Implementing a Stochastic Process Algebra within the Möbius Modeling Framework *

G. Clark and W.H. Sanders

Dept. of Electrical and Computer Engineering and Coordinated Science Laboratory
University of Illinois at Urbana-Champaign
1308 W. Main St., Urbana, IL, U.S.A.

Abstract. Many formalisms and solution methods exist for performance and dependability modeling. However, different formalisms have different advantages and strengths, and no one formalism is universally used. The Möbius tool was built to provide multi-formalism multi-solution modeling, and allows the modeler to develop models in any supported formalism. A formalism can be implemented in Möbius if a mapping can be provided to the Möbius Abstract Functional Interface, which includes a notion of state and a notion of how state changes over time. We describe a way to map PEPA, a stochastic process algebra, to the abstract functional interface. This gives Möbius users the opportunity to make use of stochastic process algebra models in their performance and dependability models.

1 Introduction

Many performance and dependability modeling formalisms and model solution methods have been developed. The most suitable formalism for a particular model depends on many factors, including the experience of the modeler, the results required, and the resources available to solve the model. In addition, large modeling projects may be split among several teams, each with different modeling backgrounds. For these reasons, it is desirable to provide techniques and tools for constructing heterogeneous models that may be composed of submodels of different formalisms, each seamlessly interacting with each other and with model solvers. The Möbius project aims to provide such techniques and tools.

The theory of Möbius is designed to support heterogeneous modeling by the use of an *abstract functional interface* (AFI [1, 2]), a specification that any candidate modeling formalism must implement. By doing so, the formalism ensures that its models may share state and synchronize with other models, support the definition of performance variables, and be solved using one of several methods. The first formalism supported by Möbius was stochastic activity networks (SANs [3]). This paper gives details of the incorporation of a new modeling formalism,

* This material is based upon work supported in part by the National Science Foundation under Grant No. 9975019 and by the Motorola Center for High-Availability System Validation at the University of Illinois (under the umbrella of the Motorola Communications Center).

PEPA [4], which is a stochastic process algebra (SPA). Our work means that PEPA models may now be specified, composed with other submodels, and solved within the Möbius tool.

Section 2 provides some background on the Möbius framework and tool, and a review of the required SPA concepts. Section 3 discusses equivalence-sharing in Möbius, and some basic notions of state for a PEPA process. Section 4 presents an extension to PEPA that makes use of process parameters, and gives details of how this extension can be employed in providing a useful and intuitive mapping to the Möbius AFI. Section 6 presents an example of modeling with PEPA using the Möbius tool, and finally Section 7 discusses ways in which the Möbius framework could be extended in the future, and concludes the paper.

2 Background

In this section, we first describe the Möbius framework and its implementation as the Möbius tool. Following that, we provide a reminder of the relevant features of our stochastic process algebra, PEPA.

2.1 The Möbius Framework

The Möbius framework [5, 6] provides an abstract set of requirements for building a particular modeling formalism. It is based upon a theory that is motivated by many existing modeling formalisms, and seeks to capture the essential components of models built using these formalisms. Any model that is built according to the Möbius framework must present a specified interface to other models and solvers within the framework. The implementation of this interface is known as the *AFI*. The AFI includes:

- a set S of *state variables*, and
- a set A of *actions*.

As described in [6], a state variable consists of a type, a value, and a distribution over initial values. Its value is typically used to represent the state of a component or subcomponent, such as the number of customers at a service center in a queuing network. The Möbius framework specifies a rich and structured set of variable types T. In particular, the integers and subsets of T are all state variable types. In theory this set is of infinite size; in practice we assume that it is arbitrarily large, but finite. Of course, this is a reasonable assumption when it comes to implementing the Möbius tool. The type of a state variable is given by a function $type : S \rightarrow T$; the value of a state variable is given by a function $val : S \rightarrow V$ where S is the set of all state variables, and $V \in T$. Two state variables s_1 and s_2 are *compatible* if and only if $type(s_1) = type(s_2)$, and are equal if and only if they are compatible, and $val(s_1) = val(s_2)$.

An action consists of a set of action functions (and in general, some state of its own, used when building the stochastic process of the model). An action is responsible for changing the values of state variables. Just as state variables provide an abstraction of model state, actions are intended to be an abstraction of the state-changing methods of existing modeling formalisms. For a given action $a \in A$, the action function $Enabled_a : V \to bool$ determines whether or not a is "active" in the current state and capable of changing the state at some point in the future if uninterrupted. $Delay_a : V \to (\mathbb{R} \to [0, 1])$ associates a probability distribution function (PDF) with an action in the current state; this PDF describes the time from the point of enabling until completion. Given a state, $Complete_a : V \to V$ specifies the state that will result from the completion of the action. Because every constructed Möbius model is intended for performance and dependability evaluation, non-probabilistic non-determinism is not directly supported in the AFI. There are more subtle action functions that capture the effect that action interruptions have on delay distributions; for more details, see [6].

2.2 The Möbius Tool [2]

Models are built hierarchically with Möbius. The modeler begins by specifying a set of one or more *atomic* models. For example, one of these atomic models may be a SAN, just as would have been supplied to the Möbius tool's predecessor, *UltraSAN* [7]. A *composed model* is built from a set of atomic (or indeed composed) models. The Möbius tool currently features two composed model formalisms, both based on the notion of *equivalence sharing*. In both formalisms, submodels are linked together such that compatible state variables may be identified. The AFI ensures that the formalism need not know the implementation details of its submodels' state. We refer to a set of composed submodels as *partner models*. A benefit of using these composed model formalisms is that the theory of *reduced base model construction* [8] is employed to construct a lumped stochastic process, which often results in significantly smaller state spaces. An atomic or composed model is then combined with a *performability variable* defined on the model to generate a *solvable model*. The performability variable is a description of the particular measure that the modeler wishes to calculate. A solvable model may be parameterized on a set of *global variables* to produce a *study* which is composed of a set of *experiments*. Current work is focusing on inferring statistics from a constrained set of experiments using a design of experiments approach [9]. Finally, the modeler must choose a technique for solving the collection of experiments, and the particular solver to use. The Möbius tool provides a number of analytical solvers that use a variety of linear algebra techniques for solving for steady-state and transient measures. Alternatively it is possible to employ an efficient discrete event simulator, which provides confidence intervals for the required performance measures.

