Specifications in Stochastic Process Algebra for a Robot Control Problem

Stephen Gilmore Jane Hillston*

Department of Computer Science,

The University of Edinburgh

Robert Holton

Department of Computing Science,

University of Aberdeen

Michael Rettelbach

Informatik VII,

Universität Erlangen-Nürnberg

Abstract

We present a novel approach to specification of dynamic systems. This approach, a stochastic extension of process algebra, facilitates quantitative, or performance, analysis, in addition to qualitative analysis. For unreliable systems this integrated approach encourages the investigation of the impact of functional characteristics on the performance of the system. Throughout the paper details of the stochastic process algebra are made concrete via an example: a robot control problem.

Two specifications are presented of this problem. The first, an idealisation, does not represent the possibility of failures. The second models both failures and recoveries. Each is solved to obtain performance measures for the system.

*Corresponding Author, address: University of Edinburgh, Kings Buildings, Edinburgh EH9 1NN; tel: 0131 650 5188; fax: 0131 667 7209

1 Introduction

Classical process algebras such as CCS (Milner, 1989) and CSP (Hoare, 1985) are perfectly suited for modelling systems composed of concurrently operating agents which meet when it is necessary to exchange information and attend to their own work at other times. The inherent non-determinism of such systems is also reflected in a process algebra model. An equational theory can be used to decide questions such as equivalence under observation of competing designs of the system. Models are built up from small components in a structured fashion.

A description in classical process algebra can never be used to answer questions about the *effective performance* of a system. This is simply because timing information is not part of the model: actions such as synchronisation are considered to be performed instantaneously and thus only the relative ordering of actions can be ascertained.

The stochastic process algebras (SPAs) such as PEPA (Hillston, 1994) and TIPP (Götz et al., 1993) enrich classical process algebras with timing information to allow questions about system performance to be answered. Typical questions would involve identifying potential performance bottlenecks; determining the throughput of the system; or measuring the utilisation of critical components.

Stochastic process algebras retain all of the pleasing virtues of classical process algebras: a foundation in a precise mathematical semantics; the existence of an equational theory of observable behaviour; and the inclusion of structuring facilities for composing small models in order to be able to build up large ones.

The control problem modelled here is pallet-filling. The task for a robot arm is to repeatedly select blocks from a conveyor belt and group them on a pallet which is removed when it becomes full. The problem is complicated by the possibility of failures. Performance measures are calculated both for the idealised, infallible system and for the version of the system with failures and recoveries.

This represents a new application of performance analysis. However the increasing use of such workcells within flexible manufacturing systems serves to emphasise the importance of both quantitative and qualitative analysis of robots before they become operational.

Structure of this paper

The next section provides a self-contained introduction to stochastic process algebras. Their syntax and semantics are presented together with some small illustrative examples. An informal description of the pallet-filling problem is given in Section 3. The first formal specification of the pallet-filling problem appears in Section 4. The second specification follows in Section 5. The SPA tools used in this work are outlined in Section 6. These tools were used to detect (syntactic and semantic) flaws in the specifications and to compute the performance results which also appear in Section 6. The performance results show the degradation in performance due to the failures. A comparison with other approaches to the specification and analysis of concurrent systems follows in Section 7. Finally, conclusions and further work are presented.

2 Stochastic process algebras

Performance modelling is concerned with the capture and analysis of quantitative information about the dynamic behaviour of systems. An abstract representation is used to capture the essential characteristics of the system and this model is analysed to answer questions about the work carried out by the system within a temporal framework. Stochastic processes have been used extensively for performance modelling as random variables provide a good representation of the variability inherent in many systems. In many cases these stochastic models are assumed to be Markov processes for pragmatic reasons of efficient solution—this is also the case for stochastic process algebras.

The stochastic process algebra languages enable the modelling of stochastic processes in a structured fashion. A small but powerful set of combinators is used to build up complex behaviour from simpler behaviour. The combinators are familiar: sequential composition; parallel composition; selection and encapsulation. We will explain each in its turn below. The language which we are using may be thought to be a subset of either PEPA or TIPP, we do not use features from either of these languages which are not available in the other. Formal operational semantics are available for both of these languages but they are not presented here (Hillston, 1994; Hermanns and Rettelbach, 1994).

Sequential composition: A component of a stochastic process algebra model may have purely sequential behaviour, repeatedly undertaking one activity after another and eventually returning again to the beginning of its behaviour. A simple example is a conveyor belt intended to supply one entity at a time. Each entity is brought into position and only when it has been successfully delivered will the belt initiate the positioning of another entity.

Since we wish to extract performance measures for our models we wish to record the trundle rate of the belt (say, μ). However, we may recognise that the rate of some activities, for example the delivery, are outside the control of this component. Such activities will be carried out in a synchronisation in which this component will play a passive rôle. A distinguished symbol, $\top$, is used to record that the belt is passive with respect to this activity.

$$\mathit{Belt} \stackrel{\mathit{def}}{=} (\mathit{trundle}, \mu).(\mathit{deliver}, \top).\mathit{Belt}$$

This simple specification has recorded that arrivals and deliveries must occur in alternation and that an arrival must occur first. The time for any particular arrival is obtained by drawing from a (negative) exponential distribution with parameter μ (mean $1/\mu$).

Selection: A choice between two possible behaviours is represented as the sum of the possibilities. For example, we may wish to elaborate the behaviour of the belt to respond to external queries, indicating whether a block is currently in position or not. We now differentiate between the two possible states, representing them as separate derivatives, *Belt* and *Belt*$'$, representing the empty and full belt respectively. The activities of the empty belt are to trundle as before, and to respond to a query indicating that it is empty. Similarly the activities of the full belt are to facilitate the delivery or to respond to a query indicating that it is full. Since the belt *responds* to these queries it is passive with respect to these activities.

$$\begin{array}{rcl}
\mathit{Belt} & \stackrel{\mathit{def}}{=} & (\mathit{trundle}, \mu).\mathit{Belt}' + (\mathit{belt}_{\mathit{empty}}, \top).\mathit{Belt} \\
\mathit{Belt}' & \stackrel{\mathit{def}}{=} & (\mathit{deliver}, \top).\mathit{Belt} + (\mathit{belt}_{\mathit{full}}, \top).\mathit{Belt}'
\end{array}$$

The specification records that responding to a query about its current state cannot alter the state of the belt: if it is empty it remains empty, etc.

In classical process algebras the selection operator is non-deterministic choice. In a stochastic process algebra the selection operator represents pre-emptive selection with re-sampling. The continuous nature of the probability distributions used ensures that the possibility of selecting both choices simultaneously is completely disallowed. Thus a sum will behave as either one summand or the other. When an action has more than one possible outcome it is represented by a choice of separate actions, one for each possible outcome. If it is possible for several of these actions to be enabled simultaneously their activity rates are chosen to reflect their relative probabilities.

