Formal Modeling and Analysis of Power-Aware Real-Time Systems[*]

Insup Lee[1], Anna Philippou[2], and Oleg Sokolsky[3]

[1]University of Pennsylvania, USA. lee@central.cis.upenn.edu

[2]University of Cyprus, Cyprus. annap@ucy.ac.cy

[1]University of Pennsylvania, USA. sokolsky@saul.cis.upenn.edu

Abstract — **The paper describes a unified formal framework for designing and reasoning about power-constrained, timed systems. The framework is based on *process algebra*, a formalism which has been developed to describe and analyze communicating, concurrent systems. The proposed extension allows the modeling of probilistic resource failures and power consumption by resources within the same formalism. Thus, it is possible to study several alternative power-consumption behaviors and tradeoffs in their timing and other characteristics. This paper describes the modeling and analysis techniques, and illustrates them with examples, including a power-aware ad-hoc network protocol.**

1 Introduction

In recent years, there have been great advances in wireless technology and portable computing. These advances have given rise to sophisticated devices and network architectures which are becoming widespread not only in the use of portable computers but also in embedded systems, as for example exist on cellular phones and digital cameras. A serious limitation of the abovementioned devices is the battery life available to them. Although a great deal of power-intensive computation has to be performed and efficiency has to be maintained, this has to be done on a limited amount of power. To cope with this fact, recently, a number of power-aware algorithms and protocols have been proposed aiming to make energy savings by dynamically altering the power consumed by a processor while still achieving the required behavior. However, in time-constrained applications often found in embedded systems, applying power-saving techniques can lead to serious problems. This is because changing the power available to tasks can affect their execution time which may lead to violation of their timing constraints and other guarantees. A challenge presented by such systems is the development of robust algorithms that incorporate power-saving techniques and task management. An example of such a proposal can be found in [9].

The main purpose of this paper is to provide a unified formal framework for designing and reasoning about power-constrained, timed systems. The framework we propose is based on *process algebra* a formalism which has been developed to describe and analyze communicating, concurrently-executing systems. The most salient aspect of process algebras is that they support the *modular* specification and verification of systems and they enable the verification of a whole system by reasoning about its parts. Process algebras are being used widely in specifying and verifying concurrent systems and they have been extended to take account of phenomena such as time and probabilistic behavior.

In this paper we present the process algebra P^{2}ACSR which extends our previous work on formal methods for real-time [6] and probabilistic systems [8] by incorporating the ability of reasoning about power consumption. The Algebra of Communicating Shared Resource (ACSR) [6] is a timed process algebra which represents a real-time system as a collection of concurrent processes. Each process can engage in two kinds of activities: communication with other processes by means of instantaneous *events* and computation by means of timed *actions*. Executing an action requires access to a set of resources and takes a non-zero amount of time measured by an implicit global clock. Resources are serially reusable. The notion of a resource, which is already important in the specification of real-time systems, additionally provides a convenient abstraction mechanism for capturing a variety of aspects of systems behavior. One such aspect is the failure of physical devices: in a probabilistic extension of ACSR, PACSR [8], resources

[*]This research was supported in part by NSF CCR-9988409, NSF CCR-0086147, NSF CISE-9703220, ARO DAAG55-98-1-0393, ARO DAAG55-98-1-0466, and ARO DAAD19-01-1-0473

were extended with the ability to fail, and were associated with a probability of failure. In P^{2}ACSR, the resource model of PACSR is further extended to reason about power consumption. Resources in P^{2}ACSR specifications are accompanied with information about the power consumption of each resource use. Thus, for each execution step requiring access to a set of power-consuming resources, we can compute the power consumed by the step and similarly for a sequence of steps.

We are interested in being able to specify and verify high-level requirements of P^{2}ACSR, and in particular to study their power-consumption behavior and tradeoffs in their timing and other characteristics. To do this we provide an operational semantics of P^{2}ACSR via *labeled concurrent Markov chains* [12], which are transition systems containing both probabilistic and nondeterministic behavior. Probabilistic behavior is present in the model due to resource failure and nondeterministic behavior due to the fact that P^{2}ACSR specifications typically consist of several parallel processes producing events concurrently. We discuss possible analysis methods that can be carried out on this model mostly based on extending and employing existing techniques proposed in the literature such as model checking, equivalence checking and long-run average behavior computation, to allow reasoning about power consumption properties. We illustrate the usefulness of the proposed formalism for a number of examples including a power-aware protocol for wireless ad-hoc networks [1]. In the latter example, we use resources to model power-consuming nodes of a mobile network, where each resource can be used at different power levels on different occasions. Furthermore, we account for the probabilistic nature of message arrival in the network by employing probabilistic resources.

The rest of the paper is organized as follows: the next section presents the P^{2}ACSR syntax and semantics. Section 3 discusses analysis techniques for P^{2}ACSR processes, and Section 4 presents the model of a power-aware ad-hoc network protocol. We conclude with some final remarks and discussion of further work.

