UNE IMP EHE CWI QMIPS INR UER ZAR UTO

Quantitative
Modeling
In
Parallel
Systems

A Stochastic Process Algebra Based Modelling Tool

Holger Hermanns and Vassilis Mertsiotakis
University of Erlangen-Nürnberg, IMMD VII

D W1.T2.2b-2.v1

22 September 1995

Distribution Level: Public

Approved by: U. Herzog

Abstract The incorporation of time into classical process algebras aiming at the integration of functional design and performance analysis has become very popular recently. There are many ways to include time in process algebras. Current research in this area concentrates mainly on the annotation of actions with exponentially distributed random variables. This allows us to make use of a large repertoire of analysis algorithms. This paper presents some first results of ongoing work which aims at providing a tool for the efficient performance evaluation and functional analysis of computer and communication systems based on the stochastic process algebra paradigm. It provides facilities for model creation, reachability analysis, as well as several numerical algorithms for the solution of the underlying Markov chain and the computation of characteristic performance measures.

The QMIPS Project: The QMIPS project is a collaborative research project supported by the CEC as ESPRIT-BRA project no 7269. It is being carried out by the following organisations: CWI (Amsterdam), EHEI (University of Paris V), Imperial College (London), INRIA (Sophia-Antipolis), University of Erlangen, University of Newcastle, University of Torino and University of Zaragoza.

1 Introduction

Stochastic process algebras (SPA) have been introduced as an extension of classical process algebras, like CCS [21] or CSP [17], with timing information aiming mainly at the integration of functional design and quantitative analysis of computer systems [8]. Time is represented by associating exponentially distributed random variables to activities in the model.

As in classical process algebras, the main characteristic of SPA is constructivity, i.e. the ability to describe complex systems as a composition of several smaller ones. This feature together with a powerful abstraction mechanism to hide internal structures form the basis of the successful application of SPA for describing and analysing systems of various domains [11, 12, 13].

TIPP-tool (*timed processes and performability evaluation*) is a prototype modelling tool for creating and evaluating stochastic process algebra models of parallel and distributed systems. It supports a LOTOS-oriented input language [1] and apart from facilities to apply functional analysis based on reachability analysis, it comes up with a few numerical solution modules for the stationary as well as transient analysis of the Continuous Time Markov Chain (CTMC) underlying a TIPP-specification. The input language and the emphasis on numerical evaluation rather than analytical evaluation represent the main difference to the *PEPA Workbench*, another modelling tool for SPA that relies upon symbolic evaluation of the underlying Markov chain [6].

The structure of the TIPP-tool is shown in Fig. 1. Model descriptions are syntactically and semantically analysed using a parser written in the functional programming language *Standard ML*. The parser has been generated using the tools *ML-YACC* and *LEXGEN* from the New Jersey Standard ML toolkit. The use of this programming language has shown to be especially useful for implementing the formal semantics of TIPP, the calculus the tool is based on [14].

The state space that has been derived formally serves as a base for further reduction into a CTMC. For the steady state analysis of the underlying CTMC we are using a variety of numerical algorithms. TIPP-tool supports also transient analysis by providing methods to compute the mean time to absorption of an absorbing Markov chain or the transient state probabilities. For the latter, a refined randomization scheme is provided, similar to the method adopted by Lindemann in [20]. All methods work on a sparse matrix data structure [18].

The result of numerical analysis is usually a vector with state probabilities. In order to obtain more sophisticated and expressive results the user can specify measures. This is done via weights that are assigned to states that match a certain regular expression the user has to specify. Experiment series are also supported by allowing rates to be symbolic variables. The specification of the model as well as the measures and experiments is supported through a Tcl/Tk-graphical user interface.

The model specification language of the tool is explained in detail in the following section. Section 3 describes the analysis modules. We conclude the paper with a discussion on intended extensions to our tool.

