Proceedings of the 2003 Winter Simulation Conference
S. Chick, P. J. Sánchez, D. Ferrin, and D. J. Morrice, eds.

DESIGN ENVIRONMENTS FOR COMPLEX SYSTEMS

Corrado Priami

Dipartimento di Informatica e Telecomunicazioni
University of Trento
Via Sommarive, 14
38050 Povo (TN), ITALY

ABSTRACT

The paper describes an approach for modeling complex systems by hiding as much formal details as possible from the user, still allowing verification and simulation of the model. The interface is based on UML to make the environment available to the largest audience. To carry out analysis, verification and simulation we automatically extract process algebras specifications from UML models. The results of the analysis is then reflected back in the UML model by annotating diagrams. The formal model includes stochastic information to handle quantitative parameters. We present here the stochastic π-calculus and we discuss the implementation of its probabilistic support that allows simulation of processes. We exploit the benefits of our approach in two applicative domains: global computing and systems biology.

1 INTRODUCTION

Complex systems are more and more pervasive of our current research. They arise directly from computer science development (e.g., the Internet and its potential usage as a big resource to be allocated on demand to computing tasks - global computing, `http://www.cordis.lu/ist/fetgc.htm`) and also from other disciplines that needs computer help (e.g., the emerging attitude of life scientists to look at the dynamic evolution of biological systems rather than at their structure only–systems biology, Kitano 2002).

Many different skills and disciplines have taken a system approach to model the objects of their research and all of them need computer aid to handle the huge amount of data and relations that describe temporal evolution of systems. Although different applicative domains introduce peculiar features in the modeling, verification and simulation phases of the design process, many principles can be shared altogether. Furthermore, most of the designers coming from applicative domains different from computer science do not have deep knowledge of the mathematics upon which verification and simulation is built. According to the above, we claim that effective sharing of principles and tools can be achieved in design environments only if most of the technical details on the verification and simulation side are hidden from the user.

We implement hiding of details from the user by splitting the architecture of our environment into two parts: an interface environment and a kernel environment. The interface allows the user to specify the model and to select the kind of verification or simulation s/he wants to carry out. The kernel contains the machinery to perform formal analysis and simulation.

Since one of the goal of the approach is to make the environment usable by the largest audience, we choose a standard as the modeling formalism: UML (Booch et al. 1997). Most of the IT companies already use UML to model their software products and hence dissemination of the approach should be easier. Furthermore, the very same formalism is proposed as a modeling standard in systems biology (Roux-Rouquie and Le Moigne 2002). This opens another huge applicative domain to our environment.

The kernel environment is concerned with the formalization of critical aspects of the application in hand for verification or simulation purposes. We rely here on process algebras for mobile systems (Milner et al. 1992, Degano and Priami 2001). They are simple calculi made up of very few operators (sequentialization, parallel composition, scope delimiters, alternatives) that mainly describe the interaction between processes and their interconnection topology during dynamic execution. These calculi are equipped with many tools for static as well as dynamic analysis of behavioral properties that can be reused in our setting. Furthermore, recently stochastic extesions of these calculi have been proposed thus allowing the management of quantitative information. Notably, a probabilistic runtime support implemented for a version of the stochastic π-calculus (Priami et al. 2001) allows simulation of peculiar behavior.

An issue of the overall approach is the connection between the two environments. This is in fact the step needed to hide formal details from the user. The simplest way we devised to implement the interaction is through manipulation of the XML Metadata Interchange format (XMI) (OMG 2000) that provides a standard textual format in which UML models can be exchanged between tools. XML is the Extensible Markup Language (W3C 2000), which, along with its many supporting technologies, provides a high-level infrastructure to allow all kinds of structured information to be exchanged and manipulated by a common set of tools and languages.

The information flow between the two environments is implemented through an *extractor* and a *reflector*. The extractor is a software module that takes the XMI representation of the UML specification, possibly enriched with tagged values provided by the user, and produces a program written in a process calculus that contains the relevant information for analysis and simulation. The *reflector* takes the output of the analyzer or simulator, together with the original XMI file, and alters the XMI file accordingly with tagged values or constraints. We also need the UML tool to be able to display such information. To automate the process, providing an "Apply property check" or "Run simulation" menu option, for example, we need some way of adding scripts to tools.

The approach described above has been introduced and developed by the EU project DEGAS in the FET global computing proactive initiative (`http://www.omnys.it/degas`).

We will mainly concentrate here on the kernel environment by introducing the stochastic π-calculus and showing how it can be used to model, analyze and simulate complex systems in the field of global computing and systems biology.

2 THE STOCHASTIC π-CALCULUS

We review the π-calculus (Milner et al. 1992), a model of concurrent communicating processes based on the notion of *naming*, and we introduce its stochastic semantics.

Definition 2.1 *$\mathcal{N}$ is a countable infinite set of* names *ranged over by $a, b, \ldots, x, y, \ldots$ and $\mathcal{S} = \{\tau_0, \tau_1, \tau_2, \ldots\}$ is a countable infinite set of* invisible actions *ranged over by τ_i, with $\mathcal{N} \cap \mathcal{S} = \emptyset$. We also assume a set of* agent identifiers*, each with an arity, ranged over by $A, A_1, \ldots$.* Processes *in $\mathcal{P}$, ranged over by $P, Q, R, \ldots$ are defined as*

$$P ::= \mathbf{0} \mid X \mid \pi.P \mid (\nu x)P \mid [x = y]P \mid P|P \mid P + P \mid A(y_1, \ldots, y_n)$$

where π may be either $x(y)$ for input*, or $\overline{x}y$ for* output *(where x is the* subject *and y is the* object*) or τ_i for* silent *moves. The order of* precedence *among the operators is the order* (*from left to right*) *listed above. Hereafter, the trailing* **0** *will be omitted.*

In the above definition we used a set of silent moves to distinguish their different durations.