Parallel composition: We have already anticipated that the belt will be working cooperatively with some other component within the system. For example, there may be some workcell with which the belt must interact. This workcell will wait for an entity to become available whilst repeatedly querying the belt. When an entity is available it will adopt a suitable position to effect delivery, process it and then pass it out of the workcell.

$$\begin{aligned}
\mathit{Workcell}_1 &\stackrel{\mathit{def}}{=} (\mathit{belt}_{\mathit{empty}}, q_b).\mathit{Workcell}_1 + (\mathit{belt}_{\mathit{full}}, q_b).\mathit{Workcell}_2 \\
\mathit{Workcell}_2 &\stackrel{\mathit{def}}{=} (\mathit{position}, p).(\mathit{deliver}, d).\mathit{Workcell}_3 \\
\mathit{Workcell}_3 &\stackrel{\mathit{def}}{=} (\mathit{process}, r).(\mathit{export}, s).\mathit{Workcell}_4 \\
\mathit{Workcell}_4 &\stackrel{\mathit{def}}{=} (\mathit{reset}, t).\mathit{Workcell}_1
\end{aligned}$$

In order to query the state of the belt, or effect the delivery, the workcell must *synchronise* with the belt. In contrast, the other activities of the workcell, such as processing, are carried out independently of the belt. Synchronisation over given activities is reflected in the parallel combinator by the *synchronisation set*, $S_1 = \{\mathit{belt}_{\mathit{empty}}, \mathit{belt}_{\mathit{full}}, \mathit{deliver}\}$ in this case.

$$\mathit{System} \stackrel{\mathit{def}}{=} \mathit{Belt} \underset{S_1}{\bowtie} \mathit{Workcell}_1 \qquad S_1 = \{\mathit{belt}_{\mathit{empty}}, \mathit{belt}_{\mathit{full}}, \mathit{deliver}\}$$

Note that the query activity was represented by two separate actions in the workcell, $\mathit{belt}_{\mathit{empty}}$ and $\mathit{belt}_{\mathit{full}}$, corresponding to the two possible outcomes. Only one of these actions will ever be enabled since they must be synchronised with the belt which only enables one of them at any given time.

A degenerate form of the parallel combinator is formed when two components behave completely independently, without any synchronisation between them. For example, if we consider a pallet which takes finished entities away from the workcell it will synchronise with the workcell on the export activity, but have no interaction at all with the belt. Thus we arrive at an extended version of the system:

$$\begin{aligned} \mathit{System} &\stackrel{\mathit{def}}{=} (\mathit{Belt} \parallel \mathit{Pallet}) \underset{S_2}{\bowtie} \mathit{Workcell}_1 \\ &\qquad\qquad S_2 = \{\mathit{belt}_{\mathit{empty}}, \mathit{belt}_{\mathit{full}}, \mathit{deliver}, \mathit{export}\} \end{aligned}$$

This pure parallel composition can be thought of as synchronisation over the empty set: $(\mathit{Belt} \underset{\emptyset}{\bowtie} \mathit{Pallet})$. Now we may see the pure parallel composition introduced earlier as only a degenerate form of synchronisation over the empty set.

Encapsulation: Finally we wish to hide some activities, making them private to the component or components involved. The duration of the activities is unaffected. On elaborating the workcell model we may wish to differentiate between a plan element and a processing element which communicate via signals, as well as synchronising on the delivery and exporting actions.

$$\begin{aligned} \mathit{Plan}_1 &\stackrel{\mathit{def}}{=} (\mathit{belt}_{\mathit{empty}}, q_b).\mathit{Plan}_1 + (\mathit{belt}_{\mathit{full}}, q_b).\mathit{Plan}_2 \\ \mathit{Plan}_2 &\stackrel{\mathit{def}}{=} (\mathit{signal}, \delta).(\mathit{deliver}, \top).\mathit{Plan}_3 \\ \mathit{Plan}_3 &\stackrel{\mathit{def}}{=} (\mathit{signal}, \top).(\mathit{export}, \top).\mathit{Plan}_1 \end{aligned}$$

$$\begin{aligned} \mathit{Proc}_1 &\stackrel{\mathit{def}}{=} (\mathit{signal}, \top).(\mathit{position}, p).(\mathit{deliver}, d).\mathit{Proc}_2 \\ \mathit{Proc}_2 &\stackrel{\mathit{def}}{=} (\mathit{process}, r).(\mathit{signal}, \delta).(\mathit{export}, s).\mathit{Proc}_3 \\ \mathit{Proc}_3 &\stackrel{\mathit{def}}{=} (\mathit{reset}, t).\mathit{Proc}_1 \end{aligned}$$

$$\mathit{Workcell} \stackrel{\mathit{def}}{=} (\mathit{Plan}_1 \underset{L}{\bowtie} \mathit{Proc}_1)/\{\mathit{signal}\} \qquad L = \{\mathit{signal}, \mathit{deliver}, \mathit{export}\}$$

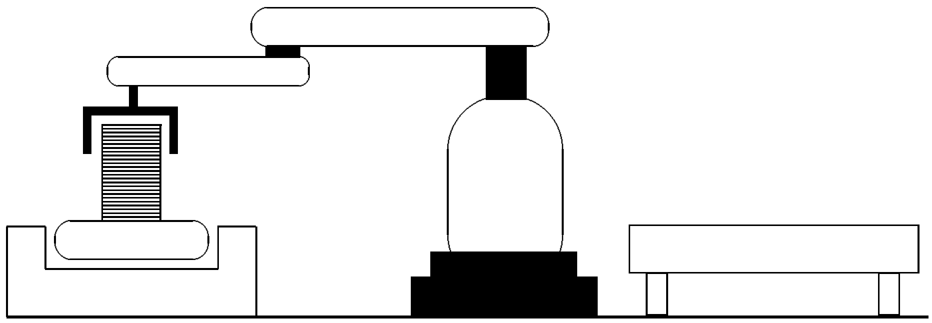
Here the interaction between the plan and the processing element is considered private, so the signalling action is hidden, making it inaccessible to any other components.

3 The pallet-filling problem

The problem of pallet-filling arises in many application areas in factory control, manufacturing and chemical processing. A robot arm receives from a supply some items (we term them *blocks*) and batches these onto a nearby pallet. When the pallet is full it is cleared away, to be replaced by another which is to be filled in the same way.

We shall consider the supply process to be a conveyor belt. We shall also consider the process which replaces a full pallet by an empty one to be *mechanical*, not requiring human intervention. The complete robot workcell therefore may be viewed as an abstraction of a self-contained flexible manufacturing process.

The schematic below shows the robot arm about to pick a block from the end of the conveyor belt (on the left) in order to carry it over to be placed on top of the empty pallet (on the right).



The robot arm, the conveyor belt and the pallet replacement mechanism are all logically separate components in the workcell and the responsibility for their correct synchronisation and control rests with the workcell *plan*. This structure will be reflected in our specifications of the workcell: the belt, the robot arm, and the pallet will never communicate with each other directly, they will only respond to instructions from the plan.

