Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 162 (2006) 221–226
www.elsevier.com/locate/entcs

A Family of Resource-Bound Real-Time Process Algebras

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

[a] *Department of Computer and Information Science, University of Pennsylvania, 200 South 33rd Street, Philadelphia, PA, USA*

[b] *Department of Computer Science, University of Cyprus, P.O. Box 20537, Nicosia, Cyprus*

Abstract

The Algebra of Communicating Shared Resources (ACSR) is a timed process algebra which extends classical process algebras with the notion of a *resource*. It takes the view that the timing behavior of a real-time system depends not only on delays due to process synchronization, but also on the availability of shared resources. Thus, ACSR employs resources as a basic primitive and it represents a real-time system as a collection of concurrent processes which may communicate with each other by means of instantaneous events and compete for the usage of shared resources. Resources are used to model physical devices such as processors, memory modules, communication links, or any other reusable resource of limited capacity. Additionally, they provide a convenient abstraction mechanism for capturing a variety of aspects of system behavior. The resulting framework combines the areas of process algebra and real-time scheduling, and can facilitate the reasoning about systems that are sensitive to deadlines, process interaction and resource availability.

In this paper we give an overview of ACSR and three of its extensions PACSR, P^2ACSR and MCSR, which take into account probabilistic failures, power consumption and multi-capacity resources.

Keywords: real-time process algebra, resource modeling, probabilistic behavior, power consumpion

1 Introduction

Modeling timing aspects of system behavior has a long history in process-algebraic formalisms. In this paper, we advocate the use of resources in the modeling of real-time systems as a means of arriving at simpler and more faithful models.

Process algebras, such as CCS [7], CSP [4], and ACP [2], have been developed to describe and analyze communicating, concurrently-executing systems. They are based on the premises that the two most essential notions in understanding complex

⋆ This research was supported in part by ARO DAAD19-01-1-0473, ARO W911NF-05-1-0182, NSF CCR-0209024, NSF CNS-0509327, and NSF CNS-0509143.

[1] Email: lee@cis.upenn.edu

[2] Email: annap@cs.ucy.ac.cy

[3] Email: sokolsky@cis.upenn.edu

1571-0661/$ – see front matter
doi:10.1016/j.entcs.2005.12.085

dynamic systems are concurrency and communication [7]. The Algebra of Communicating Shared Resources (ACSR) introduced by Lee *et. al.* [6], is a timed process algebra which can be regarded as an extension of CCS. The timing behavior of a real-time system depends not only on delays due to process synchronization, but also on the availability of shared resources. Most real-time process algebras adequately capture delays due to process synchronization. However, they abstract out resource-specific details by assuming idealistic operating environments. On the other hand, scheduling and resource-allocation algorithms used for real-time systems ignore the effect of process synchronization except for simple precedence relations between processes. The ACSR algebra provides a formal framework that combines the areas of process algebra and real-time scheduling, and, thus, can help us to reason about systems that are sensitive to deadlines, process interaction and resource availability.

The computation model of ACSR is based on the view that a real-time system consists of a set of communicating processes that use shared resources for execution and synchronize with one another. The notion of real time in ACSR is quantitative and discrete, and is accommodated using the concept of timed actions. Executing a timed action requires access to a set of resources and takes one unit of time. Resources are serially reusable, and access to them is governed by priorities. To ensure the uniform progression of time, processes execute timed actions synchronously. Similar to CCS, the execution of an event is instantaneous and never consumes any resource. The notion of communication is modeled using events through the execution of complementary events, which are then converted into an internal event. Processes execute events asynchronously except when two processes synchronize through matching events. Priorities are used to direct the choice when several events are possible at the same time.

We have extended ACSR into a family of process algebras, GCSR [1], Dense-time ACSR [3], ACSR-VP [5], PACSR [8] and P^2ACSR [10]. GCSR allows the visual representation of ACSR processes. ACSR-VP extends ACSR with value-passing capabilities, extending the class of scheduling problems that can be handled. PACSR allows the modeling of resource failure with probabilities, whereas P^2ACSR adds the notion of power consumption. Some of these extensions are informally described below.

2 Resource-Bound Processes

2.1 The Computation Model

We distinguish two types of actions: those which consume time, and those which are instantaneous. Timed actions may require access to system resources, e.g., cpu's, memory, batteries, etc. In contrast, instantaneous actions provide a synchronization mechanism between concurrent processes.

Timed Actions. A system has a finite set of serially-reusable resources, $\mathcal{R}$. An action consumes one "tick" of time and employs a set of resources, each with an integer priority. For example, action $\{(r, p)\}$ denotes the use of some resource

$r \in \mathcal{R}$ running at priority level p. The action $\emptyset$, consuming no resources, represents idling for one time unit.