2.3 Stochastic Process Algebras and PEPA

In recent years, interest has grown in the use of process-algebra-based methodologies for performance modeling and evaluation. PEPA [4] is a well-known stochastic process algebra. Process algebra notations are based on formal languages, and the PEPA language provides a small set of combinators, presented below:

$$S ::= (\alpha, r).S \mid S + S \mid A_S$$
$$P ::= P \underset{L}{\bowtie} P \mid P/L \mid A \mid S$$

Any process described by S is termed a *sequential component*. A process P consists of a *model configuration* of sequential or model components. A PEPA model consists of a set of definitions and a *system equation*, a distinguished PEPA term that can be interpreted as the starting state. The language is deliberately parsimonious as this helps keep the theory manageable and reduce the proof burden.

For a detailed description of PEPA's combinators, see [4]. *Prefix* is the most fundamental combinator; a process $(\alpha, r).P$ may perform activity (α, r), which has *action type* α and is exponentially distributed with mean $1/r$, and then evolve into process P. We use $\mathtt{a}$ to denote an arbitrary activity. Process $P + Q$ expresses a competitive *choice* between P and Q in which the enabled activities of P and Q compete, and a race condition distinguishes the component into which the process evolves. The *cooperation* $P \underset{L}{\bowtie} Q$ is a process that expresses the parallel and synchronizing execution of both P and Q. Both components proceed independently on activities whose types are not in the set L. However, those activities with types captured by L require the concurrent participation of both subprocesses, and this results in an activity with a rate that reflects the rate of the slower participant. Finally, P/L is the process that *hides* activities with types in L. These become *silent* activities with type τ. This combinator is used to express abstraction.

Processes may be recursively defined as the least solution to a set of equations where each is of the form $A \stackrel{\text{def}}{=} P$. A classical process algebra combinator missing from PEPA is the nullary combinator 0, representing the *deadlocked* process. To date, the focus with PEPA modeling has been on steady-state measures, for which 0 has little application. 0 can still be represented in PEPA (for example, if $P \stackrel{\text{def}}{=} (\alpha, r).P$, $Q \stackrel{\text{def}}{=} (\beta, s).Q$, then $P \underset{\{\alpha,\beta\}}{\bowtie} Q$ is deadlocked). We make use of a deadlocked process later in the paper. The operational semantics of PEPA infer the *transitions* of a compound process from the transitions of its subcomponents. If $P \xrightarrow{(\alpha,r)} P'$ then P' is called an $((\alpha, r)$-$)$*derivative* of P. If $P \xrightarrow{(\alpha,r)}$, then there exists some P' such that P' is an (α, r)-derivative of P. The *derivative set* of a PEPA process P is denoted by $ds(P)$, and is the smallest set of components that is closed under the transitive closure of the transition relation. This captures all "reachable states" from P. Both prefix and choice are termed *dynamic* combinators, meaning that the combinators do not persist (in general) over transitions. In contrast, cooperation and hiding do persist over transitions, and are termed *static*. From the transition system, a Markov chain can be produced by essentially discarding activity labels on arcs, providing a performance model. In order to perform a steady-state analysis with a finite state space, it is required that the underlying Markov chain be irreducible and ergodic. A necessary (but not sufficient) condition for this is that the PEPA process be *cyclic*. A *cyclic* PEPA process is a process with a structure such that no static combinator is within the scope of a dynamic combinator. Since static combinators are never "destroyed" by activity transitions, this syntactic condition ensures that the structure of the process does not grow unboundedly over time. For more details on PEPA, including its well-developed equational theory, see [4].

3 Equivalence Sharing

In the Möbius framework, models M_1 to M_n exhibit an *equivalence sharing* relationship if there exists a state variable s_i from each model such that for $1 \leq i \leq j \leq n$, s_i and s_j are compatible, and at all times, $val(s_i) = val(s_j)$. Möbius uses equivalence sharing relationships in the construction of Replicate/Join and Graph composed models. The Möbius tool first uses state identification and modification methods provided in the implementation of the AFI to link together the appropriate portions of submodel state. Any component of Möbius that requests information about the state of the composed model must also make use of the AFI, ensuring that the correct data is returned. In this way, Möbius presents a uniform view of model state, but allows for internal efficiencies in storing composed model state. Equivalence sharing allows for what can be viewed as a form of "two-way communication" between models. By altering the value of the shared portion of model state, one submodel can influence the behavior of its partners, and similarly, can have its behavior influenced by its partners.