Before going on to consider the necessary synchronisations in some detail we will briefly outline the means of communication between the plan and the other three workcell components.

3.1 Communication within the workcell

The conveyor belt provides a line which it keeps low until a block is available. When a sensing device detects the presence of a block the line goes high and stays high until the block is removed. The plan interrogates the conveyor belt line in order to obtain information about the availability of blocks.

The communication between the plan and the robot takes place via the *controller* of the robot arm. The plan cannot instruct the manipulator directly. The controller and the plan use (possibly contiguous) dual-ported memory locations. The controller writes the current position information and the plan writes the desired position information. Communication is by vector reads and writes, giving the data for all the joints of the manipulator in a single memory access.

The controller also prevents direct access to the end-effector of the robot arm. Directions to hold or release will be buffered by the controller, which passes on the appropriate instruction to the end-effector.

Finally, the plan communicates with the pallet in order to determine whether spaces are still available. The pallet provides a line connected to a sensing device. The line goes high when the pallet is full and remains high until it is flushed.

3.2 Synchronisation issues

There are many points of interaction between the components of the workcell. As we have stated, these interactions are orchestrated by the workcell plan and the robot arm controller never communicates directly with either the conveyor belt or the pallet. Thus the SPA components representing the robot arm, the belt and the pallet proceed independently of each other, although each must synchronise with the plan.

Interactions between the plan and the belt take the form of queries and instructions, initiated by the plan and both represented as synchronisations. Interaction between the plan and the pallet is analogous. Similarly, direct communication takes place between the plan and the end-effector controller, which then passes on instructions to the end-effector via a private synchronisation. This privacy is ensured by the hiding operator.

Since communication between the robot arm joint controller and the plan takes place via a dual-ported memory location, the memory is represented as a distinct component in the SPA model which synchronises with both the controller and the plan. The controller relays instructions to the joints via a private synchronisation, analogous to that used between the end-effector and its controller.

4 A first specification without failures

After some delay, a block will be available. After this the conveyor belt delivers other blocks with pauses in between. The belt may be interrogated in order to determine whether or not it is full (i.e. has a block available). These tests do not cause a change of state.

$$\begin{aligned} \mathit{Belt} &\stackrel{\mathit{def}}{=} (\mathit{trundle}, \mu).\mathit{Belt}' + (\mathit{belt}_{\mathit{empty}}, \top).\mathit{Belt} \\ \mathit{Belt}' &\stackrel{\mathit{def}}{=} (\mathit{deliver}, \top).\mathit{Belt} + (\mathit{belt}_{\mathit{full}}, \top).\mathit{Belt}' \end{aligned}$$

A pallet must initially be empty. Blocks are added one by one until the pallet is full. Then it is flushed back to its empty state. There are m locations on the pallet. As with the conveyor belt, the pallet may be interrogated about its state without a change of state.

$$\begin{aligned} \mathit{Pallet}_i &\stackrel{\mathit{def}}{=} (\mathit{pallet}_{\mathit{space}}, \top).\mathit{Pallet}_i + (\mathit{accept}, \top).\mathit{Pallet}_{i+1} \qquad i < m \\ \mathit{Pallet}_m &\stackrel{\mathit{def}}{=} (\mathit{pallet}_{\mathit{full}}, \top).\mathit{Pallet}_m + (\mathit{flush}, \lambda).\mathit{Pallet}_0 \end{aligned}$$

Now the robot. It is built from a controller and a manipulator synchronising on a set of activities. These activities position the joints and activate the end effector and write joint positions to a global memory that will be specified later. All of the synchronisation set activities are private to the robot and so they are all hidden.

$$\mathit{Robot} \stackrel{\mathit{def}}{=} (\mathit{Controller} \underset{S_1}{\bowtie} \mathit{Manipulator})/S_1$$

where $S_1 = \{\, \mathit{position}, \mathit{effect} \,\}$.

The controller is made up of sub-controllers for the joints (*JC*) and the end-effector (*EC*).

$$\mathit{Controller} \stackrel{\mathit{def}}{=} (\mathit{JC} \parallel \mathit{EC})$$

The behaviour of the two sub controllers is sketched below. The joint-controller reads the current position and directly requests the position from the global memory, performs the necessary calculations and starts the movement of the joints by transmitting movement data. The controller for the end-effector also may be interrogated about its state.

$$\begin{aligned} \mathit{JC} &\stackrel{\mathit{def}}{=} (\mathit{vread}_J, R_J).(\mathit{vread}_P, R_P).(\mathit{calculate}, c).(\mathit{position}, p).\mathit{JC} \\ \mathit{EC} &\stackrel{\mathit{def}}{=} (\mathit{hold}, \top).(\mathit{effect}, \top).\mathit{EC}' + (\mathit{release}, \top).\mathit{EC} + (\mathit{opened}, \top).\mathit{EC} \\ \mathit{EC}' &\stackrel{\mathit{def}}{=} (\mathit{hold}, \top).\mathit{EC}' + (\mathit{release}, \top).(\mathit{effect}, \top).\mathit{EC} + (\mathit{closed}, \top).\mathit{EC}' \end{aligned}$$

A manipulator is a composition of n identical joints and an end effector. Joints have a relatively sophisticated behaviour for a small component. They must continually report their current position to the global memory: this is an asynchronous activity, taking place independently from receiving instructions about future positions. Thus a joint is the composition of a receiving component and a reporting component. The receipt of future position information (*position*) is synchronous and can also happen during movement. The level of abstraction we chose within this example does not require a detailed report of each of the positions the joints hold. It is therefore sufficient to solely distinguish the continuous report of these positions during movement and while idling in one case (vwrite_J) and the report of the end position after a completed movement in the other case ($\mathit{vwrite\_ready}_J$).

The rôle of the end effector is to be switched from its active state to its inactive state whenever the controller requires.

$$\begin{aligned} \mathit{Manipulator} &\stackrel{\mathit{def}}{=} \mathit{Joints} \parallel \mathit{Effector} \\ \mathit{Joints} &\stackrel{\mathit{def}}{=} \mathit{Receiver} \underset{\{\mathit{moves}\}}{\bowtie} \mathit{Reporter} \\ \mathit{Receiver} &\stackrel{\mathit{def}}{=} (\mathit{position}, \top).\mathit{Receiver}' \\ \mathit{Receiver}' &\stackrel{\mathit{def}}{=} (\mathit{position}, \top).\mathit{Receiver}' + (\mathit{moves}, r_1).\mathit{Receiver} \\ \mathit{Reporter} &\stackrel{\mathit{def}}{=} (\mathit{vwrite}_J, r_2).\mathit{Reporter} \\ &\quad + (\mathit{moves}, \top).(\mathit{vwrite\_ready}_J, r_2).\mathit{Reporter} \\ \mathit{Effector} &\stackrel{\mathit{def}}{=} (\mathit{effect}, r_3).\mathit{Effector} \end{aligned}$$