2 The Syntax of P^{2}ACSR

As in ACSR we assume that a system contains a finite set of serially reusable resources drawn from a countably infinite set of resources $\mathcal{R}$. Resources can correspond to physical entities, such as processor units and communication channels, or to abstract notions such as message arrival. They can have a set of numerical attributes that let us capture quantitative aspects of resource-constrained computations. In general, resource attributes can be associated to the resource itself, or separately to each resource use. We illustrate the use of both kinds of attributes. As in PACSR, one attribute will capture the probability of resource failure, which will be constant throughout a system specification. A second attribute will represent power consumption, which may be different in different uses of the resource.

Probabilistic resource failures. As in PACSR, we associate with each resource a probability. This probability captures the rate at which the resource may fail. A failure may correspond to either a physical failure, such as a message loss in a communication channel, or a failure of some abstract condition, for example no message arrival when one was expected. We assume that in each execution step, resources fail independently. To capture the notion of a failed resource we also consider the set $\overline{\mathcal{R}}$ that contains for each $r \in \mathcal{R}$, its dual element $\overline{r}$, representing the *failed* resource r and write R for $\overline{\mathcal{R}} \cup \mathcal{R}$. Thus, for all $r \in \mathcal{R}$ we denote by $\mathsf{p}(r) \in [0,1]$ the probability of resource r being up in a given step, while $\mathsf{p}(\overline{r}) = 1 - \mathsf{p}(r)$ denotes the probability of r failing in a given step. The use of failed resources is useful when we need to specify a recovery from the failure.

Resources and power consumption. In order to reason about power consumption in distributed settings, the set of resources $\mathcal{R}$ is partitioned into a finite set of disjoint classes R_i, for some index set I. Intuitively, each R_i corresponds to a distinct power source which can provide a limited amount of power at any given time. This limit is denoted by c_i. Each resource $r \in R_i$ consumes a certain amount of power from the source R_i. As we will see below, the rate of power consumption is specified in *timed actions*.

2.1 The Syntax

As PACSR, P^{2}ACSR has three types of actions: instantaneous events, timed actions, and probabilistic actions. We discuss these three concepts below.

Instantaneous Events. Instantaneous actions are called *events*. Events provide the basic synchronization primitives in the process algebra. An event is denoted by a label a drawn from the set $\mathsf{L} = \mathcal{L} \cup \overline{\mathcal{L}} \cup \{\tau\}$, where if $a \in \mathcal{L}$, $\overline{a} \in \overline{\mathcal{L}}$ is its *inverse* label. The special label τ arises when two events with inverse labels are executed concurrently. We let a, b range over labels. Further, we use $\mathcal{D}_E$ to denote the domain of events.

Timed actions An action consists of several resources, each resource being used at some level of power consumption. Formally, an action is a finite set of pairs of the form (r, p) where r is a resource and p is the rate of power consumption. We let A, B range over actions and $\mathcal{D}_R$ to denote the domain of actions. The restrictions on actions state that resources used simultaneously in an action must be distinct. We denote the set of resources used in an action A as $\rho(A)$.

An example of an action is given by $\{(cpu,3),(msg,0)\}$. This action uses resource cpu, which represents a processor unit, consuming three units of power. The processor can fail with probability $\mathsf{p}(cpu)$. This action also assumes that the processor receives a message, represented by resource msg. The fact that the message may or may not arrive is modeled as a failure of resource msg. This is not a physical failure, but rather a failed assumption. The action takes place assuming that none of the resources fail. On the other hand, action $\{(cpu,3),(\overline{msg},0)\}$ takes place given that resource msg fails and resource cpu does not.

Probabilistic transitions. As already mentioned resources are associated with a probability of failure. Thus, the behavior of a resource-consuming system has certain probabilistic aspects to it which are reflected in the operational semantics of the algebra. For example consider action $\{(cpu,3),(msg,0)\}$, where resources cpu and msg have probabilities of failure 0 and 1/3, respectively, that is $\mathsf{p}(cpu) = 1$ and $\mathsf{p}(msg) = 2/3$. Then the action takes place with probability $\mathsf{p}(cpu) \cdot \mathsf{p}(msg) = 2/3$ and fails with probability 1/3.

Processes We let P, Q range over processes and we assume a set of process constants each with an associated definition of the kind $X \overset{\text{def}}{=} P$. The following grammar describes the syntax of P^{2}ACSR processes.

$$P ::= \text{NIL} \mid a.P \mid A{:}P \mid P+P \mid P\|P \mid P\backslash F \mid rec\ X.P \mid X$$

Process NIL represents the inactive process. There are two prefix operators, corresponding to the two types of actions. The first, $a.P$, executes the instantaneous event a and proceeds to P. The second, $A : P$, executes a resource-consuming action A during the first time unit and proceeds to P. As we will specify precisely when we give the semantics of the language, an action can take place if none of the resources used by it fail and assuming that it does not violate the system's power constraints. Otherwise, $A : P$ cannot execute the action and behaves as NIL. As a shorthand notation, we will write $A^n : P$ for a process that performs n consecutive actions A and then behaves as P. Process $P + Q$ represents a nondeterministic choice between the two summands. Process $P\|Q$ describes the concurrent composition of P and Q: the component processes may proceed independently or interact with one another while executing events, and they synchronize on timed actions. In $P\backslash F$, where $F \subseteq L$, the scope of labels in F is restricted to process P: components of P may use these labels to interact with one another but not with P's environment. Finally, process $rec\ X.P$ denotes standard recursion.

