sink could be used instead, that runs independently in parallel and just consumes `response!0` actions).

```
process MLS :=(time_mls, lambda); MLS2 [ ] response!0; MLS
endproc
process MLS2 :=request!0; MLS [ ] response!0; MLS2
endproc
```

The following portion of code specifies the behaviour of subsystem CS. In subprocess `CStodo` a `query` is submitted to the CDB. Depending on the value `x` received through action `request`, this corresponds to a query for a patient data record or for observation results. In subprocesses `CStodo` and `CStransmit`, checking the condition [x=0 or x=1] is redundant. It is, however, an example for a useful mechanism for debugging during model development, since the receiving of a request or an answer with a value different from 0 or 1 would constitute an error and result in a deadlock. Concerning the rate of timed actions such as time_cs, the basic time unit is 1 ms.

```
processCS := request?x:int;  (time_cs, 0.02); Cstodo(x)    []
       answer?x:int;  (time_cs, 0.02); Cstransmit(x)       []
       load_in; (time_cs, 0.02);  load_out; CS
endproc
process CStodo(x) := [x = 0 or x = 1] → (query!x;CS)
endproc
process CStransmit(x) := [x = 0 or x =  1] → (response!x; CS)
endproc
```

Similar value passing mechanisms are employed for the communication between the CS and the CDB. In order to perform the correct type of data base lookup, the CDB has to recall the initiator of each query. To this end, queries are stored in front of the CDB in a queue with multiple job classes, i.e., for each queue position the type of the query is stored. Again, several equivalent ways are possible to represent this data type inside a TIPP specification. The following fragment of code illustrates the concept of a queue with three waiting positions and multiple job classes.

```
process
CDB(f,p1,p2,p3):=
  [f=0] →       (query?x:int;  CDB(1,x,0,0))                 []
  [f=1] →       (query?x:int; CDB(2,p1,x,0)  []
                Lookup(1,p1,0,0))                            []
  [f=2] →       (query?x:int; CDB (3,p1,p2,x)  []
                Lookup(2,p1,p2,0))                           []
  [f=3] →       (query?x:int; (full,10); CDB(3,p1,p2,p3)  []
                Lookup(3,p1,p2,p3))
endproc
```

Parameter f denotes the current queue population. The remaining three parameters are used to store the class of the job in the first, second and third queue position. Action query sets the next free position with the value passed from the CS and increases parameter f. (If a query action meets a full queue, i.e., in the case where f=3, an exception is raised by action full). The converse operation, sending (one or several) answer(s) back to the CS and thereafter removing a job from the queue, is not shown in this fragment. It is part of the Lookup process and its subprocesses which can be entered under the condition that the CDB queue is not empty. If the Lookup process is processing a query originating from the OPS, it generates a geometrically distributed number of answers. This geometric distribution is modelled by a loop with a non-zero probability of reentry after an answer has been generated.

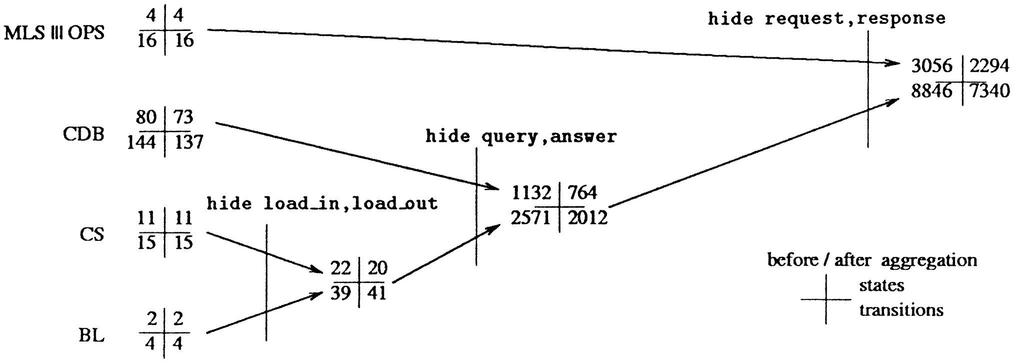


Fig. 13. Compositional aggregation applied to the hospital communication system.