3.1 Representations of State

PEPA's operational semantics provide a translation from a collection of algebraic expressions into a graph model, which leads to a continuous-time Markov chain. Each state of the Markov chain is associated with a particular process expression (or an equivalence class of process expressions). Therefore, the PEPA process can be viewed as evolving over time according to transition rules, with the current state represented by a particular term of the process algebra. As mentioned in Section 2.3, we restrict ourselves to considering cyclic PEPA processes only, in order to prevent the structure of the PEPA terms from growing without bound over time. Since we consider only cyclic PEPA processes, we can view a PEPA process as having a tree structure such that no "static" nodes lie below a "dynamic" node. Since the static structure is invariant over activity transitions, we consider everything at the level of, and below, a dynamic combinator to be an evolving subcomponent. A simple tree representation is presented in Figure 1, in which the dotted rectangles highlight the subcomponents.

$$Sys \stackrel{\mathrm{def}}{=} (((\alpha, r).P' \underset{\{\alpha\}}{\bowtie} (\alpha, s).Q')/\{\alpha\}) \underset{L}{\bowtie} (R + S)$$

Taken together, the individual states of these subcomponents, along with the invariant static structure of the process, are enough to characterize the state of the model as a whole.

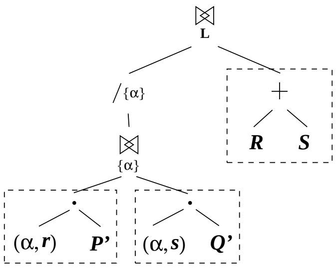


Fig. 1. Highlighting submodel state for *Sys*

A simple vector notation for *Sys* is $\langle(\alpha, r).P', (\alpha, s).Q', R + S\rangle_{Sys}$. The states of each sequential component may be enumerated, leading to a simple first mapping to a set of state variables for the AFI. Concretely, for $\langle R_1, \ldots, R_n\rangle_P$ and $1 \leq i \leq n$, state variable s_i is such that $type(s_i) = \mathbb{N}$ and $val(s_i) = enum(R_i)$. However, after considering the use of these state variables in an equivalence sharing relationship, we reject this as a workable mapping for two reasons:

– There is no obviously meaningful way to interpret the natural number enumeration of a sequential component, and there seems to be no compelling way in which a partner model could use this data.

– If a partner model changed the value of one or more state variables, this would effect a "jump" in the structure of the PEPA model. This is too uncontrolled.

Instead, our technique will rely on extending PEPA with *process parameters* and providing these as state variables. We next present PEPA$_k$, our extension to PEPA.

4 Extending PEPA with Process Parameters

In this section, we present PEPA$_k$, an extension to PEPA that makes use of process parameters. Extending process algebras with parameters is certainly not a new idea; for example, see [10–13]. What is novel is our use of this extension to implement equivalence sharing between models. We present the syntax of PEPA$_k$ next.

Definition 1 (Syntax of PEPA$_k$) *Let A range over a set C of* process constants*, A_S over a set $C_S \subseteq C$ of* sequential process constants*, and x over a set X of* process parameters*. Let e represent the syntax of arithmetic expressions over X, and b represent the syntax of predicates over X. Then the syntax of PEPA$_k$ is given by:*

$$S ::= (\alpha, r).S \;\mid\; (\alpha!\mathsf{e}, r).S \;\mid\; (\alpha?\mathtt{x}, r).S \;\mid\; S + S \;\mid\; \mathsf{if}\ b\ \mathsf{then}\ S \;\mid\; A_S \;\mid\; A_S[\underline{e}]$$
$$P ::= P \underset{L}{\bowtie} P \;\mid\; P/L \;\mid\; A \;\mid\; A[\underline{e}] \;\mid\; S$$

PEPA$_k$ provides the following additions to PEPA:

Formal Parameters: process variables now have an arity and may be instantiated with parameters. Furthermore, processes may be specified through the definition of equations of the form $P[x_1, \dots, x_n] \stackrel{\text{def}}{=} Q$.

Guards: process expressions may now be *guarded*, meaning that the behaviour specified by the process expression is only present if the guard, which may feature references to parameters, evaluates to true given the current parameter values.

Value-Passing: values may now be communicated between sequential components via activities.

Additional features could have been added, but they would not have greatly strengthened the usefulness of the language. In the next section, we present a mapping from PEPA$_k$ to PEPA, and illustrate that quite deliberately, the underlying algebra has not been changed.

4.1 A Semantics for PEPA$_k$

In this section, we provide a PEPA semantics for PEPA$_k$. We do this so that the behavior of PEPA$_k$ models within Möbius can be understood formally, and it is important to note that this translation is not carried out within the tool itself. Before we give our semantics, we give a preliminary definition that is used to construct PEPA cooperation and hiding sets.

Definition 2 (Refining data-passing activities) *Let L be a set of action types, and T be the set of Möbius state variable types (from Section 2.1). The function* $\mathsf{refine}(L)$ *is defined as:*

$$L \cup \{\alpha_{\mathtt{i}} : \alpha \in L, i \in T\}$$

As is conventional, we map all α-activities used in value passing to T-indexed α-activities. Due to the construction of T, all such refined cooperation and hiding sets remain countable. Now we can define a PEPA semantics for PEPA$_k$.

Definition 3 (PEPA Semantics for PEPA$_k$) *Let* $\mathsf{eval}(e, E)$ *represent the simple evaluation of expression e in an environment E. Then $[\![.]\!]_E$ is a function that given an environment E mapping variables to values, maps a PEPA$_k$ process or* *defining equation to a set of PEPA processes or defining equations as follows:*