We use μ as a metavariable for transition labels. We introduce set $\mathcal{A}$ of visible actions ranged over by α (i.e., $x(y)$ for input, $\overline{x}y$ for free output, and $\overline{x}(y)$ for bound output. The effect of a bound output is vanishing a ν operator. Consider for instance the transition $Q = (\nu x)\overline{y}x.P \xrightarrow{\overline{y}(x)} P$. The intuition behind this operation is to make the private name x of Q available to the external environment. In fact, operator ν can be interpreted as a delimiter of an environment, while the bound output is an open of that environment). Note that the transition labels differ from prefixes π because of the presence of bound outputs.

We recall the notion of free names $fn(\mu)$, bound names $bn(\mu)$, and names $n(\mu) = fn(\mu) \cup bn(\mu)$ of a label μ; only the bound names are the objects of input and of the bound output. Functions *fn*, *bn* and *n* are extended to processes by inducing on their syntax and considering input prefixes and ν operators as binders. We define the *structural congruence* $\equiv$ on processes as the least congruence that satisfies the following clauses:

- $P \equiv Q$ if P and Q differ only in the choice of bound names (α-equivalent),
- $(\nu x)(\nu y)P \equiv (\nu y)(\nu x)P$.

We sometimes write $(\nu x, y)P$ for $(\nu x)(\nu y)P$. Each agent identifier A has a unique defining equation in the form $A(\tilde{y}) = P$ (hereafter, $\tilde{y}$ denotes $y_1, \ldots, y_n$), where the y_i are all distinct and are the only free names in P.

We define our *enhanced labels* in the style of Degano and Priami (1999, 2001). A transition label records the inference rules used during its deduction, besides the action itself. We call *proof term* the encoding of the proof in an enhanced label. Finally, we use an ℓ function that takes an enhanced label to the corresponding standard action label.

Definition 2.2 *If $\mathcal{L} = \{\|_0, \|_1\}$ with $\chi \in \mathcal{L}^*$, $\mathcal{O} = \{+_0, +_1, =_m, (\nu x), (\tilde{y})\}$ with $o \in \mathcal{O} \cup \mathcal{L}$, and if $\vartheta \in (\mathcal{L} \cup \mathcal{O})^*$, then the set Θ of* enhanced labels *(with metavariable θ) is defined by the following syntax:*

$$\theta ::= \vartheta\alpha \mid \vartheta\tau_i \mid \vartheta\langle \|_0\vartheta_0\alpha_0, \|_1\vartheta_1\alpha_1 \rangle$$

where $\alpha_0 = x(y)$ iff α_1 is either $\overline{x}y$ or $\overline{x}(y)$, and vice versa. Function ℓ is defined as $\ell(\vartheta\alpha) = \alpha$, $\ell(\vartheta\tau_i) = \ell(\vartheta\langle \|_0\vartheta_0\alpha_0, \|_1\vartheta_1\alpha_1 \rangle) = \tau$.

A $+_0$ ($+_1$) tag means that a nondeterministic choice has been made in favour of the left (right) component. Similarly, a $\|_0$ ($\|_1$) tag records that the left (right) component of a parallel composition is moving. Restriction is reported on the labels to record that a filter has been passed. We record the resolution of a matching through $=_m$ tag, where m is the size of the data to be compared. Communications are labelled by a pair instead of a τ to show the components

Table 1: Proved Transition System for the π-Calculus

$$Act: \pi.P \xrightarrow{\pi} P \qquad Ide: \frac{P\{\tilde{y}/\tilde{x}\} \xrightarrow{\theta} P'}{Q(\tilde{y}) \xrightarrow{(\tilde{y})\theta} P'}, Q(\tilde{x}) = P \qquad Res: \frac{P \xrightarrow{\theta} P'}{(\nu x)P \xrightarrow{(\nu x)\theta} (\nu x)P'}, x \notin n(\ell(\theta))$$

$$Par_0: \frac{P \xrightarrow{\theta} P'}{P|Q \xrightarrow{||_0\theta} P'|Q}, bn(\ell(\theta)) \cap fn(Q) = \emptyset \qquad Par_1: \frac{P \xrightarrow{\theta} P'}{Q|P \xrightarrow{||_1\theta} Q|P'}, bn(\ell(\theta)) \cap fn(Q) = \emptyset \qquad Open: \frac{P \xrightarrow{\vartheta \overline{x} y} P'}{(\nu y)P \xrightarrow{\vartheta \overline{x}(y)} P'}, x \neq y$$

$$Sum_0: \frac{P \xrightarrow{\theta} P'}{P + Q \xrightarrow{+_0\theta} P'} \qquad Sum_1: \frac{P \xrightarrow{\theta} P'}{Q + P \xrightarrow{+_1\theta} P'}$$

$$Com_0: \frac{P \xrightarrow{\vartheta \overline{x} y} P',\ Q \xrightarrow{\vartheta' x(w)} Q'}{P|Q \xrightarrow{\langle ||_0 \vartheta \overline{x} y, ||_1 \vartheta' x(w)\rangle} P'|Q'\{y/w\}} \qquad Close_0: \frac{P \xrightarrow{\vartheta \overline{x}(y)} P',\ Q \xrightarrow{\vartheta' x(w)} Q'}{P|Q \xrightarrow{\langle ||_0 \vartheta \overline{x}(y), ||_1 \vartheta' x(w)\rangle} (\nu y)(P'|Q'\{y/w\})},\ y \notin fn(Q)$$

which interacted (and proof of the relevant transitions). We also record in the labels the actual parameters $\tilde{y}$ of a definition.