As an example of a process, consider the process $P \overset{\text{def}}{=} \{(cpu,3),(msg,0)\} : P_1 + \{(cpu,2),(\overline{msg},0)\} : P_2$. Process P represents a processor that can accept messages from a channel. We assume that reading the message from the channel requires additional power. Depending on whether the message arrives or not, P has two alternative behaviors. If the message arrives, that is, resource msg is up, the processor receives the message, consuming 3 units of power, and proceeds to process it as P_1. Otherwise, if the message does not arrive, msg is down and that action cannot proceed. This is specified as $\overline{msg}$ is up, and the processor consumes only 2 units of power and continues as P_2.

As a syntactic convenience, we allow P^{2}ACSR processes to be parametrized by a set of index variables. Each index variable is given a fixed range of values. This restricted notion of parameterization allows us to represent collections of similar processes concisely. For example, the parameterized process

$$P_t \overset{\text{def}}{=} t < 2 \rightarrow a_t.P_{t+1}, \quad t \in \{0..2\}$$

is equivalent to the following three processes:

$$P_0 \overset{\text{def}}{=} a_0.P_1, \quad P_1 \overset{\text{def}}{=} a_1.P_2, \quad P_2 \overset{\text{def}}{=} \text{NIL}$$

2.2 Operational Semantics

The semantics of the process algebra is given operationally by a transition system that captures the nondeterministic and probabilistic behavior of processes. As for PACSR, it is based on the notion of a *world*, which keeps information about the state of the resources of a process. Given $Z \subset \mathsf{R}$, the set of possible worlds involving Z is given by $\mathcal{W}(Z) = \{Z' \subseteq Z \cup \overline{Z} \mid x \in Z' \text{ iff } \overline{x} \notin Z'\}$, that is, it contains all possible combinations of the resources in Z being up or down. Given a world $W \in \mathcal{W}(Z)$, we can calculate its probability by multiplying the probabilities of every resource in W.

Behavior of a given process P can be given only with respect to the world P is in. A *configuration* is a pair of the form $(P,W) \in \mathsf{Proc} \times 2^{\mathsf{R}}$, representing a P^{2}ACSR process P in world W. We write S for the set of all configurations and we partition this set into the subset of probabilistic configurations, S_p, and the subset of nondeterministic configurations S_n. A configuration (P,W) is in S_n if every resource that can be used in a first step of P is included in W and it is in S_p otherwise. The semantics is given in terms of a labeled transition system whose states are configurations and whose transitions are either probabilistic or nondeterministic.

The intuition for the semantics is as follows: for a process P, we begin with the configuration $(P,\emptyset)$. As computation proceeds, probabilistic transitions are performed from probabilistic configurations to determine the status of resources immediately relevant for execution but for which there is no knowledge in the configuration's world. Once the status of a resource is determined by some probabilistic transition, it cannot change until the next timed action occurs. Once a timed action occurs, the state of resources has to be determined anew, since in each time unit resources can fail independently from any previous failures. Nondeterministic transitions (which can involve events or actions) are performed from nondeterministic configurations.

The transition relation for configurations $(P, W) \in S_n$ is presented in Table 1. In this table and hereafter, we use α, β to range over $\mathcal{D}_E \cup \mathcal{D}_R$. Further, we use the predicate $\mathsf{valid}(A)$ to distinguish actions that do not violate their power consumption requirements. Specifically, $\mathsf{valid}(A) = \bigwedge_{i \in I}(\sum_{r \in R_i, (r,p_r) \in A} p_r \leq c_i)$. Note that the symmetric versions of rules (Sum) and (Par1) have been omitted. The rules are the same as for PACSR except from the action prefix operator and the parallel composition operator. In both (Act2) and (Par3), for an action to take place, in addition to all other constraints, it must be valid according to the power requirements.

We illustrate the rules of the semantics with an example. Consider $P \overset{\text{def}}{=} \{(r_1, 1), (r_2, 2)\} : P_1 + e.P_2$. The relevant resources of P are $\{r_1, r_2\}$. From the initial configuration, $(P, \emptyset)$ we have four probabilistic transitions that determine the states of r_1 and r_2: $(P, \emptyset) \xrightarrow{\mathsf{p}(r_1)\cdot\mathsf{p}(r_2)}_p (P, \{r_1, r_2\})$, $(P, \emptyset) \xrightarrow{\mathsf{p}(r_1)\cdot\mathsf{p}(\overline{r_2})}_p (P, \{r_1, \overline{r_2}\})$, $(P, \emptyset) \xrightarrow{\mathsf{p}(\overline{r_1})\cdot\mathsf{p}(r_2)}_p (P, \{\overline{r_1}, r_2\})$ and $(P, \emptyset) \xrightarrow{\mathsf{p}(\overline{r_1})\cdot\mathsf{p}(\overline{r_2})}_p (P, \{\overline{r_1}, \overline{r_2}\})$. All of these configurations are nondeterministic since they contain full information about the relevant resources. Further, $(P, \{r_1, r_2\})$ has two nondeterministic transitions: $(P, \{r_1, r_2\}) \xrightarrow{\{(r_1,1),(r_2,1)\}}_n (P_1, \emptyset)$ and $(P, \{r_1, r_2\}) \xrightarrow{e}_n (P_2, \{r_1, r_2\})$. The other configurations allow only the e-labelled transition since either r_1 or r_2 is failed.