$$
\begin{aligned}
\llbracket P[\underline{x}] \stackrel{\text{def}}{=} Q \rrbracket_E &= \{ P_{\underline{i}} \stackrel{\text{def}}{=} \llbracket Q \rrbracket_{E'} : i_1, \ldots , i_n \in T, E' = E[i_1/x_1, \ldots , i_n/x_n] \} \\
\llbracket P \underset{L}{\bowtie} Q \rrbracket_E &= \llbracket P \rrbracket_E \underset{\mathsf{refine}(L)}{\bowtie} \llbracket Q \rrbracket_E \\
\llbracket P/L \rrbracket_E &= \llbracket P \rrbracket_E / \mathsf{refine}(L) \\
\llbracket P + Q \rrbracket_E &= \begin{cases} \llbracket P \rrbracket_E & \text{\textit{if} } \llbracket Q \rrbracket_E = 0 \\ \llbracket Q \rrbracket_E & \text{\textit{if} } \llbracket P \rrbracket_E = 0 \\ \llbracket P \rrbracket_E + \llbracket Q \rrbracket_E & \textit{otherwise} \end{cases} \\
\llbracket \mathsf{if}\ b\ \mathsf{then}\ P \rrbracket_E &= \begin{cases} \llbracket P \rrbracket_E & \text{\textit{if} } \mathsf{eval}(b, E) = \mathsf{true} \\ 0 & \textit{otherwise} \end{cases} \\
\llbracket (\alpha!\mathtt{e}, r).Q \rrbracket_E &= (\alpha_{\mathsf{eval}(\mathtt{e},E)}, \mathsf{eval}(r, E)).\llbracket Q \rrbracket_E \\
\llbracket (\alpha?\mathtt{x}, r).Q \rrbracket_E &= \sum_{\{\alpha_i \,:\, i \in T\}} (\alpha_{\mathtt{i}}, \mathsf{eval}(r, E)).\llbracket Q \rrbracket_{E[i/x]} \\
\llbracket (\alpha, r).Q \rrbracket_E &= (\alpha, \mathsf{eval}(r, E)).\llbracket Q \rrbracket_E \\
\llbracket A[\underline{e}] \rrbracket_E &= A_{\mathsf{eval}(e_1,E),\ldots,\mathsf{eval}(e_n,E)}
\end{aligned}
$$

A guarded PEPA$_k$ process P is mapped to P if the guard is true in the current environment; otherwise it is mapped to the deadlocked process. Deadlocked processes are then removed if they are found to appear in choice contexts. Of course it is still possible to write a deadlocked PEPA$_k$ process, just as it is possible to write a deadlocked PEPA process. Parameterized process definitions are mapped to a set of indexed definitions for each possible combination of parameter values, and process constants provide a way for the PEPA$_k$ process to change the values of variables itself. A PEPA$_k$ process of the form $(\alpha?\mathtt{x}, r).P$ is mapped to a sum (choice) over all $\alpha_{\mathtt{i}}$-guarded copies of P, for $i \in T$ (recall that the set T is arbitrarily large but finite). A PEPA$_k$ process of the form $(\alpha!\mathtt{e}, s).P$ is mapped to a single PEPA process, specifically $(\alpha_{\mathtt{j}}, s).P$, where j is the result of evaluating e in the environment E. This means that a PEPA$_k$ process of the form $(\alpha?\mathtt{x}, r).P \underset{\{\alpha\}}{\bowtie} (\alpha!\mathtt{e}, s).P$ would be capable of one transition, via a single activity of type $\alpha_{\mathtt{j}}$. This has the effect of setting the value of x in the left subcomponent to be equal to j. This scheme also means that $(\alpha!\mathtt{f}, r).P \underset{\{\alpha\}}{\bowtie} (\alpha!\mathtt{e}, s).P$ is deadlocked unless $\mathsf{eval}(e, E) = \mathsf{eval}(f, E)$. This is reasonable, with the interpretation that if both subcomponents are trying to force the setting of a variable in another subcomponent, then they must agree on the value for the variable. This scheme has been used in previous parameterized process algebras; for example, see [11, 12].

Our chosen semantics is not suitable for understanding a PEPA$_k$ process such as $P \stackrel{\text{def}}{=} (\alpha?\mathtt{x}, r).P'$ in isolation. As described in [13], the choice over all $\alpha_{\mathtt{i}}$-guarded copies of P' means that the sojourn time in state P is not $1/r$ as would be expected, but rather decreases arbitrarily. However, this does not cause problems for our implementation. We insist that all PEPA$_k$ processes specified evolve such that if an input activity of type α is enabled, it must be within a cooperation context that enables an output activity of type α. If, at any point during its evolution, the PEPA$_k$ process represented by the model's system equation enables an unmatched input activity, then the model is in error. This should be considered in the same light as the specification of a model with a deadlock; it is perfectly possible to write an incorrect model specification. In our implementation, Möbius catches this error during model execution and halts. It can be shown that for any PEPA$_k$ model satisfying the condition given above, our semantics lead to a PEPA model with an equivalent performance model. Alternative semantic models do exist for value-passing algebras; for example, the STGLA model [13] avoids branching problems by maintaining process variables and expressions in a symbolic form.

Below we give a PEPA$_k$ model of a simple *M/M/s/n* queue and illustrate the translation to PEPA.

Example 1.

$$\begin{aligned} Queue[m,s,n] \stackrel{\mathrm{def}}{=} & \ \mathsf{if}\ (m < n)\ \mathsf{then}\ (\mathtt{in}, \lambda).Queue[m+1,s,n] \\ & + \mathsf{if}\ (m > 0)\ \mathsf{then}\ (\mathtt{out}, \mu * \min(s,m)).Queue[m-1,s,n] \end{aligned}$$

translates to a set of definitions over values of s and n, including the following:

$$\begin{aligned} Queue_{0,s,n} &\stackrel{\mathrm{def}}{=} (\mathtt{in}, \lambda).Queue_{1,s,n} \\ Queue_{i,s,n} &\stackrel{\mathrm{def}}{=} (\mathtt{in}, \lambda).Queue_{i+1,s,n} + (\mathtt{out}, \mu * i).Queue_{i-1,s,n} \ \text{ for } 0 < i < s \\ Queue_{i,s,n} &\stackrel{\mathrm{def}}{=} (\mathtt{in}, \lambda).Queue_{i+1,s,n} + (\mathtt{out}, \mu * s).Queue_{i-1,s,n} \ \text{ for } s \le i < n \\ Queue_{n,s,n} &\stackrel{\mathrm{def}}{=} (\mathtt{out}, \mu * m).Queue_{n-1,s,n} \end{aligned}$$

We have presented PEPA$_k$, an extension to PEPA, and shown that while expressiveness has been improved and additional modeling flexibility has been provided, the underlying process algebra has not changed. In the next section, we present our application of PEPA$_k$ to the identification of Möbius state variables.

