Compositional Performance Modelling with the TIPPtool

H. Hermanns, U. Herzog, U. Klehmet, V. Mertsiotakis, M. Siegle

Universität Erlangen-Nürnberg, IMMD 7, 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.

1 Introduction

Process algebras are an advanced concept for the design of distributed systems. Their basic idea is to systematically construct complex systems from smaller building blocks. Standard composition operators allow one to create highly modular and hierarchical specifications. An algebraic framework supports the comparison of different system specifications, process verification and structured analysis. Classical process algebras (e.g. CSP [20], CCS [26] or LOTOS [5]) describe the functional behaviour of systems, but no temporal aspects.

Starting from [17], we developed an integrated design methodology by embedding stochastic features into process algebras, leading to the concept of *Stochastic Process Algebras* (SPA). SPAs allow to specify and investigate both functional and temporal properties, thus enabling early consideration of all major design aspects. Research on SPA has been presented in detail in several publications, e.g. [11, 19, 4, 28, 15, 8] and the 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. 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 high degree of maturity, supporting compositional modelling and analysis of complex distributed systems via a user-friendly graphical front end.

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 [15]. Beside support for analysis of functional aspects, the tool offers algorithms for the numerical analysis of the underlying stochastic process. Exact and approximate evaluation techniques are provided for stationary as well as transient analysis. As a very advanced feature, the tool supports semi-automatic compositional reduction of complex models based on equivalence-preserving reduction. This enables the tool to handle large state spaces (the running example given here is small, due to didactical reasons and limited space).

Among related work, the PEPA Workbench [9] is another tool for performance evaluation, where Markov chain models are also specified by means of a process algebra.

The paper is organised as follows: In Sec. 2, we summarise the theoretical background of stochastic process algebras. Sec. 3 gives an overview of the tool's components. All aspects of model specification are discussed in Sec. 4, and analysis algorithms are the subject of Sec. 5. The paper concludes with Sec. 6.

2 Foundations of Stochastic Process Algebras

2.1 Process algebras

Classical process algebras 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, with some distinguishing features.

Specifications are built from *processes* which may perform *actions*. The description formalism is *compositional*, i.e. it allows to build highly modular and hierarchical system descriptions using composition operators. A parallel composition operator is used to express concurrent execution and possible synchronisation of processes. Another important operator realises *abstraction*: Details of a specification which are internal 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 transformation can be profitably employed in order to reduce the complexity of the system. This can also be performed in a compositional way, by replacing system parts through behaviourally equivalent but minimised representations.

Let us exemplify the basic constructs of process algebras on a simple queueing system. It consists of an arrival process $Arrival$, a queue with finite capacity, and a $Server$. First, we model an arrival process as in infinite sequence of incoming arrivals ($arrive$), each followed by an enqueue action (enq), using the *prefix* operator '$;$'.

$$Arrival := arrive;\, enq;\, Arrival$$

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.

$$\begin{aligned} Queue_0 &:= enq;\, Queue_1 \\ Queue_i &:= enq;\, Queue_{i+1} \;[]\; deq;\, Queue_{i-1} \qquad 1 \leq i < max \\ Queue_{max} &:= deq;\, Queue_{max-1} \end{aligned}$$

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

$$Server := deq;\, serve;\, Server$$

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

$$System := Arrival \quad |[enq]| \quad Queue_0 \quad |[deq]| \quad Server$$

A formal semantics associates each language expression with an unambiguous interpretation, a *labelled transition system* (LTS). It is obtained by structural operational rules which define for each language expression a specific LTS as the unique semantic model. Fig. 1 (top) shows the semantic model for our example queueing system (assuming 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 labelled with the corresponding action. Since we assume that we are not interested in the inter-