3 Analysis

In this section we discuss the types of analysis that can be performed on P^{2}ACSR specifications. We begin by presenting the formal model underlying P^{2}ACSR processes which is that of *labeled concurrent Markov chains* [12].

Definition 3.1 A *labeled concurrent Markov chain* (LCMC) is a tuple $\langle S_n, S_p, \mathit{Act}, \longrightarrow_n, \longrightarrow_p, s_0\rangle$, where S_n is the set of nondeterministic states, S_p is the set of probabilistic states, Act is the set of labels, $\longrightarrow_n \subset S_n \times \mathit{Act} \times (S_n \cup S_p)$ is the nondeterministic transition relation, $\longrightarrow_p \subset S_p \times (0, 1] \times S_n$ is the probabilistic transition relation, satisfying $\Sigma_{(s,\pi,t) \in \longrightarrow_p} \pi = 1$ for all $s \in S_p$, and $s_0 \in S_n \cup S_p$ is the initial state. □

We may see that the operational semantics of P^{2}ACSR yields transition systems that are LCMCs where $\mathit{Act} = \mathcal{D}_E \cup \mathcal{D}_R$ and the sets S_n, S_p are the sets of nondeterministic and probabilistic configurations, respectively. In what follows, we let ℓ range over $\mathit{Act} \cup [0, 1]$.

Computations of LCMCs arise by resolving the nondeterministic and probabilistic choices: a *computation* in $T = \langle S_n, S_p, \mathit{Act}, \longrightarrow_n, \longrightarrow_p, s_0\rangle$ is either a finite sequence $c = s_0\, \ell_1\, s_1 \ldots \ell_k\, s_k$, or an infinite sequence $c = s_0\, \ell_1\, s_1 \ldots \ell_k\, s_k \ldots$, such that $s_i \in S_n \cup S_p$, $\ell_{i+1} \in \mathit{Act} \cup [0, 1]$ and $(s_i, \ell_{i+1}, s_{i+1}) \in \longrightarrow_p \cup \longrightarrow_n$, for all $0 \leq i$. Given a computation $c = s_0\ell_1 \ldots \ell_k s_k$, we define

$$
\begin{aligned}
\mathsf{trace}\,(c) &= \ell_1 \ldots \ell_k \restriction \mathit{Act} - \{\tau\}, \\
\mathsf{time}\,(c) &= \#(\ell_1 \ldots \ell_k \restriction \mathcal{D}_R) \\
\mathsf{power}\,(c, j) &= \sum_{0 \leq i \leq k} \mathsf{pow}\,(\ell_i, j), \text{ where} \\
\mathsf{pow}\,(\ell, j) &= \begin{cases} \sum_{(r,p_r) \in \ell, r \in R_j} p_r, & \text{if } \ell \in \mathcal{D}_R \\ 0, & \text{otherwise} \end{cases}
\end{aligned}
$$

To define probability measures on computations of an LCMC the nondeterminism present must be resolved. To achieve this, the notion of a scheduler has been employed [12, 5], which is an entity that, given a partial computation ending in a nondeterministic state, chooses the next transition to be executed. Each scheduler induces a probability space [4] on the set of computations allowed by the scheduler and allows us to conclude that the probability of all computations that are a prefix of some partial computation $s_0\ell_1 s_1 \ldots \ell_k s_k$ is equal to the product of all probabilities arising along the path, that is $\Pi\{\!|\ell_i \in [0, 1] \mid 1 \leq i \leq k|\!\}$.

By using and appropriately extending techniques known from the literature on LCMC, we can perform various kinds of analysis on P^{2}ACSR processes. We distinguish the following:

1) *Model checking and Equivalence checking.* Model checking and equivalence checking are techniques that have been proposed and developed in the last twenty years, and are widely used for the verification of transition systems, including LCMCs. They can both be usefully applied to P^{2}ACSR processes.

On the one hand, model checking analysis is a verification technique aimed at determining whether a system specification satisfies a property typically expressed as a temporal logic formula. Specifically, one may capture desired liveness and safety properties in an appropriate logical framework, such as the presence of deadlock, and then test whether a P^{2}ACSR process satisfies the property by performing a search on the LCMC