For the pallet filling problem the plan begins under the assumption that the robot is in its home position (near the end of the conveyor belt). Under the control of the plan, the robot descends; activates its end effector; rises to clear the conveyor belt; moves across to the pallet, selecting an unused location on the pallet; descends; deactivates its end effector; rises to clear the pallet; and returns to the home position.

$$
\begin{aligned}
\mathit{Plan}_1 &\stackrel{\mathit{def}}{=} (\mathit{belt}_{\mathit{empty}}, q_b).\mathit{Plan}_1 \;+\; (\mathit{belt}_{\mathit{full}}, q_b).\mathit{Plan}_2\\
\mathit{Plan}_2 &\stackrel{\mathit{def}}{=} (\mathit{vwrite}_P, w).\mathit{Plan}_3\\
\mathit{Plan}_3 &\stackrel{\mathit{def}}{=} (\mathit{vread}_J, R).(\mathit{sleep}, s).\mathit{Plan}_3 \;+\; (\mathit{vread\_ready}_J, R).\mathit{Plan}_4\\
\mathit{Plan}_4 &\stackrel{\mathit{def}}{=} (\mathit{hold}, r_4).\mathit{Plan}_5\\
\mathit{Plan}_5 &\stackrel{\mathit{def}}{=} (\mathit{opened}, q_e).\mathit{Plan}_5 \;+\; (\mathit{closed}, q_e).\mathit{Plan}_6\\
\mathit{Plan}_6 &\stackrel{\mathit{def}}{=} (\mathit{vwrite}_P, w).\mathit{Plan}_7\\
\mathit{Plan}_7 &\stackrel{\mathit{def}}{=} (\mathit{deliver}, \top).\mathit{Plan}_8\\
\mathit{Plan}_8 &\stackrel{\mathit{def}}{=} (\mathit{vread}_J, R).(\mathit{sleep}, s).\mathit{Plan}_8 \;+\; (\mathit{vread\_ready}_J, R).\mathit{Plan}_9\\
\mathit{Plan}_9 &\stackrel{\mathit{def}}{=} (\mathit{closed}, q_e).\mathit{Plan}_{10}\\
\mathit{Plan}_{10} &\stackrel{\mathit{def}}{=} (\mathit{release}, r_6).\mathit{Plan}_{11}\\
\mathit{Plan}_{11} &\stackrel{\mathit{def}}{=} (\mathit{opened}, q_e).\mathit{Plan}_{12} \;+\; (\mathit{closed}, q_e).\mathit{Plan}_{11}\\
\mathit{Plan}_{12} &\stackrel{\mathit{def}}{=} (\mathit{accept}, r_5).\mathit{Plan}_{13}
\end{aligned}
$$

$$
\begin{aligned}
\mathit{Plan}_{13} &\stackrel{\mathit{def}}{=} (\mathit{pallet}_{\mathit{space}}, r_7).(\mathit{vwrite}_P, w).\mathit{Plan}_{14}\\
&\quad + (\mathit{pallet}_{\mathit{full}}, r_7).(\mathit{vwrite}_P, w).(\mathit{flush}, \top).\mathit{Plan}_{14}\\
\mathit{Plan}_{14} &\stackrel{\mathit{def}}{=} (\mathit{vread}_J, R).(\mathit{sleep}, s).\mathit{Plan}_{14} \;+\; (\mathit{vread\_ready}_J, R).\mathit{Plan}_1
\end{aligned}
$$

A workcell consists of robot, belt and pallet. These only communicate through the plan. Communication with the plan itself is done via the global memory mentioned above.

$$
\begin{aligned}
\mathit{Memory} &\stackrel{\mathit{def}}{=} (\mathit{vwrite}_J, \top).\mathit{Memory} \;+\; (\mathit{vwrite\_ready}_J, \top).\mathit{MemoryJ}\\
&\quad + (\mathit{vread}_J, \top).\mathit{Memory} \;+\; (\mathit{vwrite}_P, \top).\mathit{MemoryP}
\end{aligned}
$$

$$
\begin{aligned}
\mathit{MemoryJ} &\stackrel{\mathit{def}}{=} (\mathit{vwrite}_J, \top).\mathit{MemoryJ} + (\mathit{vread\_ready}_J, \top).\mathit{Memory} \\
&\quad + (\mathit{vwrite}_P, \top).\mathit{MemoryJP} + (\mathit{vwrite\_ready}_J, \top).\mathit{MemoryJ} \\
\mathit{MemoryP} &\stackrel{\mathit{def}}{=} (\mathit{vwrite}_J, \top).\mathit{MemoryP} + (\mathit{vwrite\_ready}_J, \top).\mathit{MemoryJP} \\
&\quad + (\mathit{vread}_J, \top).\mathit{MemoryP} + (\mathit{vwrite}_P, \top).\mathit{MemoryP} \\
&\quad + (\mathit{vread}_P, \top).\mathit{Memory} \\
\mathit{MemoryJP} &\stackrel{\mathit{def}}{=} (\mathit{vwrite}_J, \top).\mathit{MemoryJP} + (\mathit{vread\_ready}_J, \top).\mathit{MemoryP} \\
&\quad + (\mathit{vwrite}_P, \top).\mathit{MemoryJP} + (\mathit{vwrite\_ready}_J, \top).\mathit{MemoryJP} \\
&\quad + (\mathit{vread}_P, \top).\mathit{MemoryJ}
\end{aligned}
$$

$$
\begin{aligned}
\mathit{Workcell} &\stackrel{\mathit{def}}{=} (\mathit{Plan}_1 \underset{S_3}{\bowtie} (\mathit{Robot} \parallel \mathit{Belt} \parallel \mathit{Pallet}_0)) \underset{RW}{\bowtie} \mathit{Memory} \\
S_3 &\stackrel{\mathit{def}}{=} \{\, \mathit{belt}_{\mathit{empty}}, \mathit{belt}_{\mathit{full}}, \mathit{deliver}, \mathit{hold}, \mathit{release}, \mathit{pallet}_{\mathit{space}}, \mathit{pallet}_{\mathit{full}}, \mathit{accept}, \mathit{flush} \,\} \\
RW &\stackrel{\mathit{def}}{=} \{ \text{ all read and write actions } \}
\end{aligned}
$$

5 The specification with failures

When we consider the reliability aspects of the system it is clear that failures may occur in almost all of the specified components. To illustrate the combined reliability and performance modelling potential of the SPA formalism here we consider only the possibility of failure in the gripping mechanism of the end effector. This will result in the robot dropping a block if it is holding one. We assume that the average operational time (i.e. holding time) between such failures is known. The SPA model allows us to calculate the probability that any block is dropped and also the degradation to the performance of the workcell which results from this type of failure. When the robot drops a block three different situations may occur:

The block is dropped over the belt. The belt will have already been signalled that delivery has occurred and, in general, the block will not fall in a position from which it can be retrieved. Therefore we assume that this block is lost: the situation is reported to the plan which initiates failure recovery.