7.3. State space construction

The size of the state space for this model is 4951 states and 16 236 transitions. The computation time for generating this state space was about 80 s on a SUN Ultra 1 C equipped with 512 MB of main memory. Building the generator matrix and solving the linear system for obtaining steady state probabilities took about 30 s. It was also possible to construct an aggregated state space for this model in a compositional fashion, applying stepwise aggregation by means of bisimulation, see Fig. 13. In this way, for instance, the state space of CDB could be aggregated from 80 to 73 states, and the parallel composition of CS and BL could be reduced from 22 to 20 states, after abstraction of `load_in` and `load_out`. Combining these intermediate state spaces and hiding `query` as well as `answer` we obtained 1132 states which, again, were aggregated to 764 states. Finally, we obtained 3056 states for the overall system specification which could be aggregated to an equivalent specification with 2294 states, instead of the original 4951 states. This very last aggregation step took more than 15 h, indicating that compositional aggregation still deserves some implementation effort. All aggregations were based on the notion of weak Markovian bisimulation and made use of the mouse drag-and-highlight feature to steer compositional aggregation (cf. Fig. 8).

7.4. Numerical analysis and evaluation

We have calculated a variety of numerical results for this model. There are no queues in front of process CS, i.e., subsystems wishing to communicate via the CS may have to wait until the CS is ready for synchronisation. This waiting time can become quite significant if the CS is very heavily loaded. For instance, experiments revealed that under heavy background load the subsystem MLS spends up to 11% of total time waiting for the CS (cf. Fig. 14, left). In this as in other experiments, the offered background load was increased exponentially, starting with an initial value $load_0 = 1$ request per second which was doubled in every step.

This raises the question of the utilisation of CS which of course depends on the offered load. The traffic generated by subsystems MLS and OPS is constant, namely 1 request per second for MLS and 0.25 requests per second for OPS. The offered background load, as mentioned, is increased dramatically from $load_0 = 1$ to 1024 requests per second. Fig. 14 (right) shows the proportion of time the CS is idle, depending on the offered background load. It should be noted that due to the average message processing

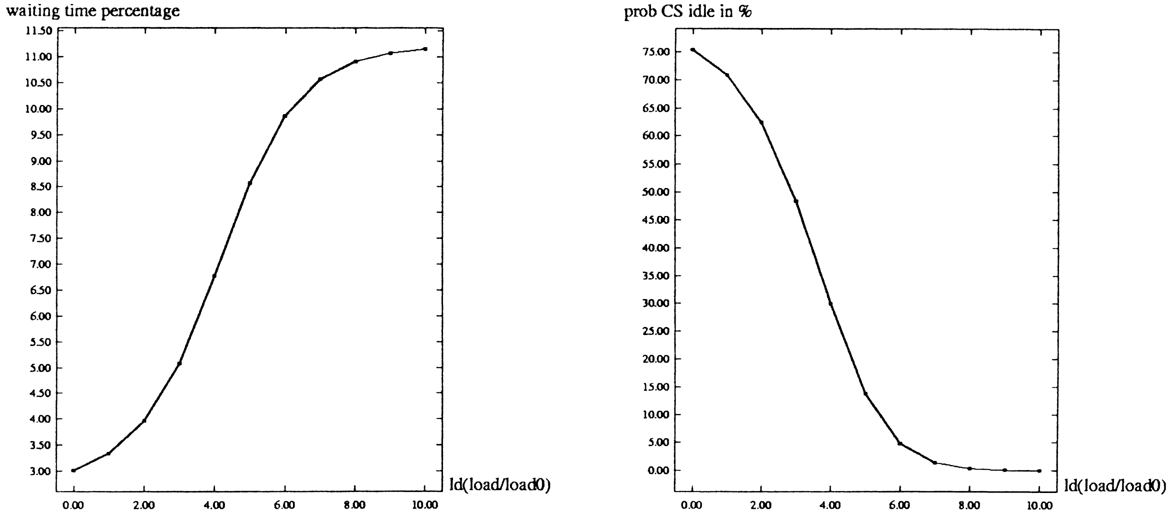


Fig. 14. (Left) waiting time percentage in subsystem MLS dependent on background load; (right) probability of CS being idle. (Note the logarithmic scale.)

time of 50 ms per message, a maximum of 20 messages per second can be carried by the CS, no matter how much background load is offered.