Our transition system for the π-calculus is shown in Table 1. A *variant* of $P \xrightarrow{\mu} Q$ is a transition which only differs in that P and Q have been replaced by structurally congruent processes, and μ has been α-converted, where a name bound in μ includes Q in its scope (Milner et al. 1993). The transitions in the conclusion of each rule stand for all their variants. The Com_1 and $Close_1$ rules are obvious and are therefore omitted.

Hereafter, we write a transition as $P \xrightarrow{\theta} Q$ only if it is deductible according to the inference rules in Table 1; we simply write it as θ, when it is unambiguous.

Definition 2.3 *A* proved transition system *is a quadruple $\langle \mathcal{P}, \Theta, \rightarrow, P_0 \rangle$, where $\mathcal{P}$ is the set of states (processes), Θ is the labelling alphabet, $\rightarrow$ is the transition relation defined in Table 1, and $P_0 \in \mathcal{P}$ is the initial state.*

We now define proved computations.

Definition 2.4 *If $P_0 \xrightarrow{\theta} P_1$ is a transition, then P_0 is the* source *of the transition and P_1 is its* target. *A* proved computation *of P_0 is a sequence of transitions $P_0 \xrightarrow{\theta_0} P_1 \xrightarrow{\theta_1} \ldots$ such that the target of any transition is the source of the next one. We let ξ, ξ' range over proved computations. The notions of source and target are extended to computations.*

2.1 Stochastic Semantics

We now show how to derive a probabilistic distribution F from a θ label. The intended meaning of F is the cost of execution (duration) of the action $\mu = \ell(\theta)$. The actual cost of μ depends on the basic operations that the run-time support of the target architecture performs for firing μ. For example, the resolution of a choice imposes various operations on the target architecture such as checking the ready list or implementing fairness policies. An action fired after a choice costs more than the same action occurring deterministically. The other operations of our calculus reflect analogous routines of the run-time support and delay the execution of an action as well. Therefore, we first assign a cost to the transition corresponding to μ on a dedicated architecture that only has to perform μ. We then model the performance degradation due to the run-time support by introducing a scaling factor for any routine implementing the transition. The new semantics tconsiders the target architecture on which a system is run.

We derive the distributions of transitions by inspecting the syntactical contexts into which the actions which originate them are plugged. In fact, the context in which a μ action occurs represents the operations that the target machine performs for firing μ just because the structural operational semantics of a language specifies its abstract machine in a syntax-driven logical style. Accordingly, a linearization of a transition deduction (a proof term θ) represents the execution of the corresponding run-time support routines on the target machine.

For instance, look again at the sample deduction $\theta = +_0||_0 +_0 \overline{a}(x)$ reported at the end of the previous section. The enhanced label expresses that the abstract machine resolves two choices in favour of the left alternatives, thus adding extra costs to the output operation. Similarly, the selection of the left component of the parallel composition will have a cost depending on the allocation of processes and scheduling policies. The bound output means that the abstract machine has to handle the data structure representing the process environment in order to export the name x which is required to be fresh.

Following what is discussed above, we assign a cost to each inference rule of the operational semantics via a \$ function. In other words, the occurrence of a transition receives a duration time computed according to its deduction. There is no need to fix a \$ function here, and we let it be a parameter of the definition of our model. For the sake of

simplicity, we assume that $\$_a(\mu) = F_\mu \in \mathcal{F}$ (hereafter, we use $\mathcal{F}$ to denote a set of continuous probabilistic distribution functions) and that the slow-down factor is $\$_o(\vartheta) = r \in [1, +\infty)$. $\$_o$ can be defined by inducing on the structure of ϑ. Eventually, we define $\$$ by composing $\$_a(\mu)$ and $\$_o(\vartheta)$, and by taking synchronization pairs into account.

Note that determining the distribution of synchronizations is a key point in distinguishing different proposals of stochastic process algebras. Here we demand its computation to function $\$$. This means that the distributions of synchronization vary according to the context into which they are plugged and to the architectures on which the partners run. We thus reduce the selection of distributions to the selection of suitable architectures and placement of processes. This way the designer may abstract from stochastic details and concentrate on the characteristics of the hardware. Examples of application of this mechanism can be found in Nottegar et al. (2001).

We now need to eliminate the nondeterminism introduced by the choice operator from stochastic transition systems. Hence, we introduce a *race condition* that selects the transition to be fired among the ones enabled in a state. All the enabled transitions attempt to proceed, but only the fastest one succeeds. This mechanism makes the nondeterministic choice a probabilistic one. Note that the continuous nature of probabilistic distributions ensures that the probability of two transitions ending simultaneously is 0. Moreover, as the duration of transitions is expressed by random variables, different transitions are selected on different attempts.

To get stochastic behavior from our transition systems, we must update the distributions of the random variables that express the time interval associated with transitions in correspondence with branching points. The new transition system is called stochastic.