The block is dropped in transit between the belt and the pallet. This block is also assumed to be lost. As previously the situation is reported to the plan which initiates failure recovery.

The block is dropped onto the pallet. We assume that this is not an error from the point of view of the robot workcell as the block has been successfully transported to the pallet. However some detection and recovery may be necessary in a later stage of pallet processing. If such a failure occurs it is reported to the plan but no failure recovery procedure is initiated.

The introduction of this type of failure only affects some of the components of the workcell. Fortunately, the compositionality of the SPA language means that only the specifications of affected components need to be modified to reflect the failure and the recovery procedure.

The failure occurs in the effector. Since the effector is unaware of its current state the specification may appear to imply that a failure may occur even when the gripping mechanism is not operational.

$$\mathit{Effector} \stackrel{\mathit{def}}{=} (\mathit{effect}, r_3).\mathit{Effector} + (\mathit{drop}, D).\mathit{Effector}$$

However its controller is aware of whether the effector is currently holding a block and by including the drop action in the synchronisation set between the two components it is guaranteed that the failure may only occur when a block is gripped. The mean holding time since the previous failure is $1/D$. When a failure occurs the controller is responsible for reporting it to the plan:

$$\begin{aligned}
\mathit{EC} &\stackrel{\mathit{def}}{=} (\mathit{hold}, \top).(\mathit{effect}, \top).\mathit{EC}' + (\mathit{release}, \top).\mathit{EC} + (\mathit{opened}, \top).\mathit{EC} \\
\mathit{EC}' &\stackrel{\mathit{def}}{=} (\mathit{hold}, \top).\mathit{EC}' + (\mathit{release}, \top).(\mathit{effect}, \top).\mathit{EC} + (\mathit{closed}, \top).\mathit{EC}' \\
&\qquad\qquad + (\mathit{drop}, \top).(\mathit{error}, e).\mathit{EC}
\end{aligned}$$

The central rôle played by the plan means that it must be modified to represent the reporting of failures and the subsequent recovery procedures. Failure may occur in each state where the effector holds a block; the recovery procedure initiated will reflect the current state of the plan but will result in the robot being returned to its home position. Three new states have been introduced:

$$
\begin{aligned}
\mathit{Plan}_1 &\stackrel{\mathit{def}}{=} (\mathit{belt}_{empty}, q_b).\mathit{Plan}_1 \ + \ (\mathit{belt}_{full}, q_b).\mathit{Plan}_2 \\
\mathit{Plan}_2 &\stackrel{\mathit{def}}{=} (\mathit{vwrite}_P, w).\mathit{Plan}_3 \\
\mathit{Plan}_3 &\stackrel{\mathit{def}}{=} (\mathit{vread}_J, R).(\mathit{sleep}, s).\mathit{Plan}_3 \ + \ (\mathit{vread\_ready}_J, R).\mathit{Plan}_4 \\
\mathit{Plan}_4 &\stackrel{\mathit{def}}{=} (\mathit{hold}, r_4).\mathit{Plan}_5 \\
\mathit{Plan}_5 &\stackrel{\mathit{def}}{=} (\mathit{opened}, q_e).\mathit{Plan}_5 \ + \ (\mathit{closed}, q_e).\mathit{Plan}_6 \ + \ (\mathit{error}, \top).\mathit{PlanError}_1 \\
\mathit{Plan}_6 &\stackrel{\mathit{def}}{=} (\mathit{vwrite}_P, w).\mathit{Plan}_7 \ + \ (\mathit{error}, \top).\mathit{PlanError}_1 \\
\mathit{Plan}_7 &\stackrel{\mathit{def}}{=} (\mathit{deliver}, \top).\mathit{Plan}_8 \ + \ (\mathit{error}, \top).\mathit{PlanError}_1 \\
\mathit{Plan}_8 &\stackrel{\mathit{def}}{=} (\mathit{vread}_J, R).(\mathit{sleep}, s).\mathit{Plan}_8 \ + \ (\mathit{vread\_ready}_J, R).\mathit{Plan}_9 \\
&\quad + (\mathit{error}, \top).\mathit{PlanError}_2 \\
\mathit{Plan}_9 &\stackrel{\mathit{def}}{=} (\mathit{closed}, q_e).\mathit{Plan}_{10} \ + \ (\mathit{error}, \top).\mathit{PlanError}_2 \\
\mathit{Plan}_{10} &\stackrel{\mathit{def}}{=} (\mathit{release}, r_6).\mathit{Plan}_{11} \ + \ (\mathit{error}, \top).\mathit{PlanError}_2 \\
\mathit{Plan}_{11} &\stackrel{\mathit{def}}{=} (\mathit{opened}, q_e).\mathit{Plan}_{12} \ + \ (\mathit{closed}, q_e).\mathit{Plan}_{11} \ + \ (\mathit{error}, \top).\mathit{Plan}_{12} \\
\mathit{Plan}_{12} &\stackrel{\mathit{def}}{=} (\mathit{accept}, r_5).\mathit{Plan}_{13} \\
\mathit{Plan}_{13} &\stackrel{\mathit{def}}{=} (\mathit{pallet}_{space}, r_7).(\mathit{vwrite}_P, w).\mathit{Plan}_{14} \\
&\quad + (\mathit{pallet}_{full}, r_7).(\mathit{vwrite}_P, w).(\mathit{flush}, \top).\mathit{Plan}_{14} \\
\mathit{Plan}_{14} &\stackrel{\mathit{def}}{=} (\mathit{vread}_J, R).(\mathit{sleep}, s).\mathit{Plan}_{14} \ + \ (\mathit{vread\_ready}_J, R).\mathit{Plan}_1 \\
\mathit{PlanError}_1 &\stackrel{\mathit{def}}{=} (\mathit{deliver}, \top).\mathit{PlanError}_2 \\
\mathit{PlanError}_2 &\stackrel{\mathit{def}}{=} (\mathit{vwrite}_P, w).\mathit{PlanError}_3 \\
\mathit{PlanError}_3 &\stackrel{\mathit{def}}{=} (\mathit{vread}_J, R).(\mathit{sleep}, s).\mathit{PlanError}_3 \ + \ (\mathit{vread\_ready}_J, R).\mathit{Plan}_1
\end{aligned}
$$

Finally, as already suggested, the sets of synchronising actions, between the plan and the robot and between the controller and the manipulator, have to be enlarged by the actions *error* and *drop* respectively:

$$S_3 \stackrel{def}{=} \{ \textit{error}, \textit{belt}_{empty}, \textit{belt}_{full}, \textit{deliver}, \textit{hold}, \textit{release},$$
$$\textit{pallet}_{space}, \textit{pallet}_{full}, \textit{accept}, \textit{flush} \}$$
$$S_1 \stackrel{def}{=} \{ \textit{position}, \textit{effect}, \textit{drop} \}$$

6 Performance results

Even models of modest systems such as this one may rapidly grow to a size where functional or performance analysis will necessarily be error-prone and extremely time consuming without computer assistance. For example, the failure free robot working on pallets with space for only two blocks gives rise to a model with 316 states, and 612 states when there are four blocks per pallet. Both TIPP and PEPA are supported by prototype tools developed for these purposes. The TIPP tool is described in (Dulz and Herzog, 1995). The PEPA workbench is described in (Gilmore and Hillston, 1994).