We now briefly discuss the size of the queue in front of the CDB and its implications. In the real system, an almost unlimited number of queries can be queued in front of the CDB, the only limitation being the size of physical memory of the machine. In order to avoid state space explosion, we can only model very small queue sizes. The diagram in Fig. 15 (left) shows that for a queue size of three waiting positions the probability of the queue being full is between 5.9 and 8.3%, depending on the offered background load.

The rate at which (MLS and OPS) queries get lost is 0.077–0.097 queries per second, see Fig. 15 (right). This corresponds to a query loss probability of 6.3 (low background load) to 8.6% (high background load).

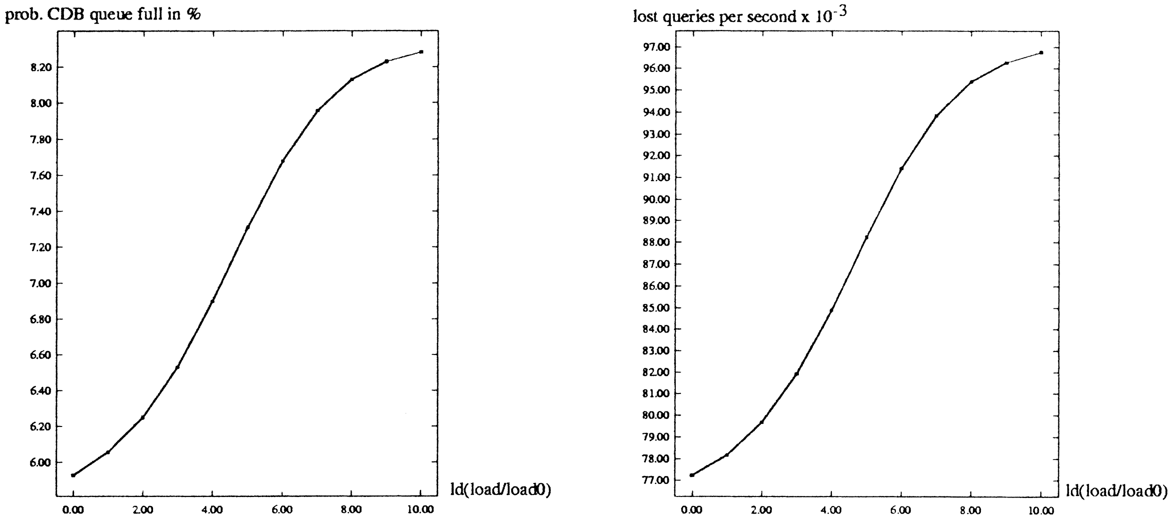


Fig. 15. (Left) probability of the CDB queue being full; (right) rate of lost queries.

In real life such high loss probabilities would of course not be acceptable. On the other hand, we observe that even for modelling a queue size of three one already obtains quite reasonable estimates for the performance measures.

The parameters used in the model were derived from measurements on the real system. We found that exponential assumptions were justified for the request inter-arrival times and even for the message processing time in the CS. The data base lookup times in the CDB were modelled as Erlang-2 distributions (as a sequence of two exponential phases), with different mean, depending on whether a query originated from the MLS or OPS subsystem.

As explained above, the specification makes excessive use of interprocess communication and of immediate actions. On the other hand, it is interesting to observe that the applied synchronisation discipline for timed actions (where the product of rates is implemented) is of no importance in this case study, because synchronisation is carried out only over immediate actions.