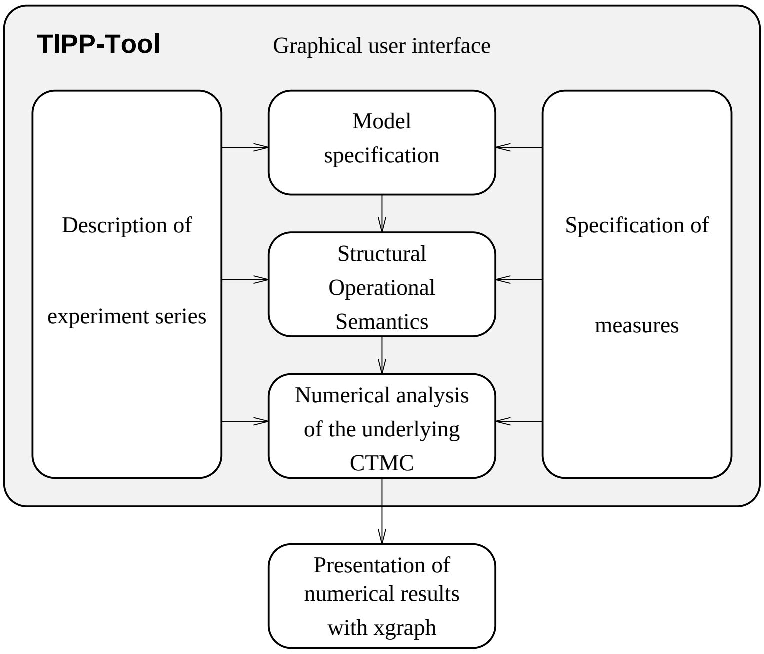

Figure 1: Tool components

2 Model Description Language

In this section we introduce the specification language of our tool designed to be suitable for describing the behaviour of a system on a high level [22]. This description can be refined step by step, in order to obtain a more accurate model of a system.

The language is closely related to LOTOS, but enriched with stochastic timing information. This is achieved by the convention that a certain random distribution function is associated with every activity specifying its duration. The set of random distributions that can be handled with TIPP is restricted to exponentially distributed ones. This implies that every term in our language can be interpreted as a high level description of a Continuous Time Markov Chain (CTMC). Exponential distributions can be characterized by a single parameter, drawn from the set of positive reals. Therefore, each activity of the system possesses a specific name and a parameter, called rate, characterizing its duration.

The major building blocks of a system specification are behaviour expressions that perform actions. Behaviour expressions can be composed out of smaller ones by combining them via a language operator. The set of operators that can be used in the input language is listed in Table 1. Their intuitive meaning is as follows:

Inaction: The process `stop` denotes an inactive process which cannot perform any action.

Action Prefixing: The process $(a, \lambda); P$ (or $(a, \lambda).P$) carries out the action with name a after a duration which is exponentially distributed with parameter λ. Thus the mean time of the delay until the action occurs is $1/\lambda$. The process subsequently behaves as process P. Internal actions, denoted by (i, λ), are unobservable for the environment, as far as functionality is concerned. A delay with mean time $1/\lambda$, however, can be witnessed. Actions that happen without delay will be discussed in Section 4.2.

Choice: The process P `[]` Q (or P + Q) represents a system which may behave either as P or as Q. The decision is taken as soon as either P or Q performs any action. The duration until this choice is made is obviously the minimum of the concerned random distributions. As a matter of fact, due to our restriction to exponential distributions, the according random variable is given by the sum of the involved rates.

Parallelism: Concurrency and cooperation of behaviour expressions can be modelled by specific operators. P `|||` Q denotes that P and Q proceed independently. In contrast P `||` Q implies that both proceed in complete synchrony, except for possible internal actions. The general case, where parts of the behaviour must proceed synchronously and other parts are independent is specified by P `|[` $a_1, \ldots, a_n$ `]|` Q. The processes P and Q proceed independently and concurrently with any action whose action name is not contained in the list of synchronizing actions $a_1, \ldots, a_n$. However, for action names a_j, the processes P and Q must cooperate in order to perform the action. Such actions can be completed only when they are enabled in both P and Q. Thus both partners have to synchronize on this action before proceeding. The rate of a synchronized action is given by the multiplication of the rates of the involved actions. This implies that each of the partners has a scalable influence on the mean duration of a synchronizing action. In practice many cooperations consist of a passive and an active component (e.g. workload/machine). Passivity is expressed by the rate 1 — the neutral element

Name	Syntax
inaction	`stop`
action prefix	
– observable	$(a, r);P$ or $(a, r).P$
– unobservable	$(i, r);P$ or $(i, r).P$
choice	P `[]` Q or P + Q
parallel composition	
– pure interleaving	P `|||` Q
– full synchronization	P `||` Q
– general case	P `|[`$a_1, \ldots, a_n$ `|]` Q
hiding	`hide` $a_1, \ldots, a_n$ `in` P
process instantiation	$P[a_1, \ldots, a_n](r_1, \ldots, r_m)$

Table 1: Syntax for behaviour expressions. P and Q are place-markers for any behaviour expressions. $a, a_1, \ldots, a_n$ stand for action names. The rates $r, r_1, \ldots, r_m$ can be instantiated by arbitrary positive real numbers.

of multiplication.