Definition 2.5 *The quadruple* $\langle \mathcal{P}, \Theta \times \mathcal{F} \times [0..1], \rightarrow, P_0 \rangle$ *is the* stochastic transition system *associated with process* P_0*, where the real in* $[0..1]$ *denotes transition occurrence probability. The relation* $\rightarrow$ *is defined as*

$$\frac{P \xrightarrow{\theta_i} P_i}{P \xrightarrow{\theta_i, \tilde{F}_i, p_i} P_i}$$

where

$$\tilde{F}_i = \frac{\int_0^t f_i(x) \cdot \prod_{\substack{P \xrightarrow{\theta_j} P_j \\ i \neq j}} (1 - \$(\theta_j)(x))dx}{\int_0^\infty f_i(x) \cdot \prod_{\substack{P \xrightarrow{\theta_j} P_j \\ i \neq j}} (1 - \$(\theta_j)(x))dx}, \text{ and}$$

$$p_i = \int_0^\infty f_i(t) \cdot \prod_{\substack{P \xrightarrow{\theta_j} P_j \\ i \neq j}} (1 - \$(\theta_j)(t))dt.$$

The labels distinguish stochastic and proved transitions.

3 GLOBAL COMPUTING

In this section we mainly concentrate on verification, while the next section will introduce simulation. Of course both approaches can be used in any applicative domain. We start introducing an enabling relation between transitions that will be used to obtain a performance congruence (Degano and Priami 1999). This relation between transitions will be used in the next section to deal with general distributions when the random variables associated with the transitions of a computation are not independent. Hereafter we write for the sake of readability ϑ meaning the string obtained from ϑ by deleting all the proved tags except for $||_i$.

We first define structural dependencies. A transition labelled $\vartheta\mu$ depends on a previous transition labelled $\vartheta'\mu'$ if ϑ' is a prefix of ϑ (the tuning needed to cover communications is explained below). The underlying idea is that the two transitions have been derived using the same initial set of rules and are thus nested in a prefix chain (or they are connected by communications in a similar way).

Definition 3.1 *If* $P_0 \xrightarrow{\theta_0} P_1 \xrightarrow{\theta_1} \ldots \xrightarrow{\theta_n} P_{n+1}$ *is a proved computation, and hereafter* $i, j \in \{0, 1\}$*, then* θ_n *has a* direct structural dependency *on* θ_h*,* $h < n$*,* $(\theta_h \preceq^1_{str} \theta_n)$ *iff*

- $\theta_n = \vartheta\mu$, $\theta_h = \vartheta'\mu'$ *and* ϑ' *is a prefix of* ϑ*; or*
- $\theta_n = \vartheta\mu$, $\theta_h = \vartheta'\langle\vartheta'_0\mu'_0, \vartheta'_1\mu'_1\rangle$ *and* $\exists i \,.\, \vartheta'\vartheta'_i$ *is a prefix of* ϑ*; or*
- $\theta_n = \vartheta\langle\vartheta_0\mu_0, \vartheta_1\mu_1\rangle$, $\theta_h = \vartheta'\mu'$, $\exists i \,.\, \vartheta'$ *is a prefix of* $\vartheta\vartheta_i$*; or*
- $\theta_n = \vartheta\langle\vartheta_0\mu_0, \vartheta_1\mu_1\rangle$, $\theta_h = \vartheta'\langle\vartheta'_0\mu'_0, \vartheta'_1\mu'_1\rangle$, $\exists i, j \,.\, \vartheta'\vartheta'_j$ *is a prefix of* $\vartheta\vartheta_i$.

The structural dependencies *of* θ_n *are obtained by reflexive and transitive closures of* $\preceq^1_{str}$*, i.e.,* $\preceq_{str} = (\preceq^1_{str})^*$.

The last two items in Def. 3.1 say that a θ transition enables a communication if it enables one of its components. Also, we need the transitive closure of $\preceq^1_{str}$ to implement the cross inheritance of the causes of the communication partners for the residual processes.

Finally, we show how proved computations can be relabeled to take enabling relation into account.

Definition 3.2 *Given a proved computation* $\xi = P_0 \xrightarrow{\theta_0} P_1 \xrightarrow{\theta_1} \ldots \xrightarrow{\theta_n} P_n$*, its associated enabling computation* $Et(\xi)$ *is derived by relabelling any transition* θ_k *as* et_k*, where*

$$et_k = \begin{cases} \tau & \ell(\theta_k) = \tau \\ \langle \ell(\theta_k), \{h \neq k | \theta_h \preceq_{str} \theta_k, \ell(\theta_h) \neq \tau\} \rangle & o.w. \end{cases}$$

By abuse of notation we will sometimes write $Et(\theta_k)$ in place of et_k.

4 EXPONENTIAL DISTRIBUTIONS

For the sake of presentation, we limit ourselves to exponential distributions (for general distributions see Priami 2002). An exponential distribution with rate r is a function $F(t) = 1 - e^{-rt}$, where t is the time parameter. The parameter r determines the shape of the curve. The greater the r parameter, the faster $F(t)$ approaches its asymptotic value. The probability of performing an action with parameter r within time t is $F(t) = 1 - e^{-rt}$, so r determines the time Δt needed to obtain a probability near to 1. The exponential density function is $f(t) = re^{-rt}$.

Exponential distributions have the *memoryless property*. Roughly speaking, transitions occur independently of when the last transition occurred. In other words, how long the transition waits before completion does not depend on how long it has already waited. Thus, the time elapsed by an activity in a state where another one is the fastest is useless. This means that any time a transition becomes enabled, it restarts its elapsing time just as it would the first time it is enabled. Consequently, the treatment of enabling memory discipline has no counterpart in a pure exponential setting.

Let us discuss how the definition of the cost function \$ changes. First, we say $\$_a(\mu) = \lambda \in I\!R^+$, where λ is the single parameter uniquely describing an exponential distribution. Then, $\$_o(\vartheta) = r \in (0..1]$. We can define $\$_o$ as in Sect. 2.1 by simply exchanging $\sum_{o\in\vartheta}$ with $\prod_{o\in\vartheta}$ to ensure that $\$_o(\vartheta) \leq 1$. Since we will use r as a multiplicative factor for λ in the definition of \$, the interval $(0..1]$ is the domain of a slowing-down parameter. Similarly to $\$_o$, we have $f_{\langle\rangle} : \mathcal{L}^* \times \mathcal{L}^* \to (0..1]$.

