PEPA: A STOCHASTIC PROCESS ALGEBRA FORMALISM

Graham Clark
gcla@crhc.uiuc.edu

Performability Engineering Research Group
Coordinated Science Laboratory and Electrical and Computer Engineering Department
University of Illinois at Urbana-Champaign

http://www.crhc.uiuc.edu/PERFORM

Motorola Center for High-Availability System Validation Review Meeting
October 4, 2000

Project funded in part by the Motorola Center for High-Availability System Validation, under the umbrella of the Motorola Communications Center, and National Science Foundation Next Generation Software Program

Introduction

This talk will introduce you to PEPA, a stochastic process algebra formalism being incorporated into UltraSAN/Möbius

- The motivation for this work
- A description of process algebra, and how it is used for modeling
- Modeling performance – PEPA
- What goes on *underneath the hood*
- A simple example – a multiprocessor system
- Integrating PEPA into the Möbius framework, and the benefits for modelers

Motivation

- UltraSAN/Möbius will be a multi-formalism modeling tool, and PEPA is a new and different modeling paradigm
- *An analogy :*
  - SANs are *graphical* – models are graphs
  - PEPA is *textual* – models are programs
- The use of Process algebras for performance modeling is a growing field
- New formalisms will allow colleagues to collaborate in constructing models *without forcing a single modeling paradigm on all parties*

What are Process Algebras?

- View as a programming language for describing performance models
- Central aims:
  - Compositionality – a methodology for systematically building the complex from the simple
  - Concurrency – *built-in for free*, as a consequence
- Prominent representatives:
  - For research: CCS [Milner], CSP [Hoare]
  - For applications: LOTOS (ISO Std. 8807) e.g. the study of communications protocols

```
process Spec :=
enter.exit.Spec
endproc


process Peterson[p1_enter,
p1_exit, p2_enter, p2_exit] :=

hide
 flag1,flag2,…

in
(Proc[…] <flag1,…> Proc[…])

endproc

…
```

What is PEPA?

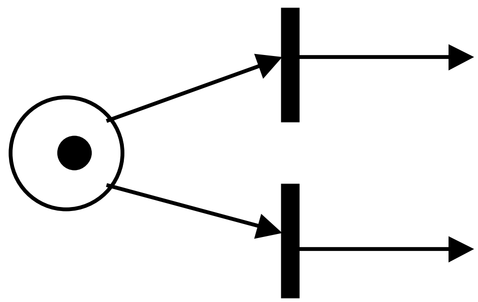
- PEPA stands for "*Performance Evaluation Process Algebra*"
- Primitive process algebra *actions* become timed PEPA *activities*:

```
enter.exit.Spec  <---> (enter,r).(exit,s).Spec
```

- r and s are the parameters of exponentially distributed random variables which determine the time it takes for each activity to complete
- What are the primitives for building PEPA models?

PEPA Combinators

- Prefix: given an activity (a,r), and a process P, (a,r).P is a process which performs the activity (a,r) and then becomes P
- Choice: P + Q is a process which expresses competition between P and Q. It is analogous to the following SAN fragment:



- Cooperation: given processes P and Q, and a set of activity names L, the process P <L> Q expresses the parallel composition of P and Q with synchronization on L activities; c.f. increasing the number of tokens in a SAN place
- Hiding: given a process P, and a set of activity names L, the process P/L hides those names in L from further interaction

The Underlying Model

- The model evolves from state to state by performing activities:

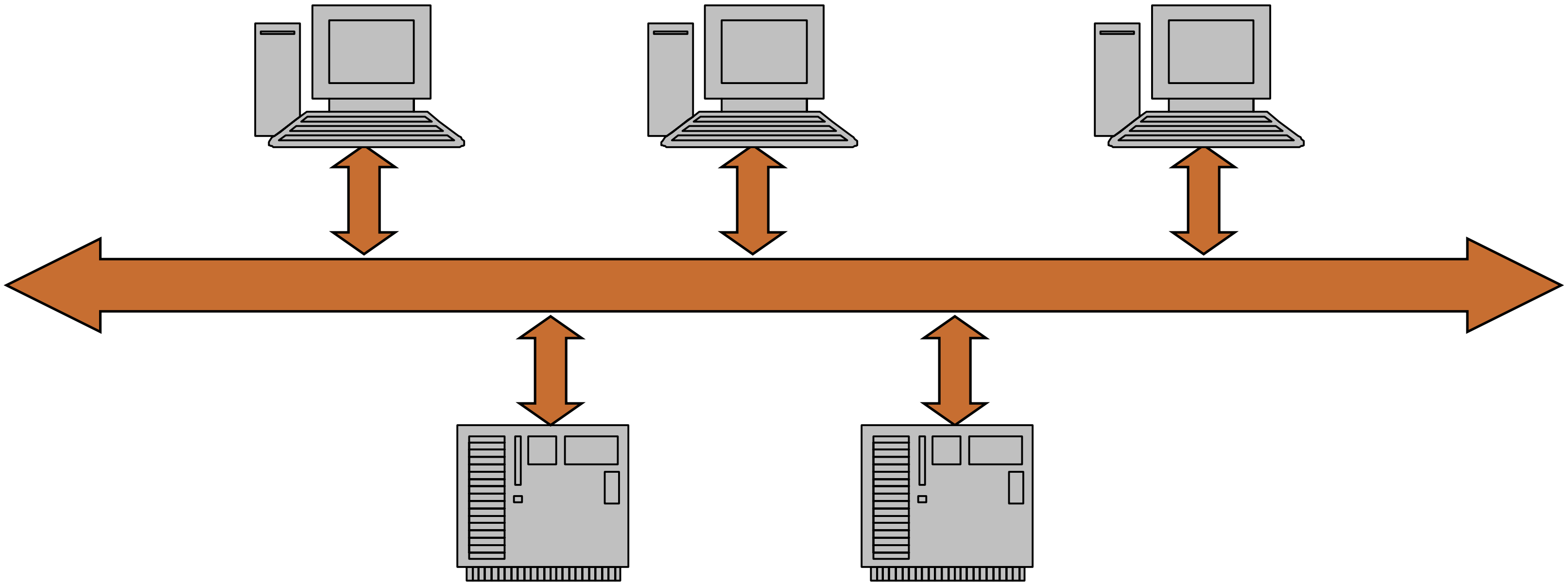
$$\texttt{(enter,r).(exit,s).Spec} \xrightarrow{\texttt{(enter,r)}} \texttt{(exit,s).Spec} \xrightarrow{\texttt{(exit,s)}} \texttt{Spec}$$

- Rules are used to calculate the behavior of processes from their subcomponents. Assume **`item`** is in **`L`**; then

$$\frac{\texttt{Producer} \xrightarrow{\texttt{(item, r)}} \texttt{Producer'} \qquad \texttt{Consumer} \xrightarrow{\texttt{(item, s)}} \texttt{Consumer'}}{\texttt{Producer <L> Consumer} \xrightarrow{\texttt{(item, R)}} \texttt{Producer' <L> Consumer'}}$$

- Leads to direct simulation, or an analytical solution

An Example Model



- An abstraction of a multiprocessor system with shared memory
- Three processors (each called `Proc`)
- Two shared memory modules (called `Mem1` and `Mem2`)
- A global bus (`Bus`) through which all communication with the shared memory takes place

An Example Specification

- **Definitions**:

```
Mem1   := (getM1,-).(relM1,-).Mem1
Mem2   := (getM2,-).(relM2,-).Mem2
Bus    := (getM1,g1).(relM1,r).Bus + (getM2,g2).(relM2,r).Bus
Proc   := (getM1,-).(use,u1).(relM1,-).(update,p1).(think,t).Proc
        + (getM2,-).(use,u2).(relM2,-).(update,p2).(think,t).Proc
```

- *"System Equation"*:

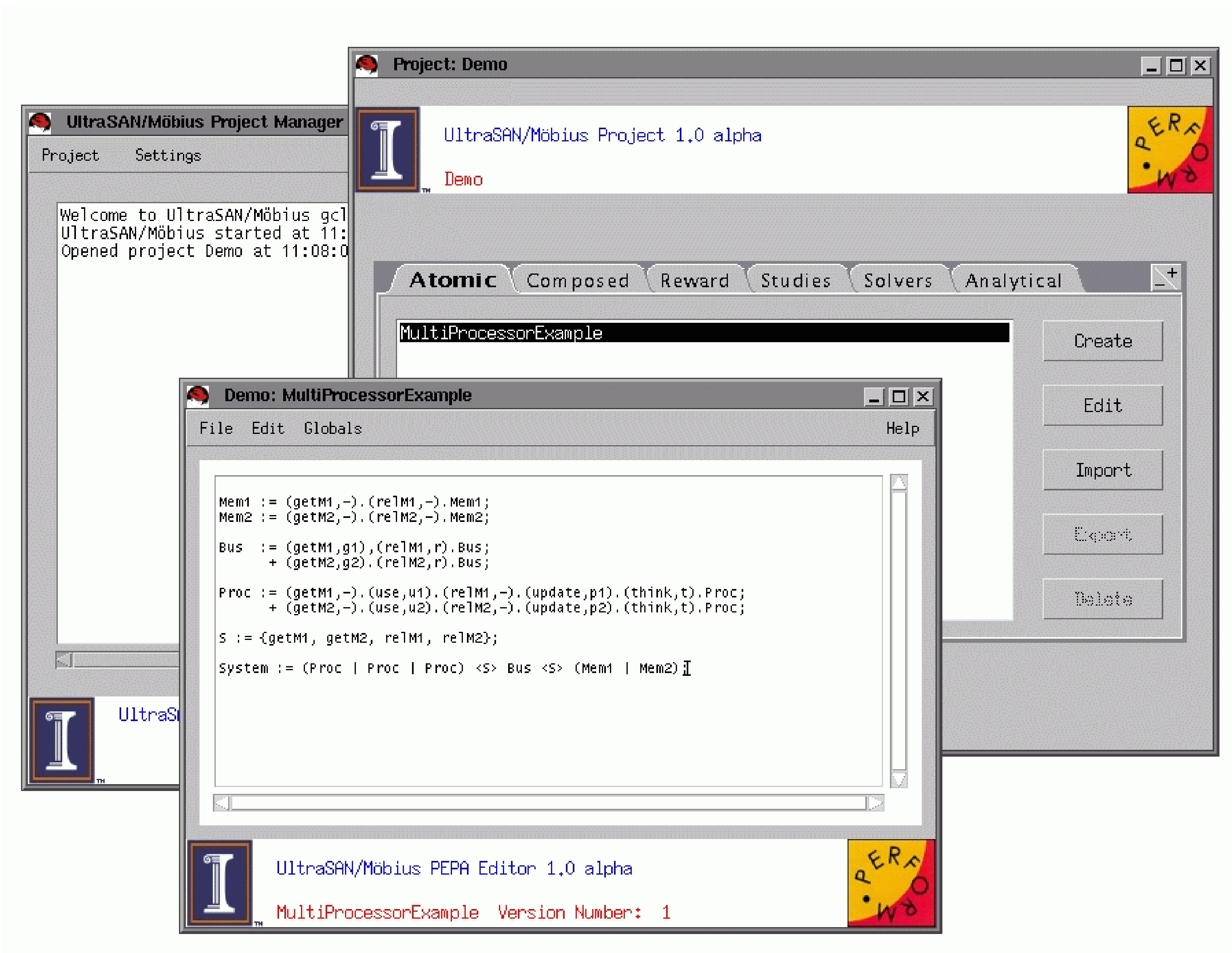
```
System :=     (Proc | Proc | Proc) <S> Bus <S> (Mem1 | Mem2)
                     (where S = {getM1,getM2,relM1,relM2})
```

Integration into UltraSAN/Möbius

- We have a mapping from a PEPA model to the Möbius Abstract Functional Interface
  - AFI actions are derived from PEPA activities
  - AFI state variables are given by the number of concurrent instances of subcomponents in a particular state

  e.g. `(Proc | Proc | Proc)` would generate a state variable with a value of **3**
- PEPA models will then be able to share state and interact with other UltraSAN/Möbius formalisms
- Changing the value of a shared state variable will alter the rate at which the PEPA model proceeds
- The modeler must provide a textual description of the PEPA model (and a little more information)

In Development...



Conclusions

- Stochastic Process Algebra, and in particular PEPA, has been introduced as an alternative paradigm for performance modeling
- With the incorporation of PEPA into UltraSAN/Möbius, users will be able to model in a style similar to more traditional computer programming
- These models can each share state, and interact with other UltraSAN/Möbius models in a meaningful way
- UltraSAN/Möbius is becoming equipped with a wider range of formalisms (SAN, Buckets and Balls, PEPA,…) – more choice for modelers!