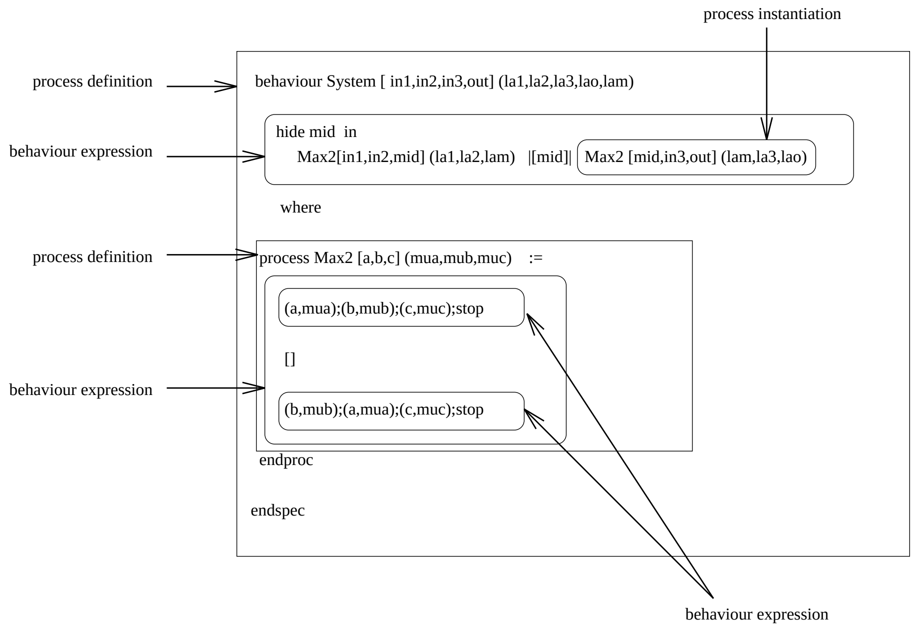

Figure 2: Process definition

Hiding: The process `hide` $a_1, \ldots, a_n$ `in` P behaves as the process P except that any actions within the list $a_1, \ldots, a_n$ are *hidden*, meaning that their name is not visible. Instead they appear with the name i and can be regarded as an internal delay by the component. Hiding is important for a structured design methodology: At every structural level the actions that do not interact with the environment can be hidden in order to be invisible for external components.

Process Instantiation: Process instantiations $P[a_1, \ldots, a_n](r_1, \ldots, r_m)$ may occur within behaviour expressions and resemble the invocation of procedures in procedural programming languages such as PASCAL. A process instance is generated by replacing the list of formal action names and rates in the corresponding process definition `process` $P[\ldots](\ldots)$:= ... `endproc` by actual action names $a_1, \ldots, a_n$ and rates $r_1, \ldots, r_m$.

A complete specification is constituted by a number of process definitions, each describing a process scheme with generic action names and action rates within a behaviour expression. The typical structure of a process definition is given in Figure 2. It defines a behaviour expression that itself is built out of smaller behaviour expressions by applying any of the language operators.

The behaviour of a specification is formally defined in a structural operational style (see [14] for more details). A Markovian Labelled Transition System (MLTS) is used as domain for the semantic model, consisting of nodes representing processes and arcs between them. These arcs are labelled with pairs of actions and rates[1]. Each of these arcs $\square \xrightarrow{a,\lambda} \square$ symbolizes that the first process can evolve to the second by carrying out the action a with rate λ. Figure 3 for example shows an MLTS, that is the semantic model of the specification in Figure 2.

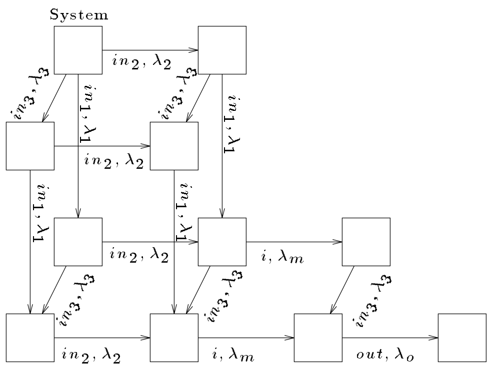

Figure 3: Markovian Labelled Transition System

3 Analysis of the Underlying CTMC

Quantitative analysis is based on the MLTS derived formally from a process specification as shown in the last section. Here, both labels of an arc, denoting the action name and rate, are regarded as being significant, as are the arc multiplicities. For any SPA model with finite state space the underlying CTMC can be derived directly by associating a state with each node of the MLTS. The transitions of the CTMC are the amalgamation of all the arcs joining the nodes, and the transition rate is the sum of the individual activity rates. The action name information is not incorporated into the CTMC representation but it is often crucial in defining the measures to be extracted from a model.

Characteristic performance and dependability measures based on equilibrium behaviour are extracted via the steady state solution of the CTMC, i.e. solving the matrix equation

$$\pi Q = 0 \qquad \text{subject to} \qquad \sum_i \pi_i = 1$$

using standard numerical techniques, where Q denotes the infinitesimal generator matrix of the CTMC. The steady state probabilities π_i can be interpreted as the relative frequency with which the behaviour corresponding to each process state i is exhibited by the system. From these values it is straightforward to obtain high-level measures such as throughput, or utilization. Unfortunately the only sufficient condition for a unique equilibrium solution is strong connectivity of the MLTS.