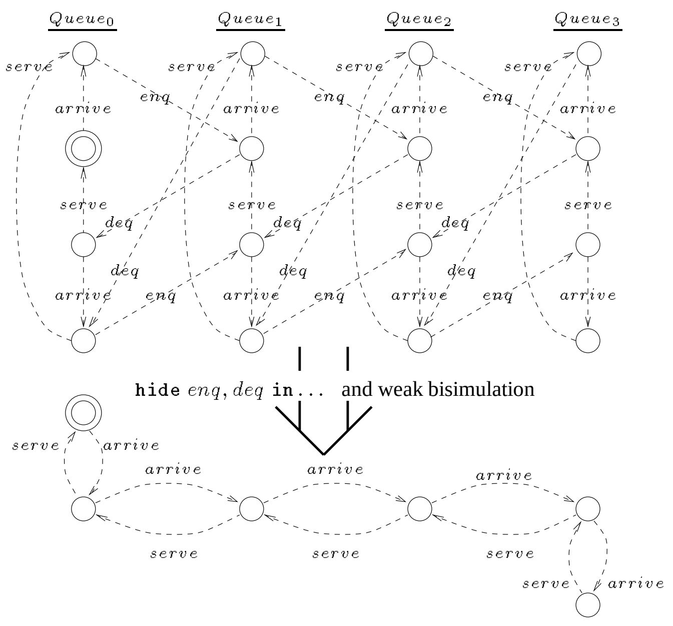

Fig. 1. Semantic model, hiding and reduction

nal 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:

$$\texttt{hide}\ enq,\ deq\ \texttt{in}\ System$$

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. 1 (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 [26]. Fig. 1 (bottom) shows an LTS, which is weakly bisimilar to the one on top (where all enq- and deq-actions have been replaced by τ). It may be surprising that the resulting LTS has 6 and not 4 states (we assumed $max = 3$). This is due to the fact that the arrival of a customer and its enqueueing into the queue are separate actions, so that one more arrival is possible if the queue is already full. Likewise, dequeueing 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

Stochastic Process Algebras (SPA) are aimed at the integration of qualitative-functional and quantitative-temporal aspects in a single specification and modelling approach [11]. In order to achieve this integration, temporal information is attached to actions, in the form of continuous random variables, representing activity durations. The additional time information in the resulting LTS 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 an event sequence)

Let us give a SPA specification for the above queueing 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 duration happen as soon as possible, therefore they are called *immediate* actions. In our example, enqueueing and dequeueing is assumed to happen without any relevant delay, thus enq and deq are immediate.

$$Arrival := (arrive, \lambda);\ enq;\ Arrival$$
$$Server := deq;\ (serve, \mu);\ Server$$

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. Fig. 2 depicts the labelled transition system associated with this model (again assuming $max = 3$). 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. They correspond to states of a Continuous Time Markov Chain (CTMC) (shown at the bottom of the figure) 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 [14]. Abstraction from the two immediate actions enq and deq is an essential prerequisite for unambiguously determining the Markov chain underlying this specification. If, say, enq is hidden, we can 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 (indeed, abstraction rules out any further synchronisation, since τ is not allowed to appear in the list '$\ldots$' of synchronising actions of a parallel composition operator '$|[\ldots]|$'.)

2.3 Bisimulation and Compositional analysis

As illustrated in the running example, the notion of bisimulation is important. 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 [26]. 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 [23]. *Markovian bisimulation* reflects lumpability and bisimulation on timed transitions, by imposing constraints on actions and rates, see [15, 19] for details. Weak Markovian bisimulation additionally allows abstraction from internal immediate actions, in analogy to ordinary weak bisimulation [16]. Equivalences are defined in terms of states and transitions, i.e. on the level of the LTS. It is possible to characterise their distinguishing power on the level of the language by means of *equational laws* [13].

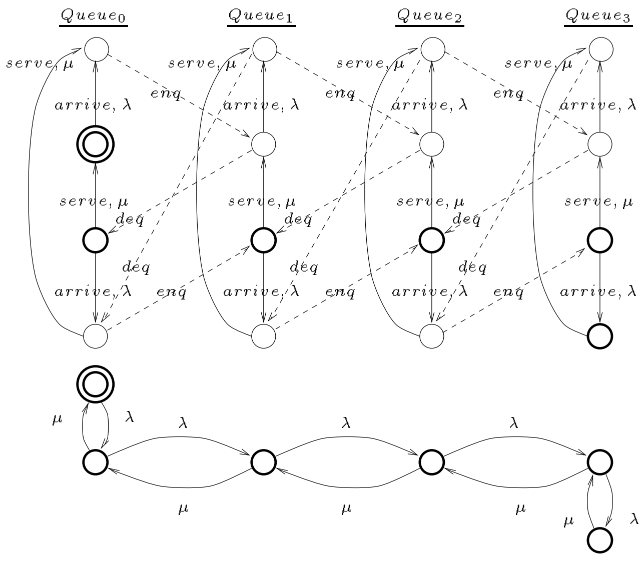

Fig. 2. Top: The LTS for the example queueing system. Bottom: The corresponding CTMC

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. Practically important, such equivalences allow *compositional reduction* techniques, where the size of a component's state space may be reduced, without affecting any significant property of the whole model. Compositional reduction has successfully been applied to a variety of systems, see e.g. [7] for an impressive industrial case study.