Definition 4.1 *The function* $\$: \Theta \to I\!R^+$ *is defined as*

$$\$(\vartheta\,\mu) = \$_o(\vartheta) \times \$_a(\mu)$$
$$\$(\vartheta\,\langle\vartheta_0\alpha_0, \vartheta_1\alpha_1\rangle) = f_{\langle\rangle}(\underline{\vartheta_0}, \underline{\vartheta_1}) \times \min\{\$(\vartheta\,\vartheta_0\alpha_0), \$(\vartheta\,\vartheta_1\alpha_1)\}$$

Our knowledge on the kind of distributions we are dealing with enables us to state the following Theorem.

Theorem 4.2 *Given a process* P*,*

$$R_P = \sum_{P \xrightarrow{\theta_i,\lambda_i} P_i} \lambda_i$$

is the exit rate *of* P*. Then, the probability of* $P \xrightarrow{\theta_i,\lambda_i} P_i$ *is*

$$\frac{\lambda_i}{R_P};$$

the distribution of the random variable T_i *which describes the time interval associated with* $P \xrightarrow{\theta_i,\lambda_i} P_i$ *is*

$$\tilde{F}_i(t) = 1 - e^{-R_P t};$$

the apparent rate of an action a *in* P *is*

$$r_a(P) = \frac{1}{R_P} \sum_{\substack{P \xrightarrow{\theta_j,\lambda_j} P_j \\ \ell(\theta_j)=a}} \lambda_j;$$

the probability of $P \xrightarrow{\theta_i,\lambda_i} P_i$*,* $\ell(\theta_i) = a$*, given that an action* a *occurred is*

$$\frac{\lambda_i}{\sum_{\substack{P \xrightarrow{\theta_j,\lambda_j} P_j \\ \ell(\theta_j)=a}} \lambda_j}.$$

In spite of the interleaving nature of exponential distributions, we still use the structural enabling relation to yield the congruence result of our equivalence.

Definition 4.3 *A binary relation* $\mathcal{S}$ *on pairs* $\langle P, \xi\rangle$ *of processes and computations is an* exponential performance bisimulation *if* $\langle P, \xi\rangle\,\mathcal{S}\,\langle Q, \xi'\rangle$ *implies that for any equivalence class* C *originating from* $\mathcal{S}$

$$\forall\theta\,.\,\ell(\theta) \in \{\overline{x}y, \overline{x}(y), \tau, x(y)\}\,.$$
$$\gamma(\langle P, \xi\rangle, \theta, C) = \gamma(\langle Q, \xi'\rangle, \theta', C),$$
$$Et(\theta) = Et(\theta') \text{ and } bn(\ell(\theta)) \notin \mathrm{fn}(P, Q)$$

where

$$\gamma(\langle P, \xi\rangle, \theta, C) = \sum_{\langle P_i,\xi_i\rangle\in C, \ell(\theta_i)=\ell(\theta)} \gamma(\langle P, \xi\rangle, \theta, \langle P_i, \xi_i\rangle) = \sum_{\langle P_i,\xi_i\rangle\in C, \ell(\theta_i)=\ell(\theta)} r_i$$

where r_i *is the exponential distribution associated with a* $P \xrightarrow{\theta_i,r_i} P_i$ *transition where* $\ell(\theta_i) = \ell(\theta)$*.* P *is exponentially performance bisimilar to* Q *(written* $P \approx^e_P Q$*) if there is an exponential performance bisimulation* $\mathcal{S}$ *such that* $\langle P, \epsilon\rangle\,\mathcal{S}\,\langle Q, \epsilon\rangle$*.*

Remember that the condition $Et(\theta) = Et(\theta')$ in the above definition ensures that γ defines the *total conditional transition rate* as defined in Hillston (1996) for PEPA.

We refer any reader interested in exponential distributions to Nottegar et al. (2001) for case studies based on our framework.

As an example we consider a multi-server multi-queue system (MSMQ, for short) which is an extension of a classical polling system to include more than one server. This system has already been studied in the settings of

stochastic process algebras (Hillston 1996) and of stochastic Petri nets (Marsan et al. 1991, Ibe and Trivedi 1990). We consider here a little variant of Hillston (1996), which stresses mobility issues. In our version the routing of servers is state-dependent instead of state-independent as in the original presentation. In particular, servers do not move randomly between nodes, but consider whether a node already has a server and whether it has something to perform. In this respect, our specification is not a proper MSMQ system, but it enables us to cope with the general problem of remote servers dispatching agents to clients.