[1] An additional third label is used for some subtle management purposes, cf. [14].

If the latter condition is not fulfilled or steady state measures do not give a sufficient insight into the system behaviour, it is possible to carry out transient analysis. Standard algorithms like randomization or calculation of higher moments may be used to find time-related properties of systems. Transient analysis is central to the integration of qualitative and quantitative aspects of model analysis as it allows quantitative implications of qualitative properties of a system to be investigated [11].

3.1 Solution Methods

Our tool offers a small set of solution algorithms for steady state analysis as well as transient analysis of the underlying CTMC. For the former we provide the following methods: *LU-factorization*, *power method* and *Gauß-Seidel* method. The first two methods can be used efficiently for systems with less than 1000 states, while the latter method enables us to analyse chains with up to 100000 states. All the algorithms are implemented in C and work on a data structure for sparse matrices that came along with the Sparse1.3 package of K.S. Kundert [18]. Apart from basic matrix subroutines this package contains a LU-factorization solver. We extended this library with two iterative solution methods for steady state analysis.

For transient analysis we implemented the moment method for the computation of the mean time to absorption. Time-related state probabilities can be computed using the randomization method that we integrated into this library as well. The main problem with the randomization method is the computation of Poisson-probabilities for stiff differential equation systems. Therefore, we adopted a similar refined randomization method based on a numerically stable subroutine for the computation of Poisson-probabilities as proposed in [4] and has been used by several other tool developers [2, 20].

3.2 Specification of Measures

The result of numerical analysis is the state probability distribution vector. This is usually a too detailed measure and not well-suited for getting insight into the system behaviour. More high-level measures can be computed out of this vector. For this, the user has to specify a number of measures using the dialog box shown in Fig. 4.

TIPP-tool supports two types of measures. *Count measures* that are typically throughputs of actions the user has to specify and *state measures*. The latter ones are specified using regular expressions the user has to supply for each state measure. The measure is computed by summing over all states that match the specified expression. Typical state measures are: blocking probabilities, probability of deadlock states, probability of being in a repair phase, etc.

3.3 Description of Experiment Series

It is very often desirable to use symbolic names for activity rates instead of fixed numerical values. In that case, the user has to describe an experiment using the dialog box shown in Fig. 5. Here, the user can specify the actual numerical values of the activity rates. In each

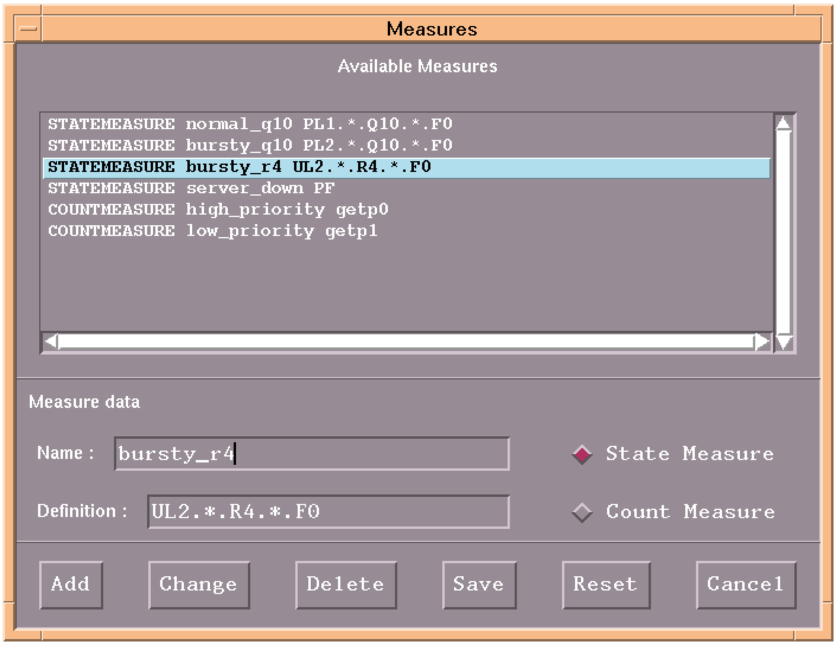

Figure 4: Measure definition dialog box

experiment there can be one activity rate which may be varied. The user needs to specify the first value, the last value, and a stepwidth.

3.4 Presentation of Numerical Results

If the user has specified an experiment using a variable rate, the tool automatically analyses the model for each value within the variable range and the results can afterwards be presented graphically. For this purpose we employ an existing plotting program called `xgraph` that is freely available from UC Berkeley. TIPP-tool transforms the numerical results into an appropriate input format (see Fig. 6).