As mentioned above, the model described in this section was our first rudimentary model, which we extended in various directions. For instance, we explicitly modelled the PMS and the fact that queries which cannot be satisfied by the CDB have to be forwarded to the PMS (see Fig. 12, where it is indicated that synchronisation between CDB and PMS is performed via actions `ask` and `put`). We conducted experiments with varying CDB “hit rates”, i.e., varying probability of forwarding a query from the CDB to the PMS. We also studied the question whether it is beneficial to enable parallel queries to the CDB by employing a separate “query” process for each subsystem generating queries. In all of these investigations we frequently dealt with state space sizes of several hundreds of thousands of states. Compositional aggregation, however, was only possible for models of modest size, due to limitations of our implementation. Recently, on the other hand, the interface between the TIPPtool and ALDEBARAN has been successfully used to circumvent this bottleneck and to compositionally analyse another case study of more than 10^7 states in total [28].

8. Conclusion

In this paper, we have presented the status quo of the TIPPtool. We have described the particular features of a process algebra based specification formalism, together with the distinguishing components of the tool. A non-trivial case study has shown how a performance model of significant complexity can be specified and analysed compositionally. We believe that the TIPPtool currently is the leading tool for compositional modelling and analysis. In particular, it is the only tool supporting semi-automatic compositional aggregation.

Although a lot has been achieved, there remain, of course, many open problems for future research. We will briefly present some aspects of ongoing work in the TIPP project. Several attempts have been made in order to incorporate generally distributed random variables into the model [22,23,34,43,53]. However, they all suffer from the problem that general distributions lead to intractable stochastic processes, i.e., it is usually impossible to evaluate them efficiently. Simulation is a possible way out, but very costly in general. Another problem is that, so far, it is not completely solved how to obtain an algebraic framework (equivalences and equational laws) for a process algebra with general distributions. A promising approach, however, is reported in [15], using *stochastic automata* as a model based on generalised semi-Markov processes.

We have built a prototype tool for graphical model specification, called DEEDO, which is an easy-to-use front-end for users who are not familiar with the syntax of the TIPPtool’s specification language. Via a graphical editor, the user can draw automaton-like models, consisting of states and transitions. A hiding operator and a parallel composition operator are also supplied, such that hiding of internal behaviour and the combination of submodels can be specified graphically in a hierarchical fashion. Currently, DEEDO produces a textual model description which is used as input for the TIPPtool.

With the view on models with large state spaces, we are currently investigating techniques for the compact symbolic representation of the semantic model of an SPA description. The basic idea is as follows: the LTS is encoded as a Boolean function and represented as a binary decision diagram (BDD) [10]. Parallel composition of submodels is done on their BDD representation. This has the major advantage that BDDs only grow linearly in size when they are composed in parallel, whereas transition systems grow exponentially with the number of parallel components. In order to incorporate the stochastic information into the symbolic representation, we developed DNBDDs, an extension of purely functional BDDs [57]. We have implemented a tool which builds a BDD from the LTS-description generated by the TIPPtool, performs BDD-based parallel composition of submodels, and — most interesting — aggregates the model by means of a Markovian bisimulation algorithm which works exclusively on BDDs. The resulting BDD can be converted back to an LTS-file for further processing by the TIPPtool.

To summarise, the TIPPtool realises state-of-the-art techniques for compositional performance and dependability modelling. As we have described in this paper, there is a lot of ongoing activity, both in theoretical research, and concerned with the further development and optimisation of components of the tool.

References

[1] S.C. Agrawal, J.P. Buzen, A.W. Shum, Response time preservation: a general technique for developing approximate algorithms for queuing networks, in: Proceedings of the 1984 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems, Performance Evaluation Rev. 12 (3) (1984) 63–77.

[2] M. Ajmone Marsan, G. Balbo, G. Conte, Performance Models of Multiprocessor Systems, MIT Press, Cambridge, MA, 1986.