6.1 Tool use and model solution

The models presented in this paper were solved using the TIPP tool developed at the University of Erlangen. The evaluation proceeds in three steps: The first is compilation of the SPA terms. The output of this tool is a transition system which contains all the information necessary for both functional and temporal analysis. Functional properties can be computed directly from this. The second step is a simple translation of the transition system to a Markov chain and the solution of the chain to find the steady state probabilities. In the third step these probabilities are used to compute state measures (e.g. utilisation of single components) or event measures (e.g. throughput of single actions).

6.2 Some results

The values assigned to the rate parameters of the model are shown in Table 1. Many of the actions in the model represent message passing between the plan and other components: these

Parameter	Value ($\sec^{-1}$)	Activity
q_b	5	plan queries the belt
μ	$0.01 \leq \mu \leq 2.0$	belt bringing in new blocks
d	10^6	plan tells belt that block has been lifted
c	50	calculation by joint controller
p	2×10^5	joint controller transmits a new joint position
R	2×10^5	reading a vector from memory
r_1	$0.1 \leq r_1 \leq 1.0$	movement of the joints
r_2	200	reporter writes a vector to memory
r_3	1	end effector changes state (grips/releases)
r_4	2×10^5	plan issues the instruction to grip
w	2×10^5	plan writes a vector to memory
s	1	plan delays between reading the robot position
r_5	10^6	plan positions block on the pallet
q_e	5	plan queries the state of the end effector
r_6	2×10^5	plan transmits to release instruction
r_7	5	plan queries the pallet
λ	0.1	full pallets are flushed
D	$0 \leq D \leq 0.1$	the failure rate for the gripping mechanism
e	2×10^5	rate at which the effector controller reports errors

Table 1: Table showing the values assigned to the rate parameters

typically proceed almost instantaneously, being implemented by a line. Similarly it is assumed that reading from memory is very fast, lasting approximately $500\mu S$. In contrast writing to memory, although fast in itself, is assumed to occur every $2mS$ on average. Calculating a new control vector typically takes about $20mS$.

Physical actions are much slower. For example, the joints will take between $0.1S$ and $1S$ on average to complete a move and the end effector will typically take approximately $1S$ to correctly grip or release a block. Slower still are the arrival rate of blocks and the removal of pallets which rely on interactions with other components outside the workcell.

We chose to investigate the impact of two of the slowest actions on the overall performance of the workcell. For ease of solution we limited the number of positions on the pallet to four although this can clearly be increased with only minor modifications to the model. In particular, in Figure 2, the topmost curve shows how the throughput of full pallets may be improved in the idealised system when the arrival rate of blocks is increased. This increase could be achieved by increasing the speed of the belt when it is moving, or by decreasing the distance between blocks when they are placed on the belt (outside our system).

The throughput is severely affected if the trundle rate (μ) is very slow as the availability of blocks to be moved becomes the limiting factor. The graph shows the behaviour of the system when it was within this range: for greater values of μ no improvement of performance was derived and so these values are not included in the graph. The curve when $D = 0$ was calculated on the basis of the first specification in which the idealised system was presented. The other curves, $D > 0$, were calculated on the basis of the second specification in which the faulty gripping mechanism was represented.

Secondly, we investigated the influence of the movement rate r_1 of the robot on the throughput of the workcell (Fig. 2) when the trundle rate is fixed at 2. The shape of the curves in this graph indicate that the throughput could be improved further by an increased value of r_1. However, this would mean setting the rate at which the robot assumes a new position to a higher value than is currently available within the technology. Clearly there is scope for greater productivity if faster moving robots can be developed.

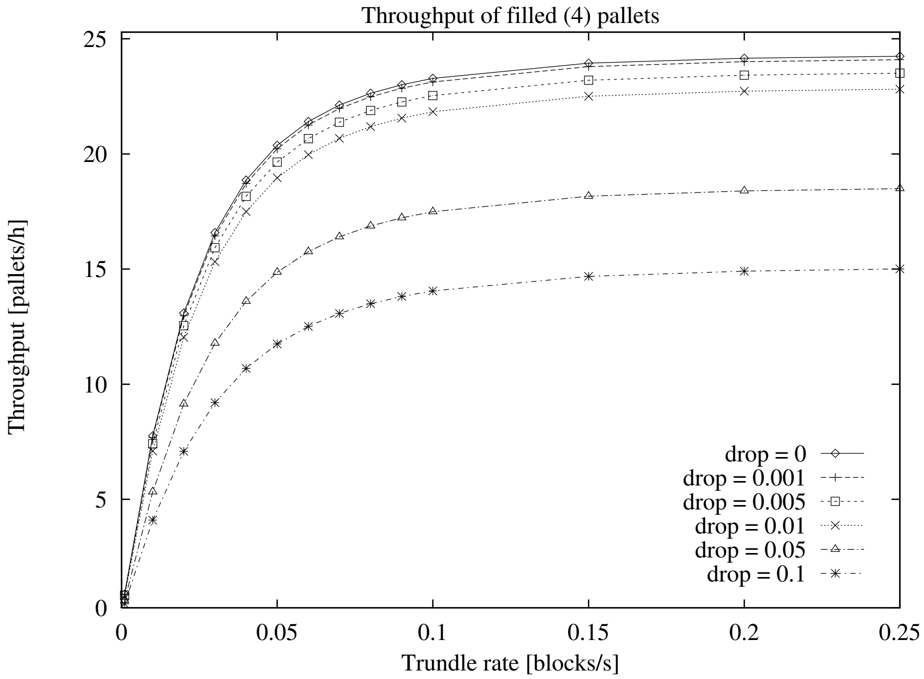


Figure 1: Throughput (full pallets/hour) of the workcell dependent on trundle rate