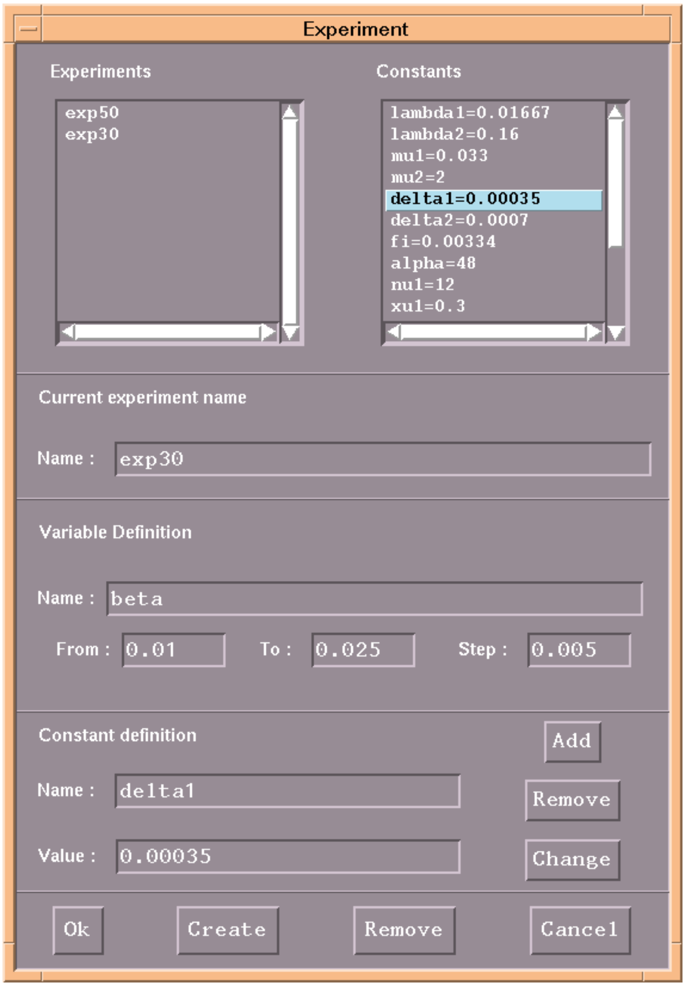

Figure 5: Experiment dialog box

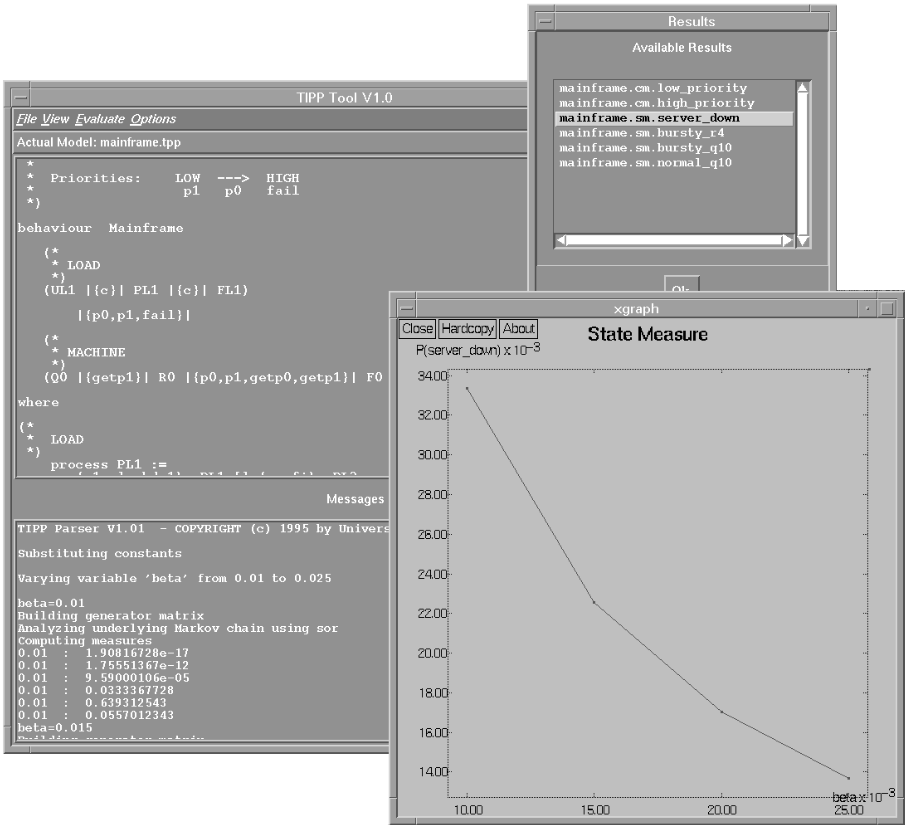

Figure 6: Graphical user interface

3.5 Case Studies

Even though the present paper is our first publication about the TIPP-tool, it has already been used successfully for a number of case studies that have been published in various conference proceedings and journals. The case studies we investigated include:

- data transfer with multiple send-and-wait communications [9]
- multiprocessor and task graph modelling [8]
- electronic mail system [11]
- alternating bit protocol [13]
- robot control system [7]
- multiprocessor mainframe with software failures [12, 16]

All these models were built up out of small modules that were composed to obtain the entire system. The multiprocessor mainframe example additionally contains concepts for the decomposition of the generated state space along the structure given by the modular process description.