[3] B. Aures, Modellierung des Erlanger Klinikkommunikationssystems mit Hilfe von stochastischen Prozeßalgebren und TIPPtool, Studienarbeit, Universität Erlangen–Nürnberg, IMMD VII, October 1998.

[4] C. Baier, Polynomial time algorithms for testing probabilistic bisimulation and simulation, in: Proceedings of the CAV'96, Lecture Notes in Computer Science, vol. 1102, Springer, Berlin, 1996, pp. 50–61.

[5] M. Bernardo, W.R. Cleaveland, S.T. Sims, W.J. Stewart, TwoTowers: a tool integrating functional and performance analysis of concurrent systems, in: FORTE/PSTV, 1998.

[6] M. Bernardo, R. Gorrieri, Extended Markovian process algebra, in: CONCUR'96, Lecture Notes in Computer Science, vol. 1119, Springer, Berlin, 1996, pp. 315–330.

[7] A. Blakemore, S. Tripathi, Automated time scale decomposition of SPNs, in: Proceedings of the Fifth International Workshop on Petri Nets and Performance Models, Toulouse, IEEE Computer Society Press, Silverspring, MD, 1993.

[8] T. Bolognesi, E. Brinksma, Introduction to the ISO specification language LOTOS, in: P.H.J. van Eijk, C.A. Vissers, M. Diaz (Eds.), The Formal Description Technique LOTOS, North-Holland, Amsterdam, 1989, pp. 23–73.

[9] E. Brinksma, A. Nijmeijer (Eds.), Proceedings of the Fifth Workshop on Process Algebras and Performance Modelling, CTIT Technical Report Series, vol. 97-14, University of Twente, 1997.

[10] R.E. Bryant, Graph-based algorithms for Boolean function manipulation, IEEE Trans. Comput. Sci. C 35 (8) (1986) 677–691.

[11] P. Buchholz, Markovian process algebra: composition and equivalence, in: Herzog and Rettelbac (Eds.), Proceedings of the Second Workshop on Process Algebra and Performance Modelling, University of Erlangen-Nürnberg, IMMD, 1994, pp. 11–30.

[12] J. Campos, J.M. Colom, H. Jungnitz, M. Silva, Approximate throughput computation of stochastic marked graphs, IEEE Trans. Software Eng. 20 (7) (1994) 526–535.

[13] K.M. Chandy, U. Herzog, L. Woo, Parametric analysis of queuing models, IBM J. Res. Develop. 19 (1) (1975) 36–42.

[14] G. Chehaivbar, H. Garavel, 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, vol. IX, Chapman & Hall, London, 1996, pp. 435–450.

[15] 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, New York, 1998, pp. 126–148.

[16] S. Donatelli, H. Hermanns, J. Hillston, M. Ribaudo, Quantitative methods in parallel systems, in: Practice — Modelling a Distributed Mail System, chapter GSPN and SPA, Springer, Berlin, 1995.

[17] J.C. Fernandez, H. Garavel, A. Kerbrat, R. Mateescu, L. Mounier, M. Sighireanu, Cadp (cæsar/aldebaran development package): a protocol validation and verification toolbox, in: R. Alur, T.A. Henzinger (Eds.), Proceedings of the Eighth Conference on Computer-Aided Verification, New Brunswick, NJ, USA, Lecture Notes in Computer Science, vol. 1102, Springer, Berlin, 1996, pp. 437–440.

[18] S. Gilmore, J. Hillston, The PEPA workbench: a tool to support a process algebra-based approach to performance modelling, in: G. Haring, G. Kotsis (Eds.), Seventh International Conference on Modelling Techniques and Tools for Computer Performance Evaluation, Wien, 1994, pp. 353–368.

[19] S. Gilmore, J. Hillston (Eds.), Proceedings of the Third Workshop on Process Algebras and Performance Modelling (special issue), Comput. J. 38 (7) (1995).

[20] N. Götz, Stochastische Prozeßalgebren — integration von funktionalem entwurf und leistungsbewertung verteilter systeme, Ph.D. Thesis, Universität Erlangen–Nürnberg, Erlangen, 1994.