(Act1)	$(a.P, B) \xrightarrow{a}_n (P, B)$	(Act2)	$(A : P, B) \xrightarrow{A}_n (P, \emptyset)$, if $\rho(A) \subseteq B$, $\mathsf{valid}(A)$
(Sum)	$\dfrac{(P_1, B) \xrightarrow{\alpha}_n (P, B')}{(P_1 + P_2, B) \xrightarrow{\alpha}_n (P, B')}$	(Par1)	$\dfrac{(P_1, B) \xrightarrow{\alpha}_n (P_1', B')}{(P_1 \| P_2, B) \xrightarrow{\alpha}_n (P_1' \| P_2, B')}$
(Par2)	$\dfrac{(P_1, B) \xrightarrow{a}_n (P_2', B'),\ \ (P_2, B) \xrightarrow{\overline{a}}_n (P_2', B')}{(P_1 \| P_2, B) \xrightarrow{\tau}_n (P_1' \| P_2', B')}$		
(Par3)	$\dfrac{(P_1, B) \xrightarrow{A_1}_n (P_1', B'),\ (P_2, B) \xrightarrow{A_2}_n (P_2', B')}{(P_1 \| P_2, B) \xrightarrow{A_1 \cup A_2}_n (P_1' \| P_2', B')}$, $\rho(A_1) \cap \rho(A_2) = \emptyset$ and $\mathsf{valid}(A_1 \cup A_2)$		
(Res2)	$\dfrac{(P, B) \xrightarrow{a}_n (P', B'),\ \ a \notin F}{(P \backslash F, B) \xrightarrow{a}_n (P' \backslash F, B')}$	(Rec)	$\dfrac{(P[rec\ X.P/X], B) \xrightarrow{\alpha}_n (P', B')}{(rec\ X.P, B) \xrightarrow{\alpha}_n (P', B')}$

Table 1: The nondeterministic relation

generated by the process. On the other hand, equivalence checking is a verification technique aimed at deciding whether one system *implements* another with respect to different notions of implementation, Various equivalence relations capturing notions of implementations have been explored for LCMCs. Among these, strong and weak bisimulations have been defined and algorithms given for their automatic verification (see for example [7]). It has been shown that bisimulations are useful for performing schedulability analysis of real-time systems [2].

2) *Probabilistic bounds on power consumption.* To extend model-checking techniques for reasoning about power consumption we may invest temporal logics with power-consumption properties. One such property involves testing whether, given a limit on the power consumption on a specific power source, with some probability we can guarantee that the system does not exceed this limit. In particular we propose the formulas $PR(i, \bowtie p, c)$ and $PR(i, \bowtie p, c, t)$ where $\bowtie \in \{\leq, \geq\}$, where the first formula expresses that for every possible scheduler of a system the consumption of power from source i does not exceed limit c with probability $\bowtie p$ and the second formula restricts the computation of interest to the first t time units.

Formally, given a LCMC T, and a scheduler of T, σ, letting $\mathsf{Paths}(T, i, \sigma, c)$ be the set of computations e of T such that $\mathsf{power}(e, i) \leq c$. T satisfies $PR(i, \bowtie p, c)$ if and only if, for all schedulers σ of T, $\mathcal{P}(\mathsf{Paths}(T, i, \sigma, c)) \bowtie p$, where $\mathcal{P}$ is the probability measure function induced by scheduler σ. Likewise, T satisfies $PR(i, \bowtie p, c, t)$ if and only, if for all schedulers σ of T, $\mathcal{P}(\mathsf{TPaths}(T, i, \sigma, c, t)) \bowtie p$, where $\mathsf{TPaths}(T, i, \sigma, c, t)$ is the set of computations e of T such that $\mathsf{power}(e, i) \leq c$ and $\mathsf{time}(e) \leq t$.

3) *Long-run average performance.* It has already been shown in the literature how to evaluate the long-run average behavior of LCMC's [3]. This is achieved by specifying the experiment of which the average behavior is to be determined and then computing this long-run average by appropriately traversing the LCMC. An experiment is described by specifying (1) the labels of LCMC that mark the end of an experiment and (2) the quantity to be measured during each an experiment for which the average is to be computed. This quantity is usually associated with labels of the LCMC under study. Average behavior is particularly interesting for power consumption studies. Average power consumption can be computed per unit of time, or if desired, per periods of interest as in the following example.

Example 1. Consider the two systems below requiring the use of a resource. The first system, P, employs a highly reliable resource r that never fails, $\mathsf{p}(r) = 1$, but consumes a large amount of power. The second system, Q, opts on using a less reliable resource r' with $\mathsf{pr}(r') = 1/2$, but consumes less power. The LCMCs for P and Q are given in Figure 1. Q keeps trying until r' is up, consumes r' and performs the event $\overline{succ}$.

$$P \overset{\text{def}}{=} \{(r, 2)\} : \overline{succ}.P$$
$$Q \overset{\text{def}}{=} \{(r', 1)\} : \overline{succ}.Q + \{(\overline{r'}, 0)\} : Q$$

We may see that although Q risks a delay in successfully using resource r', on average, it consumes less power than P per successful resource use: For experiments whose outcome is the power consumed by all actions of the experiment and whose end is marked by the label $\overline{succ}$, the the average power consumption of Q is 1. which is half of the long-run average consumption of P for the same experiment. On the other hand, considering experiments where the outcome is the time taken by actions in the experiment and whose end is again marked by the label $\overline{succ}$, the long-run average, capturing the average time necessary for a successful use of a resource, is 2 time units for Q and only 1 time unit for P.