We computed various types of performability measures such as response time, availability of components, failure probabilities, mean execution times and many more.

4 Extensions

In this chapter we will outline some extensions that shall be included in the future in order to enhance the modelling convenience. Although these extensions take place mainly on the level of the input language they have a strong impact on the underlying theory.

4.1 Value Passing

The concept of process instantiation that is already integrated in our tool allows us to parameterize processes over action names. Value passing can be regarded as a generalization of this concept. Here, values are attached as parameters to processes as well as actions in the system description. Thus, it gets a lot more easier to describe systems at a lower level of abstraction, where very often buffers have to be described and their respective scheduling strategy. This can become even in the simplest case of a FIFO-buffer a hard and errorprone task if value passing is not provided, as the following example should demonstrate:

```
process Q0[in,out] :=
        (in,1) ; Q1[in,out]
endproc
process Q1[in,out] :=
        (in,1) ; Q2[in,out] [] (out,1) ; Q0[in,out]
endproc
```

```
process Q2[in,out] :=
        (in,1) ; Q3[in,out] [] (out,1) ; Q1[in,out]
endproc
⋮
process Q_max[in,out] :=
        (out,1) ; Q_max-1[in,out]
endproc
```

Such processes can become arbitrarily complex as soon as more sophisticated scheduling strategies as FIFO have to be implemented. Often the insertion of jobs into the queue is performed by different processes possibly using different action names. This results in process descriptions with a huge number of choices. For these cases the concept of parametric processes and value passing is a very attractive means to help the user in specifying such systems.

Our current language has to be extended by a number of constructs in order to realize value passing, among them are [1]:

data types The precondition for value passing is the support of at least one basic data type, like `integer` or `boolean`. Abstract data type specification languages like ACT ONE that is used in Full LOTOS may be employed to represent more abstract data structures.

value declarations A value declaration has the form `!(`*value*`)` and is usually attached to an action. *value* may be a concrete value, a variable or a value expression if its data type provides any operators.

variable declarations The counterpart to value declarations are variable declarations, which have the form `?(`*variable*`:t)`. *variable* is the name of the variable and `t` its data type. Actions `a` attributed with variable declarations describe a set of actions `a<v>` for all values in the domain of type `t`.

interprocess communication This is achieved by composing two or more processes in parallel that offer the same structured action on which the participating processes are required to cooperate. Three different types of interaction may occur (cf. [1] for more details):

- **value matching** The synchronizing actions are combined with value declarations. If the supplied values are equal synchronization is possible.
- **value passing** Combination of value declaration and variable declaration results in value passing. The effect is that a value is transmitted from one process to the other.
- **value generation** The only remaining case is the one where actions are combined with variable declarations of the **same type** attached to them. Superposing another process in parallel that offers a value of the required type yields a form of multicast communication.

conditional constructs The flexibility of the description is increased if it is possible to describe behaviours that depend on conditions on values, e.g. an action should be enabled if a certain value (buffer size) is below a threshold. Such conditions are usually in square brackets and attached to actions as this example shows:

```
(send,1)!(5) ; stop [] (recv,1)?x:nat [x>0] ; stop
```

generalized choice With this construct it is possible to specify choice among behaviours `B(v)` for all values of `v` by writing

```
choice x:nat [] B(x)
```

parametric processes Processes can be parameterized by a list of variable declarations that is attached to the formal action name list. The syntax for such a parameterized process specification is:

`process` P`[`$a_1, \ldots, a_n$`](`$x_1 : t_1, \ldots, x_n : t_n$`) :=` $\ldots$`endproc`

Example

```
process Queue[in,out](n:nat, max:nat) :=
  (in,1)?x:nat [n+x lt max+1] ; Queue[in,out](n+1,max)
 []
  (out,1)!n [n gt 0] ; Queue[in,out](n-1,max)
endproc
```

The binary operators `lt` and `gt` denote "less than" and "greater than" respectively. We can see clearly how elegant it becomes to describe components like queues using value passing and parametric processes.