[21] N. Götz, U. Herzog, M. Rettelbach, Multiprocessor and distributed system design: the integration of functional specification and performance analysis using stochastic process algebras, in: Tutorial Proceedings of the 16th International Symposium on Computer Performance Modelling, Measurement and Evaluation, Performance'93, Lecture Notes in Computer Science, vol. 729, Springer, Berlin, 1993, pp. 121–146.

[22] N. Götz, U. Herzog, M. Rettelbach, TIPP — Einführung in die Leistungsbewertung von verteilten Systemen mit Hilfe von Prozeßalgebren, in: Verteilte Systeme — Grundlagen und zukünftige Entwicklungen aus der Sicht des SFB182, BI-Wissenschaftsverlag, Mannheim, 1993, pp. 509–531.

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

[24] F. Hartleb, A. Quick, Performance evaluation of parallel programmes — modeling and monitoring with the tool PEPP, in: B. Walke, O. Spaniol (Eds.), Proceedings der 7, GI-ITG Fachtagung Messung, Modellierung und Bewertung von Rechen- und Kommunikationssystemen, Aachen, 21–23 September 1993, Informatik Aktuell, Springer, Berlin, 1993, pp. 51–63.

[25] H. Hermanns, Interactive Markov chains, Ph.D. Thesis, Universität Erlangen–Nürnberg, Erlangen, 1998.

[26] H. Hermanns, U. Herzog, V. Mertsiotakis, Stochastic process algebras as a tool for performance and dependability modelling, in: Proceedings of the IEEE International Computer Performance and Dependability Symposium, IEEE Computer Society Press, Erlangen, 1995.

[27] H. Hermanns, U. Herzog, V. Mertsiotakis, Stochastic process algebras-between LOTOS and Markov chains, Comput. Networks ISDN Systems 30 (9–10) (1998) 901–924.

[28] H. Hermanns, J.P. Katoen, Automated compositional Markov chain generation for a plain old telephony system, Sci. Comput. Programming 36 (1) (2000) 97–127.

[29] H. Hermanns, M. Lohrey, Priority and maximal progress are completely axiomatisable, in: D. Sangiorgi, R. de Simone (Eds.), CONCUR'98 Concurrency Theory, Lecture Notes in Computer Science, vol. 1446, Springer, Berlin, 1998, pp. 237–252.

[30] H. Hermanns, M. Rettelbach, Syntax, semantics, equivalences, and axioms for MTIPP, in: U. Herzog, M. Rettelbach (Eds.), Proceedings of the Second Workshop on Process Algebra and Performance Modelling, University of Erlangen–Nürnberg, IMMD, 1994, pp. 71–88.

[31] H. Hermanns, M. Rettelbach, T. Weiß, Formal characterisation of immediate actions in SPA with nondeterministic branching, in: S. Gilmore, J. Hillston (Eds.), Proceedings of the Third Workshop on Process Algebras and Performance Modelling (special issue), Comput. J. 38 (7) (1995) 530–541.

[32] H. Hermanns, M. Siegle, Bisimulation algorithms for stochastic process algebras and their BDD-based implementation, in: J.P. Katoen (Ed.), Proceedings of the ARTS'99, Lecture Notes in Computer Science, vol. 1601, Springer, Berlin, 1999, pp. 244–264.

[33] U. Herzog, Formal description, time and performance analysis. A framework, in: T. Härder, H. Wedekind, G. Zimmermann (Eds.), Entwurf und Betrieb Verteilter Systeme, IFB 264, Springer, Berlin, 1990, pp. 172–190.

[34] U. Herzog, A concept for graph-based stochastic process algebras, generally distributed activity times and hierarchical modelling, in: M. Ribaudo (Ed.), Proceedings of the Fourth Workshop on Process Algebras and Performance Modelling, Universita di Torino, CLUT, 1996, pp. 1–20.

[35] U. Herzog, M. Rettelbach (Eds.), Proceedings of the Second Workshop on Process Algebra and Performance Modelling, University of Erlangen-Nürnberg, IMMD, 1994.