5 Mapping a PEPA$_k$ Process to the AFI

We present a practical technique for identifying state variables in an SPA model, and provide a formal mapping to the Möbius AFI. State variables will be given by PEPA$_k$ process parameters. This means that the modeler will provide explicit guidance on exactly what state may be shared, and means that the modeler will create the PEPA model in such a way that the effect of a change in shared state will be well understood. In order to list the state variables of a PEPA$_k$ process, we first provide the definition of an auxiliary function that calculates the state variables of a sequential PEPA$_k$ process.

Definition 4 (State variables of a sequential PEPA$_k$ process) *The state variables of a sequential PEPA$_k$ process P are given by* $\mathsf{svars}(P, \emptyset) \in X$*, where* $\mathsf{svars}(., W)$ *is defined as follows:*

$$\begin{aligned}
\mathsf{svars}(\mathsf{a}.S, W) &= \mathsf{svars}(S, W)\\
\mathsf{svars}(S + T, W) &= \mathsf{svars}(S, W) \cup \mathsf{svars}(T, W)\\
\mathsf{svars}(\mathsf{if}\ b\ \mathsf{then}\ S, W) &= \mathsf{svars}(S, W)\\
\mathsf{svars}(A[\underline{e}], W) &= \begin{cases} (\{\underline{x}\} \cup \mathsf{svars}(S, W \cup (A, n)) & \textit{if } A[\underline{x}] \stackrel{\text{def}}{=} S \textit{ and } (A, n) \notin W \\ \emptyset & \textit{otherwise} \end{cases}
\end{aligned}$$

This definition deliberately fails to distinguish between a parameter x of process S and another parameter with the same name of process T, if $T \in ds(S)$. We consider these to represent the same state variable. This mechanism allows the PEPA$_k$ modeler to change the value of a state variable simply; for example, if S and T both have a parameter named x then $T[x] \stackrel{\text{def}}{=} (\alpha, r).S[f(x)]$ will perform an activity (α, r), and immediately afterwards change x's current value from a to $f(a)$. Now we give the state variables for a PEPA$_k$ process, P.

Definition 5 (State variables of a PEPA$_k$ process) *Let* $P = \langle S_1, \ldots, S_n \rangle_P$ *be a PEPA$_k$ process in vector form. The state variables of P are given by:*

$$\bigcup_{1 \leq i \leq n} \{x_i : x \in \mathsf{svars}(S_i, \emptyset)\}$$

Each state variable is instrumented with an index according to its position in the vector, to ensure that duplicate names do not clash. This completes the definition of the AFI state variables. However, the state variables alone do not characterize the state of the PEPA$_k$ process; it is necessary to take into account the current PEPA$_k$ term too. Just as described in Section 3.1, the process state can be captured using a vector notation. This state is maintained for interacting with model solvers, but is not exported for use in state-sharing or the definition of performance variables. The complete state of a PEPA$_k$ process is discussed further in Section 5.1. In order to complete the mapping to the AFI, we must generate from the PEPA$_k$ process a set of Möbius actions. There are several ways in which this can be done. Given a PEPA$_k$ process $P = \langle S_1, \ldots, S_n \rangle_P$, there are two possibilities for the AFI actions:

- for every $P' \in ds(P)$, the enabled activities of P' (distinguishing duplicates). We reject this idea since it would require that we generate the entire state space of the underlying process in order to provide the mapping.
- for $1 \leq i \leq n$, for every $S_i' \in ds(S_i)$, the enabled activities of S' (distinguishing duplicates). This is reasonable, and can be implemented efficiently. One drawback is in the consideration of impulse rewards; if three sequential components cooperate over an activity, a reward could not be specified for the

firing of an activity, as the result of a cooperation between a particular two of the three components.

Instead, we chose a method that can be efficiently computed and does not sacrifice expressibility. Definition 6 specifies the type of Möbius actions.

Definition 6 (Set of Actions) *The set MA is the least set inductively defined as follows:*

- $(\mathsf{a}, P) \in MA$ *for all activities* a *and* $PEPA_k$ *processes* P
- *if* $B \in MA$ *then* $(\mathsf{a}, P, B) \in MA$
- *if* $B_1, B_2 \in MA$ *then* $(\mathsf{a}, P, B_1 \uplus B_2) \in MA$

Now let P be a PEPA$_k$ process. The AFI actions of P are given by $\mathsf{actions}(P, \emptyset) \in MA$, where $\mathsf{actions}(., W)$ is defined as follows:

Definition 7 (Actions for a PEPA$_k$ process) *Let P be a PEPA$_k$ process. The AFI actions of P are given by* $\mathsf{actions}(P, \emptyset)$*, where* $\mathsf{actions}(., W)$ *is defined as follows:*

$$
\begin{aligned}
\mathsf{actions}(S + T, W) &= \mathsf{actions}(S, W) \cup \mathsf{actions}(T, W)\\
\mathsf{actions}((\alpha?\mathtt{x}, r).S, W) &= \{((\alpha, r), S)\} \cup \mathsf{actions}(S, W)\\
\mathsf{actions}((\alpha!\mathsf{e}, r).S, W) &= \{((\alpha, r), S)\} \cup \mathsf{actions}(S, W)\\
\mathsf{actions}((\alpha, r).S, W) &= \{((\alpha, r), S)\} \cup \mathsf{actions}(S, W)\\
\mathsf{actions}(\mathsf{if}\ b\ \mathsf{then}\ S, W) &= \mathsf{actions}(S, W)\\
\mathsf{actions}(A[\underline{e}], W) &= \begin{cases} \mathsf{actions}(P, W \cup (A, n)) & \text{if } A[\underline{x}] \stackrel{\text{def}}{=} P \text{ and } (A, n) \notin W\\ \emptyset & \text{otherwise}\end{cases}\\
&\text{let } \Phi_P = \mathsf{actions}(P, W) \text{ and } \Phi_Q = \mathsf{actions}(Q, W);\ \text{then}\\
\mathsf{actions}(P \bowtie_L Q, W) &= \{((\alpha, r), P' \bowtie_L Q, B) : B = ((\alpha, r), P', A) \in \Phi_P, \alpha \notin L\} \cup\\
&\quad \{((\alpha, r), P \bowtie_L Q', B) : B = ((\alpha, r), Q', A) \in \Phi_Q, \alpha \notin L\} \cup\\
&\quad \{((\alpha, R), P' \bowtie_L Q', B_1 \uplus B_2) : (\alpha, r).P' \bowtie_L (\alpha, s).Q' \xrightarrow{(\alpha, R)},\\
&\quad B_1 = ((\alpha, r), P', A_1) \in \Phi_P,\\
&\quad B_2 = ((\alpha, s), Q', A_2) \in \Phi_Q\}\\
\mathsf{actions}(P/L, W) &= \{((\tau, r), P', B) : B = ((\alpha, r), P', A) \in \Phi_P, \alpha \in L\} \cup\\
&\quad \{((\alpha, r), P', B) : B = ((\alpha, r), P, A) \in \Phi_P, \alpha \notin L\}
\end{aligned}
$$

For each derivative of each sequential component, this function computes every enabled activity. The AFI actions of a cooperation $P \bowtie_L Q$ are

- actions associated with individual behavior of each subcomponent (types that do not match those in L).

- for each pair of AFI actions with types $\alpha \in L$, a new action consisting of a cooperation between the two.

Thus a Möbius action a is a structure consisting of a PEPA activity, the derivative that results from its completion, and then a set of the Möbius actions of the model's subcomponents that have combined to form a. The advantage of this technique is that every activity that could be enabled by any derivative of P is mapped to a Möbius AFI action. This means that the Möbius modeler can distinguish and assign an impulse reward to a composite activity resulting from the evolution of a chosen subset of model subcomponents. The disadvantage of this technique is that since we directly compute "products" of activities, there may be a proliferation of AFI actions that correspond to PEPA activities that may never be enabled.

We have presented a mapping from PEPA_k to the Möbius AFI. This means that PEPA_k models can be composed with other Möbius atomic models using equivalence sharing. The ability of another model to unilaterally change some shared state has some consequences for the parameterized process, which we discuss next.

5.1 Implications of Modeling with PEPA_k

We have described a mapping from PEPA_k to the AFI, and have shown that PEPA_k is no more powerful than PEPA itself. By providing the modeler with some of the convenience of a programming language, we

- facilitate the construction of concise PEPA_k specifications that have a natural PEPA semantics.
- cause the modeler to structure his definitions in such a way that a change in the value of a state variable (PEPA_k process parameter) due to a partner model will cause a meaningful and understandable change in the state of the PEPA_k model.

The last point is an important one, and justifies our selection of this method for implementation. The addition of such "cues" into the PEPA model makes equivalence sharing meaningful and useful.

In Section 5, we stated that the PEPA formalism maps process parameters to state variables for use in state sharing and the definition of performance variables, but that the state of the PEPA_k process is also maintained and communicated to model solvers via the AFI. This means that we can construct a PEPA_k model P that Möbius interprets as having a larger state space than $\llbracket P \rrbracket_\emptyset$. Consider the following PEPA_k definition:

$$\begin{aligned} S[x] \stackrel{\text{def}}{=} & \ \mathsf{if}\ x \neq 1\ \mathsf{then}\ (\alpha_1, r).(\beta, s).S[1] \\ & + \mathsf{if}\ x \neq 2\ \mathsf{then}\ (\alpha_2, r).(\beta, s).S[2] \\ & + \mathsf{if}\ x \neq 3\ \mathsf{then}\ (\alpha_3, r).(\beta, s).S[3] \end{aligned}$$

Translating this to PEPA leads to:

$$S_1 \stackrel{\text{def}}{=} (\alpha_2, r).(\beta, s).S_2 + (\alpha_3, r).(\beta, s).S_3$$
$$S_2 \stackrel{\text{def}}{=} (\alpha_1, r).(\beta, s).S_1 + (\alpha_3, r).(\beta, s).S_3$$
$$S_3 \stackrel{\text{def}}{=} (\alpha_1, r).(\beta, s).S_1 + (\alpha_2, r).(\beta, s).S_2$$