4.2 Immediate Actions: Disabling and Enabling

Currently, the algebraic foundations our tool strictly rely on exponentially distributed durations. Recently, TIPP has been extended with *immediate actions* that happen instantaneously if enabled [15]. Their key feature is that they can be eliminated out of the transition system under certain circumstances identified in the paper. Therefore, the extended expressiveness of the language does not influence the treatment of the underlying stochastic process. By virtue of this independence, immediate actions can be integrated into the TIPP-tool without affecting the analysis modules.

The elimination procedure for immediate actions can be implemented in two different ways. First, special graph algorithms for equivalence checking can be used for this purpose operating on the transition system. Another possibility is the syntactic manipulation of the process description in order to automatically eliminate immediate actions. The advantage of the latter is that the elimination happens on the fly, during the construction of the transition system. Both methods rely on algebraic properties that have been established for the enriched calculus.

Once immediate actions have been integrated in our tool it should become easy to enrich our language with two constructs taken from LOTOS, namely *process enabling* and *disabling*. The former construct, denoted by $P \gg Q$, models the sequential execution of two processes P and Q. By means of the disabling operator P `[>` Q the preemption of process P by Q is modelled. The process P can perform arbitrary actions as long as Q, typically representing a timer, does not perform any action. But as soon as Q executes an action, P is stopped and control is handed over to process Q.

The reason why we need immediate actions first before adding these operators to our language lies in the semantics of these operators (cf. [1]). Both make use of internal control activities that are meant to have no impact on the temporal behaviour of the system. Immediate, internal actions allow us to model those control activities in a proper way.

4.3 Measure Definition

As mentioned above, in order to extract more expressive results out of the state probabilities the user has to provide information about what he is interested in. The implemented method associates weights with states that match a certain regular expression specified by the user. This relies on the fact that the formal semantics identifies process terms with states of the transition system. Therefore it is possible to use a regular expression like

```
.*.Q3[in,out].*.
```

in order to refer to the states of the above example where three places of the queue are in use. In the future this will not be as easy as it is nowadays. The inclusion of value passing as sketched in Section 4.1 is one reason for this development. In addition, various research groups are working on different techniques to reduce the state space of the semantic model, cf. [5]. Most of them try to exclude redundant information from the model. As a consequence, groups of states are amalgamated to form a single macro state. Unfortunately, the automation of these techniques implies that those macro states are no longer identifiable by regular expressions. Therefore, new ways to define measures have to be developed.

One approach to solve this problem arises from the fact that process algebras describe the *behaviour* of a system whereas the labelling of states is not significant. States are considered to be different if their subsequent behaviour differs. *Temporal logic* is an adequate means to classify states according to their subsequent behaviour. In [10] the temporal logic CTL^* [3] is used to define classes of states by means of their behaviour. Measures are defined by associating weights with each of these classes.

Even though it is not clear to the authors if the expressive power of CTL^* is really necessary for this purpose, the future development seems to force a shift from state based definition of measures towards behaviour oriented definition. This implies that efficient techniques for investigating the behaviour have to be integrated into the tool. For this purpose it is necessary to take a close look at algorithms used for model checking of temporal logic formulae.

5 Conclusions

In this paper we presented the stochastic process algebra based modelling tool TIPP-tool. Models of parallel and distributed systems can be described in a text-oriented process algebraic specification language. The model description language that is currently supported by our implementation is Basic LOTOS oriented. The main difference to classic process algebra based specification languages is that apart from pure functional behaviour the user can specify timed behaviour by attaching exponentially distributed durations to each activity in the model. This allows the integration of qualitative and quantitative system analysis into one comprehensive design methodology.

We described briefly the various modules of our current implementation. It contains facilities for syntactic and semantic checking of the specification and reachability analysis of the model based on the formal semantics of our process algebra. The resulting state space can be checked for deadlocks or reduced into a CTMC, suitable for further numerical analysis. The various tool components – written partially in C and Standard ML – are integrated by a graphical user interface that uses `Expectk`, an enhanced version of the command language Tcl/Tk which supports dialog with existing interactive applications [19].

One of our main research aims is to integrate some concepts of Full LOTOS to make the input language more comfortable. In Section 4 we showed some first ideas to this direction and outlined some conceptual difficulties arising from this extension. It is obvious that the development of our tool is by far not yet finished and a lot of new features will be added in the near future. Nevertheless, the experience we made already with our tool has been documented in several publications and helped in getting new ideas for future extensions.

References

[1] T. Bolognesi and E. Brinksma. Introduction to the ISO Specification Language LOTOS. In P.H.J. van Eijk, C.A. Vissers, and M. Diaz, editors, *The Formal Description Technique LOTOS*, pages 23–73, Amsterdam, 1989. North-Holland.

[2] J. Couvillion, R. Freire, R. Johnson, W. D. Obal, A. Qureshi, M. Rai, W. H. Sanders, and J. E. Tvedt. Performability Modeling with UltraSAN. *IEEE Software*, 8(5):69–80, 1991.