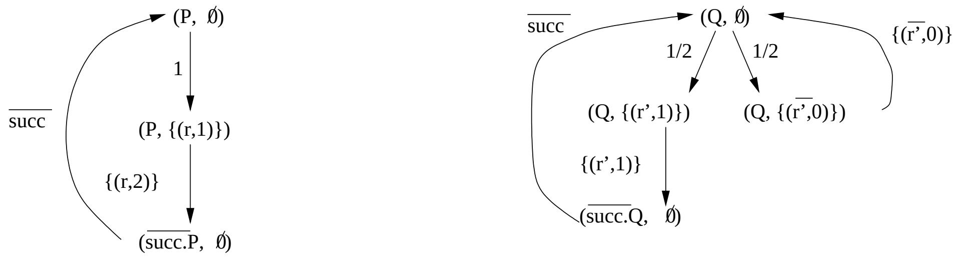

Figure 1: The LCMCs of processes P and Q

Example 2. In this example we illustrate how we can express tradeoffs in power-aware applications. Using the technique of voltage scaling [9], embedded processors can reduce their power consumption at the cost of a slower execution. Consider a periodic task with period p and deadline that is equal to the period. Assume, for the sake of the example, that the processor performs twice as fast when consuming twice the power. The execution time of the task depends on the power consumption of the processor. Let the task require e time units when the processor executes at full power, and $2 * e$ at low power. If the task shares the processor with other tasks, it will have to idle while other tasks execute and may miss its deadline.

$$\begin{array}{ll} \mathit{Task}= T_{0,0} & \\ T_{i,j} \ = j < p \rightarrow \{\} : T_{i,j+1} & i \in \{0..e-1\}, j \in \{0..p-1\} \\ \quad + \ i < e \wedge j < p \rightarrow \{(\mathit{cpu}, \mathit{low})\} : T_{i+1,j+1} & \\ \quad + \ i < e \wedge j < p \rightarrow \{(\mathit{cpu}, \mathit{high})\} : T_{i+2,j+1} & \\ T_{i,p} \ = i \geq p \wedge i = e \rightarrow \mathit{Task} & \end{array}$$

Process T has two parameters. The first one is the amount of execution time the task has accumulated in the current period and the second is the time elapsed since the start of the period. Note that when the processor, represented as the *cpu* resource, executes at full power, execution time increases more than when the power is low. The task may begin a new cycle only if the necessary amount of execution time has been accumulated. Otherwise, it misses its deadline and is blocked.

Given a set of tasks represented in this way, we can analyze whether the set is schedulable under a certain scheduling policy. The correctness criterion being that a resulting process does not deadlock can be checked either by deciding the behavior equivalence of the process to the process that idles forever, or by performing model-checking on the process to search for deadlock states. If we introduce a non-zero failure probability for the *cpu* resource, we will be able to analyze fault-tolerant scheduling algorithms (see, for example, [10]).

4 Example

In this section we illustrate the use of the proposed formalism in the specification and analysis of power-constrained embedded systems by considering the model of a power-aware protocol for wireless ad-hoc networks, PARMAC, proposed in [1]. The protocol builds upon the idea of a *beacon interval*, a power-saving approach utilized in the IEEE standard 802.11. During a beacon interval, nodes of a network may send and receive messages. In order to optimize power consumption during this interval, a reservation-based access mechanism is used. As shown in Figure 2, the beacon interval is partitioned into a reservation period (RP) and a contention-free period (CFP). During the RP period, nodes exchange control messages, following a three-way handshake protocol: when node 1 wants to send data to node 2, it sends a reservation request (RR) to node 2, listing available slots at node 1; node 2 responds with an acknowledgment (RA) that lists acceptable slots; finally, node 1 selects an acceptable slot and notifies node 2 with a second acknowledgement (RB). Note that control messages are broadcast to all nodes within the range (see Figure 3). When a node receives a message that is not addressed to it, the message is discarded. We assume that discarding a message is both faster and takes less power than processing a message.

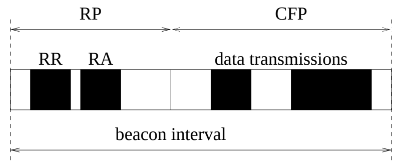

Figure 2: Structure of the Beacon Interval

Once the RP elapses, the CFP begins, when nodes send and receive messages according to the reservations made during the RP. When a node is not scheduled to send or receive, it goes to sleep until the next scheduled reservation.

The PARMAC protocol features trade-offs between power-efficiency and performance, which we capture in our model. The relative duration of RP versus CFP is a major factor. A longer RP may allow the nodes to make more reservations and thus use the CFP more efficiently. On the other hand, no data is transmitted

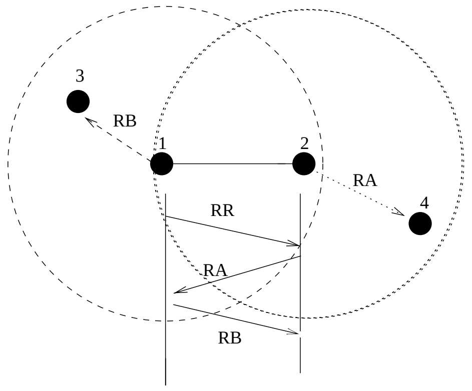

Figure 3: Reservation protocol

during the RP, so if the RP is too long, the throughput of the protocol may suffer. The optimal value of the RP likely depends on the network load.