The PEPA process has 6 states. However, using the AFI, the Möbius state space generator will detect 9 unique states for the PEPA$_k$ process. The reason for this is that the process itself can be in the state $(\beta, s).S[1]$ while the state variable x may either have the value 2 or the value 3 (and similarly for the other derivatives of $S[x]$). These states are equal by every reasonable process algebra equivalence, since $(\beta, s).S[1]$ makes no further use of the value of x. However, what is crucially different is that the AFI now allows a partner model to use the value of x while the PEPA$_k$ model is in this state. If the modeler wishes to generate the smallest reasonable state space, the model could alternatively be specified as below:

$$\begin{aligned} S[x] &\stackrel{\text{def}}{=} \mathsf{if}\ x \neq 1\ \mathsf{then}\ (\alpha_1, r).S'[1] \\ &\quad + \mathsf{if}\ x \neq 2\ \mathsf{then}\ (\alpha_2, r).S'[2] \\ &\quad + \mathsf{if}\ x \neq 3\ \mathsf{then}\ (\alpha_3, r).S'[3] \\ S'[x] &\stackrel{\text{def}}{=} (\beta, s).S[x] \end{aligned}$$

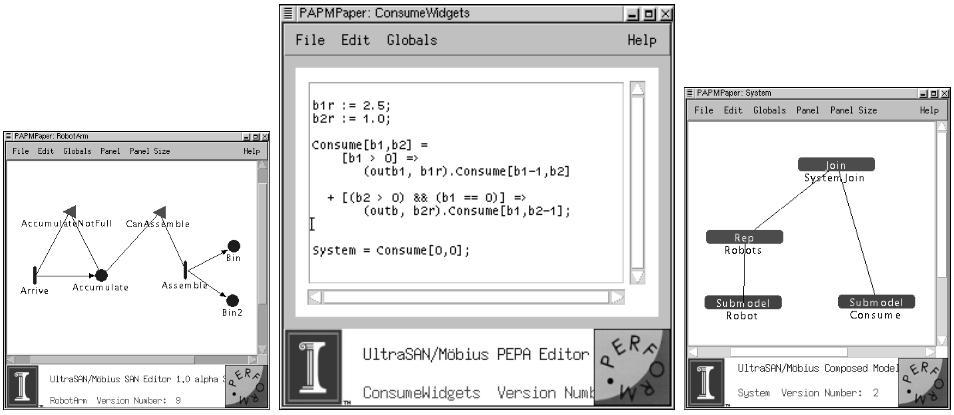
This works because the value of the state variable is changed one activity sooner. However, this process will not behave identically to its original partner process. Furthermore, although in isolation, the PEPA$_k$ model has a state space consistent with the translated PEPA model. In an equivalence sharing relationship, the partner model still has the opportunity to change the value of x at any point, leading in this case to a slightly larger state space. If the modeler aims to employ his PEPA$_k$ model in an equivalence sharing relationship, this is a novel issue of which he must be aware.

One interesting aspect of this work is the extent to which equivalence relations may be employed for aggregation when equivalence sharing is being used. We can certainly ensure that $P \underset{L}{\bowtie} P'$ is treated as equivalent to $P' \underset{L}{\bowtie} P$ and check for this as detailed in [14]. However, if P is defined with a parameter, e.g., $P[x] \stackrel{\text{def}}{=} (\alpha, r).P'[x+1]$, then $P[a] \underset{L}{\bowtie} P'[b]$ will export two state variables, x_1 and x_2, with values a and b respectively, via the AFI. A partner model may be relying on the individual values of these variables, and thus it would be incorrect to equate this process to $P'[b] \underset{L}{\bowtie} P[a]$. For processes with parameters, the order in which they appear in the static structure of a term must be preserved.

6 Example

We present an example to illustrate the implementation and use of PEPA$_k$ in Möbius. Our model is of a simple factory system that consists of three robot arms. Two robots are responsible for removing items from a conveyor belt, packaging several items together into two units and then depositing one assembled unit into each of the two bins. The third robot removes assembled units from the shared bins and places them on another conveyor belt for later processing. It does this by always attempting to remove an item from the first bin if it can, and only if it cannot, removing and passing along an item from the second bin.

The first two robot arms behave identically, and are modeled by one SAN as shown in Figure 2a. The leftmost activity fires until place Accumulate is full; meanwhile, the robot arm assembles units, incrementing the values of places Bin1 and Bin2 when activity Assemble fires. The PEPA$_k$ model of the third robot arm is shown in Figure 2b. From the specification of process *Consume*$[b1, b2]$, it can be seen that activity (`outb1`, *b1r*) is enabled if the value of parameter $b1$ is positive, and (`outb2`, *b2r*) is enabled if $b2$ is positive, and $b1$ equals zero. From here, the



(a) SAN model of first robot

(b) PEPA model of second robot

(c) Model of factory system

Fig. 2. Example models

overall model of the system can be easily built, and is shown in Figure 2c. We use the Replicate/Join formalism; two copies of the first robot are produced by applying a replicate node to the SAN model. The Replicate creates an integer number of copies of a model, and does not distinguish individual copies. We insist that both copies of the first robot share places Bin1 and Bin2. Next we use a Join node to apply equivalence sharing to the Replicated model of the first robot, and the PEPA$_k$ model of the second robot. The Möbius tool allows us to specify that shared place Bin1 should be further shared (identified) with the parameter $b1$ of the PEPA model, and similarly that Bin2 should be identified with $b2$. In this way, we create an accurate model of the system as a whole. Due to our mapping to the AFI, it is now possible to use the Möbius tool's analytical solvers or discrete-event simulator to investigate the behavior of this model over time.