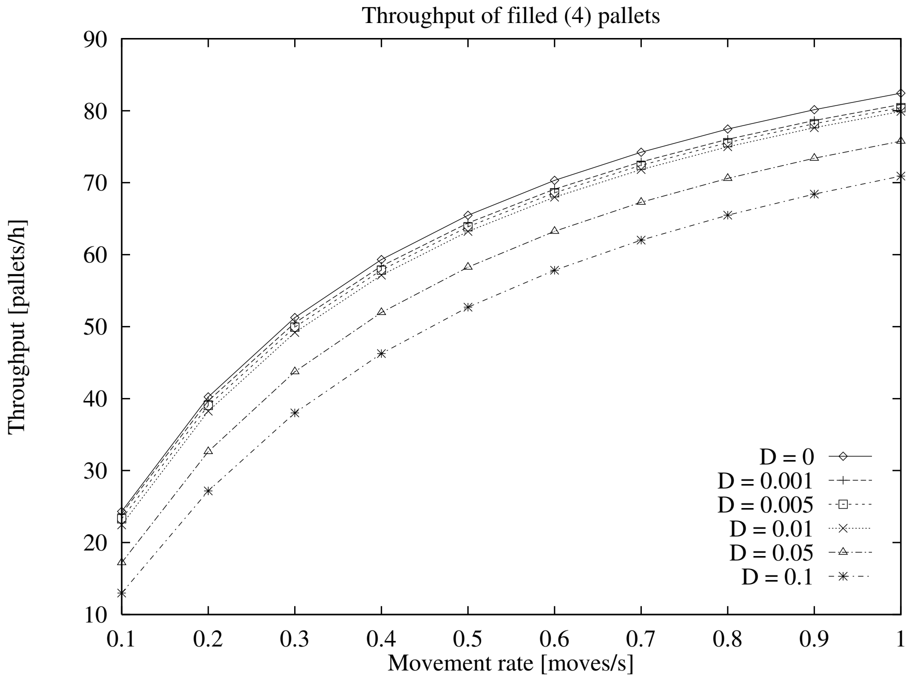


Figure 2: Throughput (full pallets/hour) of the workcell dependent on movement rate

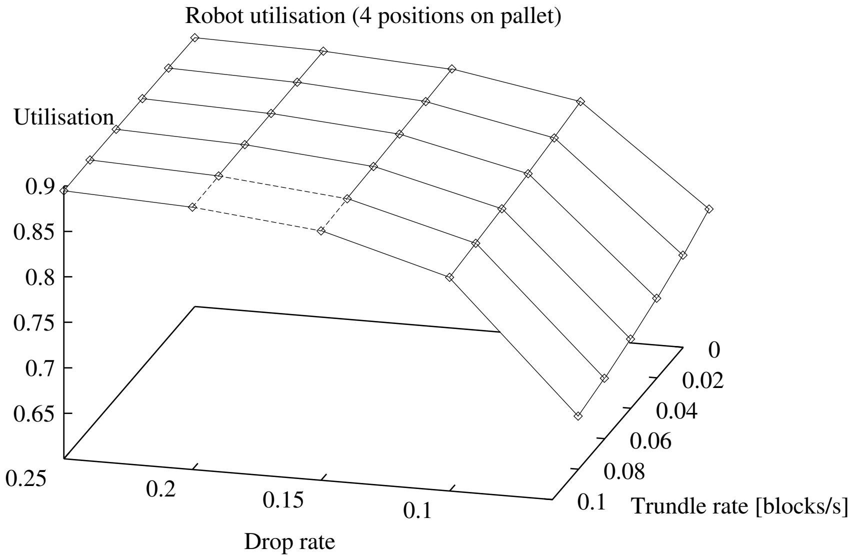


Figure 3: Utilisation of the robot dependent on the trundle and the drop rate

Finally we computed the utilisation of the robot (Fig. 3) when the movement rate of the robot is fixed at 0.1 and the trundle rate varies between 0.01 and 0.25. The robot is considered to be in use whenever it is holding and/or moving. The utilisation falls slightly as the reliability of the gripping mechanism is decreased, because the robot will spend more time waiting for the error recovery procedures to become operational.

7 Comparison with other approaches

Although performance analysis of flexible manufacturing systems (FMS) has attracted a great deal of interest in recent years (e.g. (Balbo et al., 1989; DiCesare et al., 1993)), in most of these studies any robotic component is represented as a single entity and not represented in any detail. Since the rôle of the robot is often crucial, and robots are particularly error-prone, the authors felt that there was scope for a more detailed performance analysis of these components.

In contrast, the functional behaviour of robot arms has been modelled in detail using various approaches. For instance, in (Kilpatrick et al., 1990) a specification of a pallet filling robot is developed using the model oriented specification language VDM (Jones, 1990). This approach differs markedly from the SPA approach. The VDM model pays more attention to the trajectory taken by the robot when moving from one position to another, while the internal structure of the robot is abstracted away entirely. Similarly in (Hanna et al., 1993) a detailed Petri net model is developed of a workcell in which a robot arm plays a central rôle. A hierarchical structure is introduced into the Petri net in order to represent the workcell in a modular fashion: separate nets are used to model each aspect of the workcell behaviour. In particular, details of the movement and behaviour of the robot are modelled separately, distinct control and safety nets specifying the interactions between them.

Preliminary work with the SPA specification included a similar level of detail but it was found to be unnecessary for the performance indices being investigated, needlessly increasing the state space of the underlying Markov process. Clearly the objectives of the Petri net and VDM models were different. The aim of the Petri net model was to analyse the safety properties of the system. Similarly the VDM specification was intended to establish the correct behaviour of the robot and guide implementation. These modelling techniques and the SPA approach presented in this paper are complementary, each incorporating information specific to its objectives.

From a modelling perspective, the view of the workcell provided by the SPA or Petri net specifications can be regarded as more realistic than the VDM specification. For example, the conveyor belt is modelled as a sequence of blocks in VDM as opposed to as an agent, in the SPA model, which allows a block to move along the conveyor and signals whether a block is available or not. A similar realism is achieved in the approach taken in (Holton, 1991), where a robot is specified in LOTOS (ISO, 1987), a system description language based on a classical process algebra without timing information. Like the SPA model, the LOTOS model makes a division between the robot controller and manipulator hardware. However, in this model also a little more detail is given to the various components of the system. For example, a robot is considered to consist of the composition of a number of joints, each joint represented as a sensor and an actuator. In some cases this increased detail is to the detriment of the readability of the specification.

Both the VDM and LOTOS models allow data values to be represented explicitly, a feature which is not present in the SPA model. In the case of LOTOS such values may be passed between agents allowing choices between behaviours to be made on the basis of those values.

All the techniques described so far take a modular approach to modelling a workcell, dividing the system into separate physical components (VDM, LOTOS, SPA) or into different aspects of behaviour (Petri net). In the case of the LOTOS model, the robot specification is part of a larger system for supporting robot programming, and the interface to the robot is restricted to a set of predefined interactions. The SPA model presented in this paper could similarly be embedded into a larger model.

Finally we mention the approach of Jin et al. (Jin et al., 1989) which is between the performance and functional approaches outline above. In (Jin et al., 1989) a higher level model is developed with the objective of investigating the impact of errors in a robot, especially when it is placed in a wider environment. Again a Petri net is used to represent the behaviour of the system. A reachability analysis of the net results in a graph which is transformed into a Markov renewal process by associating a firing distribution with each arc. These processes are solved using standard numerical techniques to obtain steady state probabilities of the system being in any given state. For example, a system consisting of two interacting robots is considered to find the operating ratio of the system. This approach has clear similarities to the SPA approach presented in this paper especially when the firing distributions are restricted to the exponential distribution.