We consider a system made up of two independent servers that are routed between two nodes, each with a single place buffer. Two nodes suffice for generating the smallest configuration on which we can comfortably illustrate the features of our model; having more nodes would only make the example longer. We assume that a customer occupies a place in the node until service is completed. We rely on $HO\pi$ to model the routing of servers to nodes via process migration. The specification of the polling system follows (assume $i = 1, 2$).

$$\begin{aligned} P &= (\nu\, x_i, r_i, s_i, p_i)((N_1 \mid N_2)|(S \mid S)) \\ N_i &= x_i(U).U \mid Node_i \\ Node_i &= in_i.s_i.Node_i + p_i.Node_i \\ S &= \overline{x}_1\langle S_1^1\rangle.r_1.S + \overline{x}_2\langle S_2^1\rangle.r_2.S \\ S_i^1 &= \overline{s}_i.\overline{r}_i.x_i(U).U + \overline{p}_i.\overline{r}_i.x_i(U).U. \end{aligned}$$

Servers migrate to the nodes along the links x_i. Once a server has sent its agent S_i^1 onto a node N_i, it waits for a signal $\overline{r}_i$ from S_i^1 to begin again. The agent of the server queries the node for customers waiting for service. If there is one waiting, the agent serves the customer, performing action s_i in interaction with the node N_i. Otherwise, via a pass action p_i, S_i^1 restores the initial state of N_i. In both cases, S_i^1 eventually resumes the server S, via an action $\overline{r}_i$. The arrival of a customer on node N_i is modelled by firing the action in_i, a synchronization with the operating environment. The one place buffer of N_i is implemented by blocking any other in_i action until the service of the current customer has ended.

Hillston (1996) models the polling system with a first order calculus and associates stochastic information with prefixes in the syntax. An implicit assumption is that the routing of servers takes the same time for any node in the system. We could easily relax this assumption to distinguish the time spent for moving to one node or to another, by assigning different weights to the relative distances.

The transitions of the process P are in the Appendix; there are 210 of them and they form more than 45 distinct loops. The higher-order feature of our specification reduces the number of states to 68 from 210 of the specification in Hillston (1996).

As an example of analysis we computed the usage of the link s_1 (which gives the throughput of the services on the node N_1) and we considered different performance characteristics of N_1. To model this aspect, we stipulate that all the local channels of N_1 ($x_1, s_1, \ldots$ and $x_2, s_2, \ldots$) have the same throughput. In particular we considered three cases for the throughput of the channels: 352.98, 7.74 and 6.66 Mbps. We got these values by profiling three different machines at the Department of Computer Science of the University of Pisa by using Netperf (Hewlett-Packard 1996). The corresponding usage of link s_1 is 300.089, 9.88717 and 9.64 $communications/sec$ which turns out to fit with our experimental data.

The transition system for the polling system and the throughputs above have been computed by using a prototypal tool described in Brodo et al. (2000).

5 SYSTEMS BIOLOGY

We first recall the modeling principles underlying the specification of molecular processes into stochastic π-calculus and then we introduce the way in which rates of transitions are computed. Eventually we discuss the implementation of the probabilistic run-time support of the calculus.

Biomolecular processes are carried out by networks of interacting protein molecules, each composed of several distinct independent structural parts, called *domains*. The interaction between proteins causes biochemical modification of domains (e.g. covalent changes). These modifications affect the potential of the modified protein to interact with other proteins. Since protein interactions directly affect cell function, these modifications are the main mechanism underlying many cellular functions, making the stochastic π-calculus particularly suited for their modeling as mobile communicating systems.

Processes model molecules and domains. Global channel names and co-names represent complementary domains and newly declared private channels define complexes and cellular compartments. Communication and channel transmission model chemical interaction and subsequent modifications. The actual rate of a reaction between two proteins is determined according to a constant *basal rate* empirically-determined and the concentrations or quantities of the reactants . Two different reactant molecules, P and Q, are involved, and the reaction rate is given by $Brate \times |P| \times |Q|$, where $Brate$ is the reaction's basal rate, and $|P|$ and $|Q|$ are the concentrations of P and Q in the chemical solution. However, in a chemical reaction both reactants share a single basal rate. This is resolved by letting our \$ function of the previous sections associate the basal rate with channel names.

Since reaction rates depend on the number of interacting processes, we define two auxiliary functions, $In_x, Out_x : 2^{\mathcal{P}} \times \mathcal{N} \to I\!N$ that inductively count the number of receive

and send operations on a channel x enabled in a process. The rate of a usual reaction is implemented by the three parameters r_b, r_0 and r_1, where r_b represents the basal rate, and r_0 and r_1 denote the quantities of interacting molecules, and are computed compositionally via In_x and Out_x while deducing transitions.

As an example of specification we consider the cell cycle control model (see Table 2) (Lecca and Priami 2003). The system is composed by six concurrent processes, corresponding to the main five species of proteins, which regulate the cell cycle: CYCLIN, CDK, CDH1, CKI, CDC14 plus the auxiliary process CLOCK whose meaning is explained below. First cyclin sub-units bind to CDK monomers (CYCLIN process) and make them active; then the dimers cyclin/CDK, the activator CDC14 and the CDH1 are involved in a negative feedback loop: cyclin/CDK turns on CDC14, which activates CDH1, which inhibits the cyclin/CDK activity, destroying the cyclin sub-units. The model includes also another possibility of inhibition of cyclin/CDK: the stoichiometric binding with CKI. Instead, we have neglected the inhibition of cyclin/CDK by phosphorilation of CDK sub-units (to keep the model as simple as possible). The events that our code simulates are the dimers cyclin/CDK formation, phosphorilation (de-phosphorilation) of CDH1 by CDC14 and the protein degradation. The binding of cyclin with CDK occurs through the binding site offered by cyclin on the private backbone channel *bb*. All other events occur on global channels each at different suitable rates. Phosphorilation (de-phosphorilation) of CDH1 by the catalytic unit of CDK (CDK_CATALYTIC) is mediated by *pchd1r* and *removep* global channels. The stoichiometric binding of cyclin/CDK with CKI is implemented as a local sub-process of CYCLIN process occurring on the channel *bind*.