7 Future Work and Conclusion

One area of future work is in the design of an *action-sharing* composition modeling formalism for Möbius. Here, Möbius can benefit from the experience and results of the SPA community. PEPA, and SPAs in general, are packaged with a compositional theory based around synchronization via actions. In contrast, the Möbius framework, and its implementation as the Möbius tool, currently supports the synchronization of concurrent models via shared state. One reasonable and useful extension to the theory would be to allow action-synchronization operators, such as PEPA's cooperation, to be applicable to any submodels that satisfy the AFI. Many choices of operator may be useful; for example, the operator may insist that some particular subset of "cooperating" activities complete before the submodels change state. The literature features several ways in which the rate of cooperation can be chosen [15]. Furthermore, there is work on building compositional Petri net-based modeling languages [16, 17], and also on exploiting process algebra results in the development of a compositional Petri net theory [18]. By combining such action-based operators with Möbius's Replicate/Join composition formalism, it will be possible to create a new and general model composition formalism. Furthermore, both formalisms provide support for state space aggregation based upon identifying "replicated" subcomponents, and this aggregation should be preserved in any joint formalism. With the potential to communicate data between submodels using actions, and with current work on exploiting group theory to detect symmetries [19], this new formalism has the potential to be fruitful in both theory and practice.

We have presented a method of allowing PEPA models to be composed with other models via equivalence sharing, and thus for incorporating PEPA into the Möbius tool. As a result, the Möbius tool may now be used to specify and solve PEPA models, and to combine them with existing modeling formalisms, such as SANs and an extended Markov chain formalism called Buckets and Balls [20]. We believe this illustrates the flexibility and generality of both the Möbius framework and the AFI. Furthermore, we expect that a similar mapping can be developed for SPAs other than PEPA.

ACKNOWLEDGMENTS

We would like to thank Jane Hillston for her valuable suggestions and for making this collaboration possible. We also thank Jay Doyle and Dan Deavours for their patient explanations and assistance, and Holger Hermanns for several useful discussions. Thanks also to the other members of the Möbius group: Tod Courtney, David Daly, Salem Derisavi, and Patrick Webster.

References

1. J. M. Doyle, "Abstract model specification using the Möbius modeling tool," M.S. thesis, University of Illinois at Urbana-Champaign, 2000.

2. G. Clark, T. Courtney, D. Daly, D. Deavours, S. Derisavi, J. M. Doyle, W. H. Sanders, and P. Webster, "The Möbius modeling tool," in *Proc. of PNPM'01: 10th International Workshop on Petri Nets and Performance Models, Aachen, Germany (to appear)*, September 2001.
3. J. F. Meyer, A. Movaghar, and W. H. Sanders, "Stochastic activity networks: Structure, behavior and applications," *Proc. International Workshop on Timed Petri Nets*, pp. 106–115, 1985.
4. J. Hillston, *A Compositional Approach to Performance Modelling*, Cambridge University Press, 1996.
5. W. H. Sanders, "Integrated frameworks for multi-level and multi-formalism modeling," in *Proc. PNPM'99: 8th International Workshop on Petri Nets and Performance Models, Zaragoza, Spain*, September 1999, pp. 2–9.
6. D. Deavours and W. H. Sanders, "Möbius: Framework and atomic models," in *Proc. PNPM'01: 10th International Workshop on Petri Nets and Performance Models, Aachen, Germany (to appear)*, September 2001.
7. W. H. Sanders, W. D. Obal II, M. A. Qureshi, and F. K. Widjanarko, "The *ultrasan* modeling environment," *Performance Evaluation*, vol. 24, no. 1, pp. 89–115, October-November 1995.
8. W. H. Sanders and J. F. Meyer, "Reduced base model construction methods for stochastic activity networks," *IEEE Journal on Selected Areas in Communications*, vol. 9, no. 1, pp. 25–36, Jan. 1991.
9. D. Montgomery, *Design and Analysis of Experiments*, John Wiley & Sons, Inc., 5th edition, 2001.
10. R. Milner, *Communication and Concurrency*, International Series in Computer Science. Prentice Hall, 2nd edition, 1989.
11. T. Bolognesi and E. Brinksma, "Introduction to the ISO specification language LOTOS," *Computer Networks and ISDN Systems*, vol. 14, pp. 25–59, 1987.
12. H. Hermanns and M. Rettelbach, "Toward a superset of basic LOTOS for performance prediction," *Proc. of 4th Workshop on Process Algebras for Performance Modelling (PAPM)*, pp. 77–94, 1996.
13. M. Bernardo, *Theory and Application of Extended Markovian Process Algebra*, Ph.D. thesis, University of Bologna, Italy, 1999.
14. S. Gilmore, J. Hillston, and M. Ribaudo, "An efficient algorithm for aggregating PEPA models," *IEEE Transactions on Software Engineering*, 2001.
15. J. Hillston, "The nature of synchronisation," *Proc. of 2nd Workshop on Process Algebras for Performance Modelling (PAPM)*, pp. 51–70, 1994.
16. S. Donatelli, "Superposed generalized stochastic Petri nets: Definition and efficient solution," in *Application and Theory of Petri Nets 1994, Lecture Notes in Computer Science 815 (Proc. 15th International Conference on Application and Theory of Petri Nets, Zaragoza, Spain)*, R. Valette, Ed., pp. 258–277. Springer-Verlag, June 1994.
17. I. Rojas, *Compositional Construction of SWN Models*, Ph.D. thesis, The University of Edinburgh, 1997.
18. E. Best, R. Devillers, and M. Koutny, *Petri Net Algebra*, Monographs in Theoretical Computer Science. An EATCS Series. Springer-Verlag, 2000.
19. W. D. Obal II, *Measure-Adaptive State-Space Construction Methods*, Ph.D. thesis, The University of Arizona, 1998.
20. W. J. Stewart, *Introduction to the Numerical Solution of Markov Chains*, Princeton University Press, 1994.