8 Conclusions and further work

We have introduced a new approach to performance analysis and demonstrated its application to a novel class of systems. Stochastic process algebra provides a structured system description which is amenable to both qualitative and quantitative analysis. Thus it represents a link between two important classes of system description techniques: process algebras used by designers to determine functional properties of systems and stochastic models used to determine performance characteristics. This incorporation of performance modelling features into an established system description formalism represents a significant advance towards the time consideration of quantitative characteristics of systems during design.

This integrated approach is particularly suited to the chosen application, robot control, as it allows the impact of varying functional behaviour on performance characteristics to be investigated. It is important to be able to model systems which do not behave perfectly since many real systems include such erroneous patterns of behaviour. In some cases failures will be fatal and classical process algebra techniques will identify these as deadlocks. In less drastic situations being able to quantify the effect of the failure increases the designers knowledge of the system.

A stochastic process algebra model can also be used to stimulate questions about the intended behaviour of a system long before the actual implementation is underway. For example, if the robot was at the end of the conveyor belt and a block was available but a full pallet had not been flushed should the robot pick up the block? The advantage to picking up the block (and moving it to some neutral position between the belt and the pallet) is that the conveyor belt will be able to be restarted to bring another block. The disadvantage is that the robot might then potentially be left holding the block for a long time while waiting for the pallet to be flushed. The designers of a system are stimulated to pose such questions about system behaviour when undertaking a formal model of the system. This encourages them to consider flaws or difficulties which might not otherwise be considered at that time.

The compositional nature of stochastic process algebras encourage the production of a library of specifications of commonly used components. For example, in the areas of flexible manufacturing systems these would be devices such as conveyor belts and robots. Once such a library has been established the task of constructing a model of a system is eased considerably since only some of the components, and possibly a plan, will need to be constructed from scratch.

The compositional structure also means that it is extremely straightforward to embed the specification of the workcell within a larger system of a manufacturing plant. For example, in a more realistic model the rate of the trundle and flush actions would be determined externally to the workcell. This would involve making these actions passive within the belt and the pallet respectively and defining a larger system of the form:

$$\mathit{Supply} \underset{\{\mathit{trundle}\}}{\bowtie} (\mathit{Workcell}/S) \underset{\{\mathit{flush}\}}{\bowtie} \mathit{Removal}$$

where the set S includes all the actions of the workcell except *trundle* and *flush*, indicating that the environment cannot influence the behaviour of the workcell in any other way.

Some of the actions within the model of the workcell occur much more rapidly than others, for example, the communication between the *Plan* and all other components. This results in a stiff Markov process in which transition rates differ by several orders of magnitude. In larger models such bad conditioning would cause problems for the numerical solution routines. One area for future work would be an investigation of how to overcome this problem. Some form of temporal abstraction could be introduced into the formalism so that very fast actions which have no impact on system performance could be represented as immediate actions. Following the approach of generalised stochastic Petri nets (GSPN) (Ajmone Marsan et al., 1984), states which enable such immediate actions could be eliminated before solution techniques are applied. The inclusion of this form of abstraction into TIPP is currently under investigation.

Acknowledgements

This work was supported by grant number 632 under the DAAD (Deutscher Akademischer Austauschdienst) and British Council ARC programme, and by EPSRC postdoctoral fellowship number B/93/RF/1729.

References

AJMONE MARSAN, M., CONTE, G., and BALBO, G. (1984). A Class of Generalised Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems. *ACM Transactions on Computer Systems* 2(2):93–122.

BALBO, G., CHIOLA, G., and FRANCESCHINIS, G. (1989). Stochastic Petri Net Simulation for the Evaluation of Flexible Manufacturing Systems. In Tucci, S., Mathis, A., Hahn, W., and Zobel, R., editors, *Simulation Applied to Manufacturing Energy and Environmental Studies and Electronics and Computer Engineering*.

DICESARE, F., HARHALAKIS, G., PROTH, J., SILVA, M., and VERNADAT, F. (1993). *Practice of Petri Nets in Manufacturing*. Chapman & Hall.

DULZ, W. and HERZOG, U. (1995). Entwurf, Implementierung und Bewertung von Kommunikationsdiensten – Arbeitsbericht 1993–95, SFB 182, Teilprojekt B3. Technical report Universität Erlangen–Nürnberg, IMMD VII.

GILMORE, S. and HILLSTON, J. (1994). The PEPA Workbench: A Tool to Support a Process Algebra-based Approach to Performance Modelling. In Haring, G. and Kotsis, G., editors, *Proceedings of the Seventh International Conference on Modelling Technqiues and Tools for Computer Performance Evaluation* volume 794 of *LNCS* pages 353–368. Springer-Verlag.

GÖTZ, N., HERZOG, U., and RETTELBACH, M. (1993). Multiprocessor and Distributed System Design: The Integration of Functional Specification and Performance Analysis using Stochastic Process Algebras. In *Performance'93*.

HANNA, M., BUCK, A., and SMITH, R. (1993). Modelling Safety Requirements of an FMS using Petri Nets. In *Proc. of SPIE Int. Symposium on Optical Tools for Manufacturing and Advanced Automation*.

HERMANNS, H. and RETTELBACH, M. (1994). Semantics, Equivalence and Axiomatization. In *Proc. of the 2nd Workshop on Process Algebra and Performance Modelling*. IMMD7, Universität Erlangen-Nürnberg.

HILLSTON, J. (1994). *A Compositional Approach to Performance Modelling*. PhD thesis Department of Computer Science, University of Edinburgh. CST-107-94.

HOARE, C. (1985). *Communicating Sequential Processes*. Prentice-Hall.

HOLTON, D.R.W. (1991). *A Rigorous Approach to Robot Programming*. PhD thesis The Queen's University of Belfast.

ISO (1987). *Information Processing Systems—Open Systems Interconnection—LOTOS, A Formal Description Technique based on the Temporal Ordering of Observational Behaviour*. ISO.

JIN, Q., SUGASAWA, Y., and SEYA, K. (1989). Probabilistic Behavior and Reliability Analysis for a Multi-Robot System by Applying Petri Net and Markov Renewal Process Theory. *Microelectronics and Reliability* 29(6):993–1001.

JONES, C. (1990). *Systematic Software Development using VDM*. Prentice Hall.

KILPATRICK, P., MCPARLAND, P., and GILMORE, S. (1990). The Formal Development of Robot Software. In *IEE Computing and Control Division Colloquium on The Application of CASE Tools* London. The Institution of Electrical Engineers. Digest No. 1990/058.

MILNER, R. (1989). *Communication and Concurrency*. Prentice-Hall.