The different reactions in which the components of the system are involved are implemented as a multiple non-deterministic choice, that is then turned into a probabilistic one by the BioSpi tool (see next section). For instance, the bound state of CYCLIN process (CYCLIN_BOUND), that identifies the cyclin/CDK dimer can undergo three reactions: cyclin sub-unit degradation (DEGCYC), binding with a CKI (CYC_CDK_CKI), to form the trimer cyclin/CDK/CKI (TRIM), or the degradation of CKI sub-unit (DEGCKI). The active form of Cdh1 protein (CDH1) can degradate the cyclin (DEGRCYC), can be inactivated (INACT) by the join with a phosphate group or can be activated by CDC14 (ACTCDC14) that removes from it the phosphor. The trimer CYC_CDK_CKI can be resolved in the dimer cyclin/CDK (DIM) or it can remain itself (NOTHING).

Finally, note that we introduce in the specification the process CLOCK for technical reasons. It drives the mechanism of sending - receiving on the channels *removecki* and *donothing* in the decomposition of the trimer cyclin/CDK/CDK.

Table 2: Stochastic π-Calculus Specification of the Cell Cycle Control Model

$SYSTEM ::= CYCLIN|CDK|CDH1|CDC14|CKI|CLOCK$
$CYCLIN ::= (\nu\, bb)BINDING\_SITE$
$BINDING\_SITE ::= (\overline{lb}\langle bb\rangle, R_4).CYCLIN\_BOUND$
$CYCLIN\_BOUND ::= DEGCYC + DEGCKI + CYC\_CDK\_CKI$
$DEGCYC ::= (\overline{degp}, R_1).\overline{degc}.0$
$DEGCKI ::= (\overline{degd}, R_3).CYCLIN\_BOUND$
$CYC\_CDK\_CKI ::= (\overline{bind}\langle bb\rangle, R_{11}).\overline{bb}.TRIM$
$TRIM ::= DIM + NOTHING$
$DIM ::= (removecki, R_9).(CDK|CYCLIN\_BOUND)$
$NOTHING ::= (donothing, R_{10}).TRIM$
$CDK ::= (lb(cbb), R_4).CDK\_CATALYTIC$
$CDK\_CATALYTIC ::= INACTCDH1 + NEWCDK + INACTCAT$
$INACTCDH1 ::= (\overline{cdh1r}, R_6).CDK\_CATALYTIC$
$NEWCDK ::= (degc, R_2).CDK$
$INACTCAT ::= (cbb, R_5).0$

$CDH1 ::= DEGRCYC + INACT + ACTCDC14$
$DEGRCYC ::= (\overline{degp}, R_1).CDH1$
$INACT ::= (cdh1r, R\_6).(\overline{pcdh1r}, R_7).CDH1$
$ACTCDC14 ::= (\overline{removep}, R_8).CDH1$

$CDC14 ::= (pcdh1r, R_7).CDC14P$
$CDC14P ::= (removep, R_8).CDC14$

$CKI ::= DEGRCKI + BINDCYC$
$DEGRCKI ::= (degd, R_3).0$
$BINDCYC ::= (bind(x), R_{11}).0$

$CLOCK ::= CLOCK1 + CLOCK2$
$CLOCK1 ::= (\overline{removecki}, R_9).CLOCK$
$CLOCK2 ::= (donothing, R_{10}).CLOCK$

$R_1 = 0.005 \quad R_2 = 0.001 \quad R_3 = 0.003$
$R_4 = 0.500 \quad R_5 = 0.300 \quad R_6 = 0.005$
$R_7 = 0.009 \quad R_8 = 0.009 \quad R_9 = 0.010$
$R_{10} = 0.017 \quad R_{11} = 0.020$

5.1 Implementation

We implemented the biochemical stochastic π-calculus as part of the BioPSI application, based on the FCP platform Logix (Silverman et al. 1987, Shapiro 1987). We devised an appropriate insulated surface syntax, and built a compiler to FCP. Two unique features of FCP made it suitable for our purposes. First, the ability to pass logical variables in messages is used to implement name passing. Second, FCP's support of guarded atomic unification allows synchronized interaction between input and output guards.

In BioPSI, each channel is an object (a persistent procedure) and is associated with a basal rate. BioPSI processes send requests to the channel, via an FCP vector. There are four kinds of requests: send, receive, send & receive (for homodimerization), and withdraw. Requests to a channel which has an infinite rate are satisfied as soon as possible. Requests to a channel which has a finite rate (> 0) are queued.

Each time that a new event is required the central BioPSI monitor and all channel objects with a finite, non-zero rate,

jointly determine a communication event. Each channel object determines a weighted rate, according to its basal rate and the numbers of send and receive offers. Based on an existing algorithm (detailed in Gillespie 1977), the monitor selects randomly among the weighted rates, and stochastically selects according to the sum of weighted rates an appropriate reaction time interval to advance a 'clock' counter. The chosen channel completes one transmission (send/receive pair), relaying the sent message to the receiver.

The completion of the send and receive requests is synchronized by the channel. In addition, other messages offered on this and other channels by the same two processes whose requests were completed, are withdrawn (mutually exclusive choice). The withdrawals are not synchronized, but they do precede continuation of their respective processes.

Each BioPSI process is transformed to an FCP procedure, and its channel set (global channels, arguments, newly declared channels and channels, bound by input, to be instantiated only following communication) is identified, thus allowing full use of channels as in the original calculus. Note, that the BioPSI process retains a segment of a short circuit, which is extended when the channel is passed to more than one process (including itself, recursively) and closed when the channel reference is no longer required. When all segments of the short circuit have been closed, the channel object terminates.

Several tracing and debugging tools are available for following a simulation. These include a full ordered and timed trace of all events, which is post-processed to produce a quantitative time-evolution for each kind of process. For example, the simulation outputs of the cell cycle model shown in Figure 1 are in agreement both with published simulation and analysis data and with experimental observations for the Nasmyth two states model.