Events. Instantaneous actions, or *events*, provide process synchronization in ACSR. An event is denoted by a pair (a, p), where a is the *label* of the event, and p is its *priority*. Labels represent input and output actions on channels. As in CCS, the special identity label, τ, arises when two events with input and output on the same channel synchronize.

2.2 Real-Time Processes

ACSR processes are described by the following grammar, where we assume a set of process constants each with an associated definition of the kind $C \stackrel{\text{def}}{=} P$.

$$P ::= \text{NIL} \mid (a, n).\, P \mid A{:}P \mid P + P \mid P \| P \mid P \backslash F \mid [P]_I \mid P \backslash\backslash I \mid P \triangle_t^a (P, P, P) \mid C$$

Steps of ACSR processes are constructed using the two prefix operators corresponding to the two types of actions. The process $(a, n).\, P$ executes the instantaneous event (a, n) and proceeds to P. The process $A{:}P$ executes a resource-consuming action during the first time unit and proceeds to P. The process $P + Q$ represents nondeterministic choice and the process $P \| Q$ describes the concurrent composition of P and Q. The temporal scope construct, $P \triangle_t^a (Q, R, S)$, restricts a process P by a time limit (t). If P completes its execution within this limit an *exception*, a, is thrown, in which case an exception handler (Q) is executed. If not, control is passed to a timeout process (R). In any case, P can be interrupted by a step of an interrupt process (S). Other static operators of ACSR allow us to hide the identity of certain resources $(P \backslash\backslash I)$, reserve the use of a resource for a given process $([P]_I)$, and force synchronization between processes by restricting certain events $(P \backslash F)$.

The executions of a process are defined by a timed labeled transition system (timed LTS). A timed LTS, M, is defined as $\langle \mathcal{P}, \mathcal{D}, \rightarrow, P_0 \rangle$, where $\mathcal{P}$ is a set of ACSR processes, ranged over by P, Q, $\mathcal{D}$ is a set of actions, and $\rightarrow$ is a labeled transition relation such that $P \xrightarrow{\alpha} Q$ if process P may perform an instantaneous event or timed action α and then behave as Q. $P_0 \in \mathcal{P}$ represents the initial state of the system.

Analysis of real-time systems. Within the ACSR formalism we can conduct two types of analysis for real-time scheduling: validation and schedulability analysis. Validation shows that a given specification correctly models the required real-time scheduling discipline, such as Rate Monotonic and Earliest-Deadline-First. Schedulability analysis determines whether or not a real-time system with a particular scheduling discipline misses any of its deadlines. The validation and schedulability analysis of a real-time system can be carried out by establishing appropriate equivalences between the ACSR processes representing the system under consideration and its specification.

2.3 Resource Probabilities and Actions

PACSR (Probabilistic ACSR) extends ACSR by associating each resource with a probability. This probability captures the rate at which the resource may fail. Thus, timed actions can now account for resource failure.

Timed Actions. In addition to the set of ACSR resources $\mathcal{R}$, we consider set $\overline{\mathcal{R}}$ that contains, for each $r \in \mathcal{R}$, $\overline{r}$, representing the *failed* resource r. Actions are constructed as in ACSR, but may now contain both normal and failed resources. The action $\{(r,p)\}$, cannot happen if r has failed. On the other hand, action $\{(\overline{r},q)\}$ takes place only when resource r has failed. This notation is useful for specifying recovery from failures.

Resource Probabilities. In PACSR we associate each resource with a probability at which the resource may fail. We denote by $\mathsf{p}(r) \in [0,1]$ the probability of resource r being up, while $\mathsf{p}(\overline{r}) = 1 - \mathsf{p}(r)$ is the probability of r failing. This probabilistic behavior of a resource-consuming process is reflected in the operational semantics of PACSR. For example, consider the process $\{(cpu,1)\} : \mathrm{NIL}$, with $\mathsf{p}(cpu) = 2/3$. Then, with probability $2/3$, resource *cpu* is available and the process may perform the step, while, with probability $1/3$, the resource fails and the process deadlocks.

Probabilistic Processes. The syntax of PACSR processes is the same as that of ACSR. The only extension concerns the appearance of failed resources in timed actions. Thus, it is possible on one hand to assign failure probabilities to resources of existing ACSR specifications and perform probabilistic analysis on them, and, on the other hand, to ignore failure probabilities and apply non-probabilistic analysis of PACSR specifications. The semantics of PACSR is given operationally via two transition relations that define the probabilistic and the non-deterministic behavior of processes. The resulting transition systems belong to the class of *labelled concurrent Markov chains* [11].

Analysis of probabilistic systems. We have defined a probabilistic weak bisimulation [9], which allows us to compare observable behaviors of PACSR processes similar to the case of ACSR. In addition, probabilistic information embedded in the probabilistic transitions allows us to perform quantitative analysis of PACSR specifications. In particular, we can compute the probability of reaching a given state or a deadlocked state, or we may perform model-checking of PACSR specifications [8].