We return to our queueing example in order to illustrate compositional reduction. We now consider a queueing system with one Poisson arrival process, two queues and two servers. We build this system from the same components, i.e. processes $Arrival$, $Queue$ and $Server$ are defined as above. The system is now:

$$\begin{aligned} System := Arrival \ \ |[enq]| \ \ ((&Queue_0 \ \ |[deq]| \ \ Server) \ \ ||| \\ (&Queue_0 \ \ |[deq]| \ \ Server)) \end{aligned}$$

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 reduced to 22 states and 48 transitions. However, reduction can also be performed in a compositional fashion: The subsystem consisting of one queue-server pair has 8 states, which can be reduced down to 5 states. Combining both (reduced) queue-server pairs, we obtain 25 states which can be reduced down to 15 states (this reduction step mainly exploits symmetry of the model). If this reduced system is combined with the arrival process, we get 30 states which can again be reduced to 22 states. This concept of compositional reduction is illustrated in Fig. 3, where the size of the state space and the number of transitions are given for each reduction 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

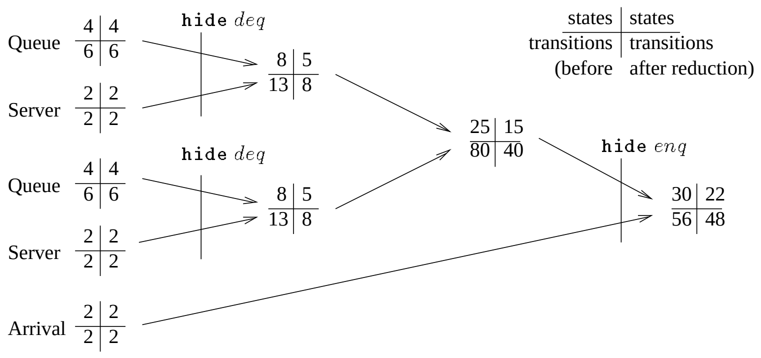

Fig. 3. Compositional reduction of the example queueing system

course, neither queue is full, in which case the behaviour is deterministic). As a consequence, the Markov chain underlying this specification is 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 Sec. 4, we will follow the latter path.

3 Tool overview

The TIPPtool consists of several interacting components. Specifications can be created with an editor which is part of the tool. A parser checks specifications for syntactic correctness. Another component is responsible for the generation of the LTS and for the reduction of the LTS according to different bisimulation equivalences (currently, four bisimulation algorithms are provided). The user can specify performance and reliability measures to be calculated (such as state probabilities, throughputs and mean values). Experiments can be specified, providing information about activity rates which may vary. 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. The tool provides several numerical solution methods for the steady state analysis as well as for transient analysis of Markov chains. The results of an experiment series are presented graphically with the tool PXGRAPH from UC Berkeley.

The export module of the tool provides interfaces to three other tools, PEPP [12], TOPO [24], and ALDEBARAN [6]. The former interface is based on a special semantics for SPAs which generates stochastic task graphs [18], 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 in order to bridge to the powerful bisimulation equivalence algorithms of the tool ALDEBARAN.

4 Model specification

In this section, we explain the details of the specification language supported by the TIPPtool. It is an extension of basic LOTOS [5], 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).

The available operators are listed in Table 1; Action prefix, choice, hiding and parallel composition (with synchronisation) have already been used in Sec. 2. 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 [15]). 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 which 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. Process instantiations $P[a_1, \ldots, a_n]$ resemble the invocation of procedures in procedural programming languages.

Name	Syntax	Name	Syntax
timed action prefix	$(a, r);\ P$	inaction	`stop`
immediate action prefix	$a;\ P$	successful termination	`exit`
choice	P `[]` Q	enabling	P `>>` Q
parallel composition	P `|[`$a_1, \ldots, a_n$ `]|` Q	disruption	P `[>` Q
– pure interleaving	P `|||` Q	process instantiation	$P[a_1, \ldots, a_n]$
hiding	`hide` $a_1, \ldots, a_n$ `in` P		

Table 1. Basic syntax. P, Q are behaviour expressions, a_i are action names.