[3] E.A. Emerson and J.Y. Halpern. "sometimes" and "not never" revisited: on branching versus linear time temporal logic. *Journal of the ACM*, 33:151–178, 1986.

[4] B.L. Fox and P.W. Glynn. Computing Poisson Probabilities. *Communications of the ACM*, 31(4):440–445, 1988.

[5] S. Gilmore and J. Hillston (editors). Special issue on Process Algebras and Performance Modelling. *The Computer Journal*, 1995. to appear.

[6] S. Gilmore and J. Hillston. The PEPA Workbench: A Tool to Support a Process Algebra-Based Approach to Performance Modelling. In G. Haring and G. Kotsis, edi-

tors, *7th Int. Conference on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 353–368, Wien, May 1994.

[7] S. Gilmore, J. Hillston, R. Holton, and M. Rettelbach. Specifications in Stochastic Process Algebra for a Robot Control Problem. *accepted for: International Journal of Production Research*, 1995.

[8] 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 the 16th International Symposium on Computer Performance Modelling, Measurement and Evaluation, PERFORMANCE 1993*. Springer LNCS 729, 1993.

[9] N. Götz, U. Herzog, and M. Rettelbach. *Verteilte Systeme - Grundlagen und zukünftige Entwicklungen aus der Sicht des Sonderforschungsbereichs 182 "Multiprozessor- und Netzwerkkonfigurationen"*, chapter TIPP - Einführung in die Leistungsbewertung von verteilten Systemen mit Hilfe von Prozeßalgebren, pages 509–531. BI-Wiss.-Verlag, 1994.

[10] H. Hermanns. Leistungsvorhersage von Verhaltensbeschreibungen mittels temporaler Logik. In Reinhard Gotzhein and Jan Bredereke, editors, *Formale Beschreibungstechniken für verteilte Systeme*. Universität Kaiserslautern, Fachbereich Informatik, 1995.

[11] H. Hermanns, U. Herzog, J. Hillston, V. Mertsiotakis, and M. Rettelbach. Stochastic Process Algebras: Integrating Qualitative and Quantitative Modelling. Technical Report 11/94, Universität Erlangen–Nürnberg, IMMD VII, Martensstr. 3, 91058 Erlangen, May 1994.

[12] H. Hermanns, U. Herzog, and V. Mertsiotakis. Stochastic Process Algebras as a Tool for Performance and Dependability Modelling. In *Proc. of IEEE International Computer Performance and Dependability Symposium*, Erlangen, April 1995. IEEE Computer Society Press.

[13] H. Hermanns, V. Mertsiotakis, and M. Rettelbach. Performance Analysis of Distributed Systems Using TIPP – a Case Study. In J. Hillston and R. Pooley, editors, *Proc. of the 10th U.K. Performance Engineering Workshop for Computer and Telecommunication Systems*, pages 131–144. Department of Computer Science, University of Edinburgh, 1994.

[14] H. Hermanns and M. Rettelbach. Syntax, Semantics, Equivalences, and Axioms for MTIPP. In U. Herzog and M. Rettelbach, editors, *Proc. of the 2nd Workshop on Process Algebras and Performance Modelling*, pages 71–88, Erlangen-Regensberg, July 1994. IMMD, Universität Erlangen-Nürnberg.

[15] H. Hermanns and M. Rettelbach. Formal Characterisation of Immediate Actions in an SPA with Non-Deterministic Branching. *The Computer Journal*, special issue on Process Algebras and Performance Modelling, 1995. to appear.

[16] U. Herzog and V. Mertsiotakis. Applying Stochastic Process Algebras to Failure Modelling. In U. Herzog and M. Rettelbach, editors, *Proc. of the 2nd Workshop on Process Algebras and Performance Modelling*, pages 107–126, Regensberg/Erlangen, July 1994. Arbeitsberichte des IMMD, Universität Erlangen-Nürnberg.

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

[18] S.K. Kundert and A. Sangiovanni-Vincentelli. A Sparse Linear Equation Solver. Technical report, University of California, Berkeley, 1988.

[19] D. Libes. *Exploring Expect: A Tcl-based Toolkit for Automating Interactive Programs*. O'Reilly & Associates, 1994.

[20] C. Lindemann. Employing the Randomization Technique for Solving Stochastic Petri Net Models. In A. Lehmann and F. Lehmann, editors, *Proc. 6. GI/ITG Conf. on Modelling, Measurement and Evaluation of Computing Systems (MMB '91)*, pages 306–319, Munich, Germany, 1991. Springer.

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

[22] R. Studt. Syntactical and Semantical Analysis of TIPP processes. Internal study, Universität Erlangen-Nürnberg, IMMD VII, March 1995.