Compositional Performance Modelling with the TIPPtool

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 an infinite sequence of incoming arrivals (*arrive*), each followed by an enqueue action (*enq*), using the prefix operator `;`.

\[
\text{Arrival} ::= \text{arrive}; \text{enq}; \text{Arrival}
\]

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.

\[
\begin{align*}
\text{Queue}_0 & ::= \text{enq}; \text{Queue}_1 \\
\text{Queue}_i & ::= \text{enq}; \text{Queue}_{i+1} | \text{deq}; \text{Queue}_{i-1} & 1 \leq i < \text{max} \\
\text{Queue}_{\text{max}} & ::= \text{deq}; \text{Queue}_{\text{max}-1}
\end{align*}
\]

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

\[
\text{Server} ::= \text{deq}; \text{serve}; \text{Server}
\]

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

\[
\text{System} ::= \text{Arrival} | [\text{enq}] | \text{Queue}_0 | [\text{deq}] | \text{Server}
\]

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 \( \text{max} = 3 \)). 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-