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 (the above specification is a simple example for a data dependent specification, since parameter i governs the synchronisation capabilities of $Queue_i$). We have incorporated the possibility to describe data dependencies 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 [5]. This is highly beneficial, in order to conveniently describe complex dependencies. Data values are declared in the form `!`$value$, attached to an action, where $value$ 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. These basic ingredients can be combined to form different types of inter-process communication (note that inter-process communication is currently only implemented for immediate actions), among them:

- **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) ; ...`
  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 ; stop |[`a`]| `a`?x:int ; `P` |[`a`]| `a`?y:int ; ...`

- **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) ; ...`

To illustrate the power of these language elements, we return to our running example of a queueing 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. 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 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 initialised with the value 100, a value larger than any real queue population (note that the tool provides the possibility to specify choice alternatives which depend on conditions ‘`[ ...] ->`’).

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

5 Analysing a specification

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 straight-forward fashion. The resulting LTS 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, depending on whether equivalence checking algorithms are selected or not.

Once the LTS is generated, it can be used for 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 a 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.

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 LTS [10, 19]. 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. 4). 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.

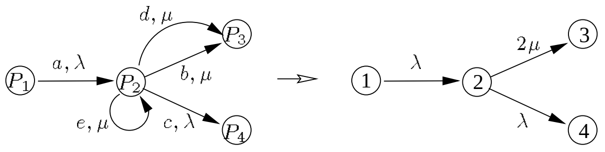

Fig. 4. Deriving a Markov chain

Models with both timed and immediate actions: As discussed in Sec. 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 (*vanishing* states) and states without such transitions (*tangible* states). 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 immediate transitions still has to be taken. One possible solution is to weight all alternatives with equal probabilities. The standard method used for eliminating immediate transitions 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. However, [29] showed that this technique should be applied with care in the SPA context, essentially because a non-determinstic 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. 2. If non-deterministic alternatives only lead (via some internal, immediate steps) into equivalent states, equivalence-preserving transformations allow to remove this non-determinism, see Sec. 5.3.

In the TIPPtool, standard numerical solution algorithms (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 two efficient approximation methods are realised. Both approaches are based on decomposition. *Time Scale Decomposition (TSD)* is a method which can exploit the *Near Complete Decomposability* (NCD) property of many Markov chains. *Response Time Approximation (RTA)* works on the specification level rather than on the CTMC level [25].

5.3 Compositional model reduction

Equivalence relations such as (weak) Markovian bisimulation, introduced in Sec. 2.3, are beneficial both for eliminating immediate transitions, and for reducing models with very large state spaces. Both effects can be achieved by means of the same strategy. For a given specification, say $System$, the key idea is to compute an equivalent specification, $System'$, which is minimal (with respect to the number of states). Performance analysis can then be based on the minimised specification which is obtained by a *partition refinement* strategy: The bisimulation algorithm computes 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 algebraic terms, a fixed-point is reached. This fixed-point is the desired result.

This general strategy can be realised by means of very efficient algorithms [21, 27]. For specifications which do not contain timed transitions, we implemented Kanellakis 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 [3] 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 bismulation (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. However, in some cases hiding of immediate transitions is 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. 3. In this way, specifications with very large state spaces become tractable. In the TIPPtool, compositional minimisation is supported in an elegant way. By dragging the mouse inside the editor 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 specification is generated automatically, where the selected component has been replaced by the minimised representation.

6 Conclusion

In this paper, we have presented the status quo of the TIPPtool. 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, see e.g. [18, 22]. However, they all suffer from the problem that general distributions lead to intractable stochastic processes. 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 [8], using *stochastic automata* as a model based on Generalised Semi-Markov processes.

We are currently building a prototype tool for graphical model specification as an easy-to-use front-end for users who are not familiar with the syntax of the TIPPtool's specification language. 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, based on Binary Decision Diagrams [30].

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

References

1. *Workshops on Process Algebras and Performance Modelling*, 1993 Edinburgh, 1994 Erlangen, 1995 Edinburgh, 1996 Torino, 1997 Twente, 1998 Nice.
2. M. Ajmone Marsan, G. Balbo, and G. Conte. *Performance Models of Multiprocessor Systems*. MIT Press, 1986.
3. C. Baier. Polynomial time algorithms for testing probabilistic bisimulation and simulation. In *Proc. CAV'96*. LNCS 1102, 1996.
4. M. Bernardo and R. Gorrieri. Extended Markovian Process Algebra. In *CONCUR '96*.