[36] J. Hillston, A compositional approach to performance modelling, Ph.D. Thesis, University of Edinburgh, Edinburgh, 1994.

[37] J. Hillston, V. Mertsiotakis, A simple time scale decomposition technique for SPAs, in: S. Gilmore, J. Hillston (Eds.), Proceedings of the Third Workshop on Process Algebras and Performance Modelling (special issue), Comput. J. 38 (7) (1995) 566–577.

[38] J. Hillston, F. Moller (Eds.), Proceedings of the First Workshop on Process Algebra and Performance Modelling, University of Edinburgh, Edinburgh, 1993.

[39] C.A.R. Hoare, Communicating sequential processes, Commun. ACM 21 (8) (1978) 666–677.

[40] C.A.R. Hoare, Communicating Sequential Processes, Prentice-Hall, Englewood Cliffs, NJ, 1985.

[41] ISO, LOTOS: A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour, ISO, Geneve, 1989.

[42] P. Kanellakis, S. Smolka, CCS expressions, finite state processes, and three problems of equivalence, Inform. and Comput. 86 (1990) 43–68.

[43] J.P. Katoen, D. Latella, R. Langerak, E. Brinksma, Partial order models for quantitative extensions of LOTOS, Comput. Networks ISDN Systems 30 (9–10) (1998) 925–950.

[44] J.G. Kemeny, J.L. Snell, Finite Markov Chains, Springer, Berlin, 1976.

[45] J.A. Manas, T. de Miguel, J. Salvachua, Tool support to implement LOTOS specifications, Comput. Networks ISDN Systems 25 (7) (1993) 815–839.

[46] K. Marzbani, Hierarchische Beschreibung und Analyse von Kommunikationssystemen mittles graphbasierten Prozeßalgebren, Master's Thesis, Universität Erlangen-Nürnberg, Erlangen, 1998.

[47] V. Mertsiotakis, Approximate analysis methods for stochastic process algebras, Ph.D. Thesis, Universität Erlangen–Nürnberg, Erlangen, 1998.

[48] V. Mertsiotakis, M. Silva, Throughput approximation of decision free processes using decomposition, in: Proceedings of the Seventh International Workshop on Petri Nets and Performance Models, St. Malo, IEEE Computer Society Press, Silverspring, MD, 1997, pp. 174–182.

[49] R. Milner, A Calculus of Communicating Systems, Lecture Notes in Computer Software, vol. 92, Springer, Berlin, 1980.

[50] R. Milner, Communication and Concurrency, Prentice-Hall, London, 1989.

[51] R. Paige, R. Tarjan, Three partition refinement algorithms, SIAM J. Comput. 16 (6) (1987) 973–989.

[52] G.D. Plotkin, A structured approach to operational semantics, Technical Report DAIMI FM-19, Computer Science Department, Aarhus University, 1981.

[53] C. Priami, Stochastic π-calculus with general distributions, in: M. Ribaudo (Ed.), Proceedings of the Fourth Workshop on Process Algebras and Performance Modelling, Universita di Torino, CLUT, 1996, pp. 41–57.

[54] C. Priami (Ed.), Proceedings of the Sixth Workshop on Process Algebras and Performance Modelling, University of Verona, September 1998.

[55] M. Rettelbach, Stochastische Prozeßalgebren mit zeitlosen Aktivitäten und probabilistischen Verzweigungen, Ph.D. Thesis, Universität Erlangen–Nürnberg, Erlangen, 1996.

[56] M. Ribaudo (Ed.), Proceedings of the Fourth Workshop on Process Algebras and Performance Modelling, Universita di Torino, CLUT, 1996.

[57] M. Siegle, Compact representation of large performability models based on extended BDDs, in: Fourth International Workshop on Performability Modeling of Computer and Communication Systems (PMCCS4), Williamsburg, VA, September 1998, pp. 77–80.