2.4 Power-aware Processes

Often, we need to model consumable resources, such as power, in addition to reusable ones. An extension of PACSR, called P^2ACSR, allows us to reason about power-aware processes by specifying the amount of power consumed when a resource is accessed.

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 $\mathcal{R}_i$. Intuitively, each $\mathcal{R}_i$ corresponds to a distinct power source

which can provide a limited amount of power c_i. Each resource $r \in \mathcal{R}_i$ consumes a certain amount of power from $\mathcal{R}_i$. As in PACSR, each resource has a fixed probability of failure.

Power-consuming timed actions. Timed actions are extended to include the amount of power consumed by resources. Formally, an action is a finite set of triples of the form (r, p, c), where r is a resource, p is the priority of the resource usage and c is the rate of power consumption. The additional restriction on an action is that the total power consumption for any of the resource classes does not exceed the limit of the class. The semantics is given again as labelled concurrent Markov chains by an extension of the transition relations of PACSR

Analysis of power-aware systems. We defined a power-aware temporal logic and a model-checking algorithm for it [10], which allows us to check bounds on power consumption. We can also compute minimum and maximum power consumption within a given time frame.

2.5 Multi-capacity resources

MCSR extends the ACSR resource framework to capture *memory use* as a different kind of resource. Memory is a critical resource in size-constrained embedded systems such as mobile phones. In the design of an embedded system, we need to consider tradeoffs between memory use and the speed of the tasks in the system.

Multi-capacity Resources. The nature of memory as a resource is different from ACSR serially-reusable resources. Two processes can use the same memory, as long as the total use does not exceed the memory capacity. Therefore, we introduce a new class of resources called *multi-capacity* resources. Specifically, we partition the set of resources $\mathcal{R}$ into classes $\mathcal{R}_s$ and $\mathcal{R}_m$. Resources in $\mathcal{R}_s$ are single-capacity resources access to which is governed by priorities, as in ACSR. The resource class $\mathcal{R}_m$ contains multi-capacity resources. Resources in $\mathcal{R}_m$ are associated with a *capacity* attribute which represents the amount of resource available in a system.

Timed actions in MCSR. A timed action in MCSR consists of several resources used according to their class, and, as before, consume one unit of time. Resources in $\mathcal{R}_s$, are associated with a priority level, whereas resources in $\mathcal{R}_m$ are associated with the amount of resource used in the action. For example, for resources $cpu \in \mathcal{R}_s$, and $mem \in \mathcal{R}_m$, timed action $\{(cpu, i), (mem, u)\}$ uses resource cpu, representing a processor unit, at priority level i, while consuming u units of the resource mem, representing a memory source.

Analysis of MCSR systems. The addition of multi-capacity resources does not affect the underlying model of ACSR. Thus, bisimulations defined for ACSR also apply to the new framework. Consequently, we may test the schedulability of a real-time system containing multi-capacity resources via checking appropriate bisimulations or searching for deadlocked states.

References

[1] Ben-Abdallah, H., "GCSR: A Graphical Language for the Specification, Refinement and Analysis of Real-Time Systems", PhD thesis, Department of Computer and Information Science, University of Pennsylvania, 1996.

[2] Bergstra, J. A., and J. W. Klop. *Algebra of Communicating Processes with Abstraction*, Theoretical Computer Science, **37**:77–121, 1985.

[3] Brémond-Grégoire P., and I. Lee, *Process Algebra of Communicating Shared Resources with Dense Time and Priorities*, Theoretical Computer Science, **189**:179–219, 1997.

[4] Hoare, C. A. R.,"Communicating Sequential Processes", Prentice-Hall, 1985.

[5] Kwak, H. H., I. Lee, A. Philippou, J. Y. Choi, and O. Sokolsky, *Symbolic schedulability analysis of real-time systems*, *Proceedings of RTSS'98*, pages 409–418, IEEE Computer Society Press, 1998.

[6] Lee, I., 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, 1994.

[7] Milner, R., "Communication and Concurrency", Prentice-Hall, 1989.

[8] Philippou, A., R. Cleaveland, I. Lee, S. Smolka, and O. Sokolsky, *Probabilistic resource failure in real-time process algebra*, Proceedings of CONCUR'98, volume 1466 of *LNCS*, pages 389–404, Springer Verlag, 1998.

[9] Philippou, A., O. Sokolsky, and I. Lee, *Weak bisimulation for probabilistic systems*, Proceedings of CONCUR'00, volume 1877 of *LNCS*, pages 334-349, 2000.

[10] Sokolsky, O., A. Philippou, I. Lee, and K. Christou, *Modeling and analysis of power-aware systems*, Proceedings of TACAS '03, volume 2619 of *LNCS*, pages 409–425, 2003.

[11] Vardi, M., *Automatic verification of probabilistic concurrent finite-state programs*, Proceedings of FOCS'85, pages 327-338, 1985.