The use of Gillespie's algorithm (Gillespie 1977) for the implementation of the race condition ensures the biochemical faithfullness of BioPSI stochastic simulations.

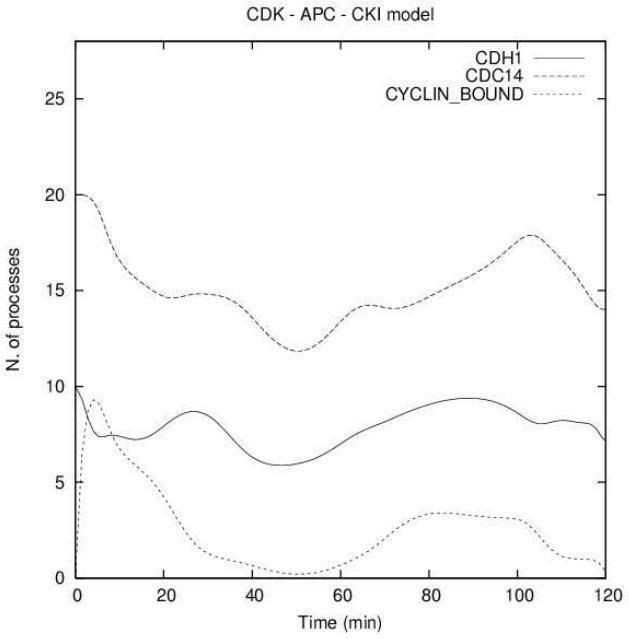

Figure 1: BioSpi Simulation Output for the Two State Nasmyth Model of Cell Cycle Control, with Time Evolution of Absolute Number of Proteins Involved in the Process: Cdh1, Cdc14 and Cyclin/CDK

ACKNOWLEDGMENTS

I would like to thank all the people involved in the DEGAS project and Paola Lecca, Aviv Regev and Ehud Shapiro for the cooperation that contributed to the results described in this paper. The author has been partially supported by the EU project IST-2001-32072 DEGAS founded under the FET proactive initiative on global computing.

REFERENCES

Booch, G., J. Rumbaugh, and I. Jacobson. 1997. *UML notation guide, version 1.1*.

Brodo, L., P. Degano, and C. Priami. 2000. A tool for quantitative analysis of pi-calculus processes. In *Proceedings of PAPM'00*, ed. R. Gorrieri. Geneva: Carleton Scientific.

Degano, P., and C. Priami. 1999. Non Interleaving Semantics for Mobile Processes. *Theoretical Computer Science* 216: 237–270.

Degano, P., and C. Priami. 2001. Enhanced operational semantics: A tool for describing and analysing concurrent systems. *ACM Computing Surveys* 33 (2): 135–176.

Gillespie, D. 1977. Exact stochastic simulation of coupled chemical reactions. *Journal of Physical Chemistry* 81 (25): 2340–2361.

Hewlett-Packard. 1996. *Netperf: A network performance benchmark, revision 2.1*. Information Networks Division, Hewlett-Packard.

Hillston, J. 1996. *A compositional approach to performance modelling*. Cambridge University Press.

Ibe, O., and K. Trivedi. 1990. Stochastic petri net models of polling systems. *IEEE Journal on Selected Areas of Communication* 8 (9): 1649–1657.

Kitano H. 1999. Systems Biology: A Brief Overview. *Science* 295: 1662–1664.

Lecca, P., and C. Priami. 2003. Cell cycle control in eukaryotes: a BioSpi model. In *Proceedings of BioConcur 2003, ENTCS*. To appear.

Marsan, M. A., S. Donatelli, F. Neri, and U. Rubino. 1991. On the construction of abstract GSPNs:an exercise in modelling. In *Proceedings of 4th PNPM*.

Milner, R., J. Parrow, and D. Walker. 1992. A calculus of mobile processes (I and II). *Information and Computation* 100 (1): 1–77.

Milner, R., J. Parrow, and D. Walker. 1993. Modal logics for mobile processes. *Theoretical Computer Science* 114: 149–171.

Nottegar, C., C. Priami, and P. Degano. 2001. Performance evaluation of mobile processes via abstract machines. *IEEE Transactions on Software Engineering* 27 (10).

OMG. 2000, November. *Xml metadata interchange (xmi) version 1.1*. OMG document 00-11-02. Available from <http://www.omg.org/>.

Priami, C. 2002. Language-based performance prediction for distributed and mobile systems. *Information and Computation* 175.

Priami, C., A. Regev, W. Silverman, and E. Shapiro. 2001. Application of a stochastic passing-name calculus to representation and simulation of molecular processes. *Information Processing Letters* 80: 25–31.

Roux-Rouquie, M., and J. Le Moigne. 2002. The systemic paradigm and its relevance to modeling biological functions. *C.R. Biologies* 325: 419–430.

Shapiro, E. 1987. Concurrent prolog: a progress report. In *Concurrent Prolog (vol. I)*, ed. E. Shapiro, 157–187. Cambridge, Massachusetts: MIT Press.

Silverman, W., M. Hirsch, A. Houri, and E. Shapiro. 1987. *The Logix system user manual, version 1.21. - concurrent prolog (vol. ii)*. MIT Press.

W3C. 2000, October. *Extensible markup language (XML) 1.0 (second edition)*. Available from <http://www.w3c.org/>.

AUTHOR BIOGRAPHY

CORRADO PRIAMI is Professor of Computer Science in the University of Trento. His e-mail address is <priami@dit.unitn.it>, and his web page is <www.science.unitn.it/priami>.