[58] M. Siegle, D. Kraska, M. Simon, B. Wentz, Analyse des Erlanger Klinikkommunikationssystems mit Hilfe von Leistungsmessungen, in: E. Greiser, M. Wischnewsky (Eds.), 43rd Jahrestagung der Deutschen Gesellschaft für Medizinische Informatik, Biometrie und Epidemiologie (GMDS), Bremen, September 1998, MMV Medien & Medizin Verlag, Munich, p. CD-ROM C24.

[59] M. Siegle, B. Wentz, A. Klingler, M. Simon, Neue Ansätze zur Planung von Klinikkommunikationssystemen mittels stochastischer Leistungsmodellierung, in: R. Muche, G. Büchele, D. Harder, W. Gaus (Eds.), 42nd Jahrestagung der Deutschen Gesellschaft für Medizinische Informatik, Biometrie und Epidemiologie (GMDS), Ulm, September 1997, MMV Medien & Medizin Verlag, Munich, pp. 188–192.

[60] H.A. Simon, A. Ando, Aggregation of variables in dynamic systems, Econometrica 29 (1961) 111–138.
[61] W.J. Stewart, Introduction to the Numerical Solution of Markov Chains, Princeton University Press, Princeton, NJ, 1994.

Holger Hermanns studied applied mathematics at the University of Bordeaux I, France, and computer science at the University of Erlangen–Nürnberg, Germany, where he received his diploma degree in 1993 (with honours). He received a Ph.D. degree from the Department of Computer Science, University of Erlangen–Nürnberg, in 1998 (with honours). Currently he is with the Systems Validation Centre, University of Twente, the Netherlands. He has been active in the area of algebraic foundations of specification and evaluation methods for performance prediction. His main research interests include compositional performance modelling, state space compression, and model checking of performance models.

Ulrich Herzog received all his degrees in electrical engineering from the University of Stuttgart. In 1964, he joined the Institute for Switching Techniques and Data Processing at the University of Stuttgart, working in the area of telephone switching systems and teletraffic research. He then spent two years in the Teleprocessing System Optimisation Group at IBM Thomas J. Watson Research Centre. Since 1976, he has been full Professor at the University of Erlangen–Nürnberg. Since 1981, he has held the chair on computer architecture and performance evaluation. His current research and teaching interests are architecture and performance evaluation of computer systems, and communication networks. In particular, he is involved in projects on system design methodology, the integration of process algebras and performance modeling, and rapid prototyping of real-time systems.

Ulrich Klehmet studied mathematics at the Ernst–Moritz–Arndt University, Greifswald from where he received his degree in 1973. From 1990 to 1995 he worked at the University of Erlangen–Nürnberg. During that time he was dealing with performance modelling and parameter optimisation of the German fieldbus protocol Profibus. In 1995 he received his Ph.D. Currently he is a researcher at the University of Erlangen–Nürnberg in the group of Prof. Ulrich Herzog. His research interests include stochastic process algebras and their application to performance and dependability evaluation.

Vassilis Mertsiotakis received a degree in computer science from the University of Erlangen–Nürnberg in 1993. From 1993 until 1998 he was research assistant at the Computer Science Department of the University of Erlangen–Nürnberg where he participated at the project SFB182 *Multiprocessor and network configurations*. He defended his doctoral thesis on approximate analysis methods for stochastic process algebras in 1998. In the same year he joined Lucent Technologies, Switching and Access Systems Group, R & D. He is involved in software architecture of access networks for POTS, ISDN, PRA-ISDN, xDSL, and FITL.

Markus Siegle studied computer science at the University of Stuttgart from 1984 to 1989, graduating with the German engineering degree. He received a Fulbright scholarship which allowed him to pursue his studies at North Carolina State University where he earned a Masters degree in 1990. From 1990 to 1995 he worked as a researcher at the University of Erlangen–Nürnberg in the group of Prof. Ulrich Herzog from where he received the doctorate degree in 1995. Markus Siegle is currently working on stochastic modelling and verification at the University of Erlangen–Nürnberg.