The following simplifying assumptions were used in constructing the model. We assume that the transmission of any control or data message uses one time unit. All packets are real-time packets that have to be transmitted in one beacon interval. Packets, for which reservations have not been made in the current beacon interval, are discarded. We also assume that all new packages to be transmitted are given to the protocol at the beginning of a beacon interval. Following [1], we assume that message losses can occur only due to collisions in the medium and that access control has been performed by a higher-level protocol, so that reservations can always be made when requested.

The following parameters are used in the model. We use one resource, called *node*, to represent the power consumption of the network node in different modes. We consider three power consumption levels. A node can be *sleeping*, when it consumes the least power. An operational node that does not transmit or receive messages is *idle*. When a mode is sending, receiving, or processing a received message, it consumes more power and is considered *active*. A number of probabilistic resources capture the load the node: *in* represents arrival of a message from the network, *tous* represents that the arrived message is addressed to this node, RR represents that the arrived message is a reservation request, and RA represents that the arrived message is an acknowledgement. The protocol description is parameterized by the durations of the RP, T_{rp}, and of the CFP, T_{cfp}, and by the number n_{new} of new messages to be transmitted in the next beacon interval[1].

The model consists of two parts, corresponding to the RP and CFP periods. We begin with the part that describes the CFP. Since we are mainly interested in the power consumption aspect of the protocol, we give a simplified description of the period. Process CFP_n represents the CFP in which n reservations have been scheduled. Since we model only one node, we can assume that all n messages are sent immediately and the node sleeps for the rest of the period. Therefore, for $n \in \{0..T_{cfp}\}$ we have:

$$CFP_n \overset{\text{def}}{=} \{(node, active)\}^n : \{(node, sleep)\}^{T_{cfp}-n} : RP.$$

Processes that represent the RP are parameterized by five variables, where for a process of the form $X_{n,m,CW,b,T_{rp}}$, n is the number of messages that the node has to transmit but has not yet reserved, m is the number of reservations successfully made for this interval, CW is the value of the backoff interval, b is the backoff value, and t is the time since the beginning of the interval. At the beginning of the interval, the reservation table is empty, the number of messages to transmit is n_{new}, there is no backoff and the backoff interval has its minimum value of 1. The process listens to the medium before starting its own transmissions:

$$RP \overset{\text{def}}{=} Listen_{n_{new},0,1,0,0}$$

For the duration of the RP, process *Listen* can either receive a message from the medium, or not, as represented by processes *Receive* and *NoReceive*, respectively:

$$Listen_{n,m,CW,b,t} \overset{\text{def}}{=} Receive_{n,m,CW,b,t} + NoReceive_{n,m,CW,b,t}$$

When the RP is over, the process begins the CFP stage for the established reservations:

$$Listen_{n,m,CW,b,T_{rp}} \overset{\text{def}}{=} CFP_m$$

Consider the *Receive* process. It corresponds to the case when there is an incoming message from the medium, thus the resource *in* is up. If the message is addressed to the node, it is processed in the next time step. If the received message has been a reservation request, an acknowledgement is sent in the following time step and the reservation table is updated with the new reservation. Otherwise, no new reservation is made and the parameter m is unchanged. If the message was not addressed to the node (*tous* is down) it is discarded. Note that the node consumes less power, as well as less time, to discard a message than to process it. In the description below, we assume for the sake of simplicity that $t \leq T_{rp} - 3$. The other case can be obtained similarly.

$$\begin{aligned}
Receive_{n,m,CW,b,t} \overset{\text{def}}{=}\ & \{(node, active), (in, 0), (tous, 0)\} : \\
& (\{(node, active), (RR, 0)\} \\
& : \{(node, active)\} : Listen_{n,m+1,CW,b-2,t+3} \\
& + \{(node, active), (\overline{RR}, 0)\} : Listen_{n,m+1,CW,b-1,t+2}) \\
& + \{(node, active), (in, 0), (\overline{tous}, 0)\} : Listen_{n,m,CW,b,t+1}
\end{aligned}$$

We now turn to the case where a message is not received in the current step. Then, the node has a chance to send.

$$\begin{aligned}
NoReceive_{n,m,CW,b,t} \overset{\text{def}}{=}\ & (b = 0 \wedge n > 0) \rightarrow Send_{n,m,CW,t} \\
& + (b = 0 \wedge n = 0) \rightarrow \{(node, idle), (\overline{in}, 0)\} \\
& \qquad : Listen_{0,m,CW,0,t+1} \\
& + b > 0 \rightarrow \{(node, idle), (\overline{in}, 0)\} : Listen_{n,m,CW,b-1,t+1}
\end{aligned}$$

[1]We chose to fix this number for simplicity. It is straightforward to extend the model to attach a probability distribution to the number of new messages.