5. T. Bolognesi and E. Brinksma. Introduction to the ISO specification language LOTOS. *Computer Networks and ISDN Systems*, 14:25–59, 1987.
6. M. Bozga, J.-C. Fernandez, A. Kerbrat, and L. Mounier. Protocol verification with the ALDEBARAN toolset. *Int. J. Softw. Tools for Techn. Transf.*, 1(1/2):166–184, 1997.
7. G. Chehaibar, H. Garavel, L. Mounier, N. Tawbi, and F. Zulian. Specification and Verification of the Powerscale Bus Arbitration Protocol: An Industrial Experiment with LOTOS. In *Formal Description Techniques IX*. Chapmann Hall, 1996.
8. P.R. D'Argenio, J-P. Katoen, and E. Brinksma. An algebraic approach to the specification of stochastic systems. In *Programming Concepts and Methods*. Chapman and Hall, 1998.
9. S. Gilmore and J. Hillston. The PEPA Workbench: A Tool to Support a Process Algebra-Based Approach to Performance Modelling. In *7th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, Wien, 1994.
10. N. Götz. *Stochastische Prozeßalgebren – Integration von funktionalem Entwurf und Leistungsbewertung Verteilter Systeme*. PhD thesis, Universität Erlangen–Nürnberg, April 1994.
11. N. Götz, U. Herzog, and M. Rettelbach. Multiprocessor and distributed system design: The integration of functional specification and performance analysis using stochastic process algebras. In *Tutorial Proc. of PERFORMANCE '93*. LNCS 729.
12. F. Hartleb and A. Quick. Performance Evaluation of Parallel Programms — Modeling and Monitoring with the Tool PEPP. In Proc. "Messung, Modellierung und Bewertung von Rechen- und Kommunikationssystemen", p. 51–63. Informatik Aktuell, Springer, 1993.
13. H. Hermanns. *Interactive Markov Chains*. PhD thesis, Universität Erlangen-Nürnberg, 1998.
14. H. Hermanns, U. Herzog, and V. Mertsiotakis. Stochastic Process Algebras as a Tool for Performance and Dependability Modelling. In *Proc. of IEEE Int. Computer Performance and Dependability Symposium*, p. 102–111, 1995. IEEE Computer Society Press.
15. H. Hermanns, U. Herzog, and V. Mertsiotakis. Stochastic Process Algebras - Between LOTOS and Markov Chains. *Computer Networks and ISDN Systems*, 30(9-10):901–924, 1998.
16. H. Hermanns, M. Rettelbach, and T. Weiß. Formal characterisation of immediate actions in SPA with nondeterministic branching. In *The Computer Journal* [1], 1995.
17. U. Herzog. Formal Description, Time and Performance Analysis. A Framework. In *Entwurf und Betrieb Verteilter Systeme*. Springer, Berlin, IFB 264, 1990.
18. U. Herzog. A Concept for Graph-Based Stochastic Process Algebras, Generally Distributed Activity Times and Hierarchical Modelling. [1], 1996.
19. J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.
20. C.A.R. Hoare. *Communicating Sequential Processes*. Prentice-Hall, 1985.
21. P. Kanellakis and S. Smolka. CCS Expressions, Finite State Processes, and Three Problems of Equivalence. *Information and Computation*, 86:43–68, 1990.
22. J.P. Katoen, D. Latella, R. Langerak, and E. Brinksma. Partial Order Models for Quantitative Extensions of LOTOS. *Computer Networks and ISDN Systems*, 1998. to appear.
23. J.G. Kemeny and J.L. Snell. *Finite Markov Chains*. Springer, 1976.
24. J.A. Manas, T. de Miguel, and J. Salvachua. Tool Support to Implement LOTOS Specifications. *Computer Networks and ISDN Systems*, 25(7), 1993.
25. V. Mertsiotakis. *Approximate Analysis Methods for Stochastic Process Algebras*. PhD thesis, Universität Erlangen–Nürnberg, 1998. to appear.
26. R. Milner. *Communication and Concurrency*. Prentice Hall, London, 1989.
27. R. Paige and R. Tarjan. Three Partition Refinement Algorithms. *SIAM Journal of Computing*, 16(6):973–989, 1987.
28. C. Priami. Stochastic π-calculus. [1], 1995.
29. M. Rettelbach. *Stochastische Prozeßalgebren mit zeitlosen Aktivitäten und probabilistischen Verzweigungen*. PhD thesis, Universität Erlangen–Nürnberg, 1996.
30. M. Siegle. Technique and tool for symbolic representation and manipulation of stochastic transition systems. TR IMMD 7 2/98, Universität Erlangen-Nürnberg, 1998.