When the node can send, that is, it is not backed off ($b = 0$) and there are messages to send ($n > 0$), it performs the *Send* operation. First, it waits to see that no other nodes are sending with a delay of two time units. Then, the node sends its message and waits to receive an acknowledgement. The case analysis performed by the process in this case is similar to *Receive*, only the process is looking for *RA* instead of *RR*. If the acknowledgement is received, the number of reservations is increased by one, and the number of messages to transmit is decreased. If the medium is not idle long enough, or the acknowledgement does not come in time, the node assumes that a collision happens and backs off. In the description below, we assume for simplicity that $t \leq T_{rp} - 6$.

$$
\begin{aligned}
&Send_{n,m,CW,t} \overset{\text{def}}{=} (t < T_{rp}) \rightarrow \{(node, idle), (\overline{in}, 0)\} : \\
&\quad (\{(node, idle), (\overline{in}, 0)\} : \{(node, active)\} : \\
&\qquad (\{(node, active), (in, 0), (tous, 0)\} : \\
&\qquad\quad (\{(node, active), (RA, 0)\} : \{(node, active)\} \\
&\qquad\qquad\qquad : Listen_{n-1,m+1,CW,0,t+6} \\
&\qquad\quad + \{(node, active), (\overline{RA}, 0)\} : Listen_{n,m,CW,b-2,t+5}) \\
&\qquad + \{(node, active), (in, 0), (\overline{tous}, 0)\} \\
&\qquad\qquad\qquad : Listen_{n,m,CW,b-1,t+4} \\
&\qquad + \{(node, active), (\overline{in}, 0)\} : \ldots) \\
&\quad + \{(node, idle), (in, 0)\} : Backoff_{n,m,CW,t+2})
\end{aligned}
$$

When the process backs off, it doubles the backoff interval and selects a random backoff value from the backoff interval. This is expressed as a non-deterministic choice between all possible backoff values.

$$Backoff_{n,m,CW,t} \overset{\text{def}}{=} \sum_{0<b<2*CW} Listen_{n,m,2*CW,b,t}$$

Analysis. By using techniques already mentioned, we can perform the following kinds of analysis on the protocol: 1) *Probabilistic bounds on power consumption.* Given a limit on the power consumption, we can compute the probability that a computation will exceed this limit. 2) *Average power consumption.* We can compute the average power consumed by the node in one beacon interval. 3) *Average throughput.* We can compute the average number of data messages transmitted during one beacon interval. 4) *Protocol comparison.* Given specifications of several protocols with different power management approaches, we can order them according to their power usage. While the exact definition of such ordering is in progress, intuitively, one protocol is more power efficient than another is if, for identical loads in a beacon interval, the first protocol spends, on the average, less power than the second one.

5 Conclusions

We have presented P^2ACSR, a process algebra for resource-oriented real-time systems. The formalism allows one to model the power consumption of resources and perform power-oriented analysis of a system's behavior. We have discussed techniques for analysing high-level properties of P^2ACSR specifications that emanate from analysis techniques of labeled concurrent Markov chains and we illustrated the utility of the proposed approach using a number of examples.

As this is still work in progress a lot remains to be done. We are currently extending the PARAGON toolset [11], which allows the specification and analysis of ACSR and PACSR processes, to handle the power consumption model of P^2ACSR. This will allow the automatic verification of P^2ACSR processes and thus analyze and compare power-aware protocols including obtaining resuts for PARMAC protocol. We also aim to define ordering relations by which to relate protocols that although behaviorally similar, differ in their power consumption rates.

References

[1] M. Adamou and I. Lee. An energy-efficient real-time medium access control protocol for wireless ad-hoc networks. In *Internal Working Report*, 2001.

[2] J-Y. Choi, I. Lee, and H-L Xie. The Specification and Schedulability Analysis of Real-Time Systems using ACSR. In *Real-Time Systems Symposium*. IEEE Computer Society Press, December 1995.

[3] L. de Alfaro. How to specify and verify the long-run average behavior of probabilistic systems. In *Proceedings 13th Annual IEEE Symposium on Logic in Computer Science*, pages 454–465. IEEE, 1998.

[4] P. Halmos. *Measure Theory.* Springer Verlag, 1950.

[5] H. Hansson. *Time and Probability in Formal Design of Distributed Systems.* PhD thesis, Department of Computer Systems, Uppsala University, 1991. DoCS 91/27.

[6] I. Lee, P. Brémond-Grégoire, and R. Gerber. A process algebraic approach to the specification and analysis of resource-bound real-time systems. *Proceedings of the IEEE*, pages 158–171, Jan 1994.

[7] A. Philippou, I. Lee, and O. Sokolsky. Weak bisimulation for probabilistic systems. In *Proceedings CONCUR 00*, pages 334–349. Springer-Verlag, 2000.

[8] A. Philippou, O. Sokolsky, R. Cleaveland, I. Lee, and S. Smolka. Probabilistic resource failure in real-time process algebra. In *Proceedings CONCUR 98*, pages 389–404. Springer-Verlag, 1998.

[9] P. Pillai and K. G. Shin. Real-time dynamic voltage scaling for low-power embedded operating systems. In *Proceedings of the 18th Annual AoCM Symposium on Operating Systems Principles*, 2001.

[10] P. Sinha and N. Suri. On the use of formal techniques for analyzing dependable real-time protocols. In *Real-Time Systems Symposium*, pages 126–135. IEEE Computer Society Press, 1999.

[11] O. Sokolsky, I. Lee, and H. Ben-Abdallah. Specification and analysis of real-time systems with paragon. *Annals of Software Engineering*, 7:211–234, 1999.

[12] M. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proceedings 26th Annual Symposium on Foundations of Computer Science*, pages 327–338. IEEE, 1985.