PARS: A Process Algebra with Resources and Schedulers

MohammadReza Mousavi, Michel Reniers, Twan Basten, Michel Chaudron

Eindhoven University of Technology, Post Box 513,
NL-5600 MB, Eindhoven, The Netherlands
Email: {m.r.mousavi,m.a.reniers,a.a.basten,m.r.v.chaudron}@tue.nl

Abstract. In this paper, we introduce a dense time process algebraic formalism with support for specification of (shared) resource requirements and resource schedulers. The goal of this approach is to facilitate and formalize introduction of scheduling concepts into process algebraic specification using separate specifications for resource requiring processes, schedulers and systems composing the two. The benefits of this research are twofold. Firstly, it allows for formal investigation of scheduling strategies. Secondly, it provides the basis for an extension of schedulability analysis techniques to the formal verification process, facilitating the modelling of real-time systems in a process algebraic manner using the rich background of research in scheduling theory.

1 Introduction

Scheduling theory has a rich history of research in computer science dating back to the 60's and early 70's. Process algebras have been studied as a formal theory of system design and verification since about the same time. These theories have remained separate until recently some connections have been investigated. However, combining scheduling theory in a process algebraic design still involves many theoretical and practical complications. In this paper, building upon previous attempts in this direction, we propose a process algebra for the design of scheduled real-time systems called *PARS* (for Process Algebra with Resources and Schedulers). Previous attempts to incorporate scheduling algorithms in process algebra either did not have an explicit notion of schedulers such as that of [3,12,13] (thus, coding the scheduling policy in the process specification) or scheduling is treated for restricted cases such as those of [4,10] (that only support single-processor scheduling).

Our approach to modelling scheduled systems is depicted in Figure 1. Process specification (including aspects such as causal relations of actions, their timing and resource requirements) is separated from specification of schedulers. System level specification consists of applying schedulers to process specifications, on the one hand, and composing scheduled systems, on the other hand. A distinguishing feature of our process algebra is the possibility of specifying schedulers as process terms (similar to resource-consuming processes). Another advantage of the proposed approach is the separation between process specification and

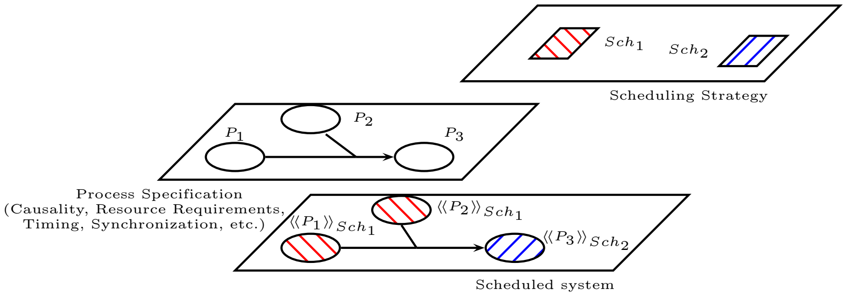
Sch_1
Sch_2
Scheduling Strategy
P_1
P_2
P_3
Process Specification
(Causality, Resource Requirements,
Timing, Synchronization, etc.)
$\langle\langle P_1 \rangle\rangle_{Sch_1}$
$\langle\langle P_2 \rangle\rangle_{Sch_1}$
$\langle\langle P_3 \rangle\rangle_{Sch_2}$
Scheduled system

Fig. 1. Schematic view of the *PARS* approach

scheduler specification that provides a separation of concerns, allows for specifying generic scheduling strategies and makes it possible to apply schedulers to systems at different levels of abstraction. Common to most process algebraic frameworks for resources, the proposed framework provides the possibility of extending standard schedulability analysis to the formal verification process.

The paper is organized as follows. We define the syntax and semantics of *PARS* in three parts. In Section 2, we build a process algebra with asynchronous relative dense time (i.e., with the possibility of interleaving timing transitions) for process specification that has a notion of resource consumption. In Section 3, a similar process algebraic theory is developed for schedulers as resource providers. Section 4 defines application of a scheduler to a process. In each section, we first give the formal syntax and semantics of our language and then explain its usage using different aspects of one or more examples. In Section 5, we compare our approach to several recent extensions of process algebra with resources and finally, Section 6 concludes the results and presents future research directions. Due to space restrictions, in this paper, we leave out a few details of the theory and some definitions. We give informal explanation for the eliminated parts and refer the interested reader to [14] for a detailed version of this paper.

2 Process Specification

The first part of specification in *PARS* consists of process specification which represents the behavior of the system with the resource requirements of its basic actions. In our framework, resources are represented by a set R. The amount of resources required by a basic durational action is modeled by a function $\rho : R \rightarrow \mathbb{R}^{\geq 0}$ (indicating required quantity of each resource). We assume the resource demand to be constant during execution of basic actions. The resources provided by schedulers are modeled using a function $\overline{\rho} : R \rightarrow \mathbb{R}^{\leq 0}$. Active tasks (actions currently being executed) that require or provide resources are represented by multisets of such tasks in the semantics. As a notational convention, we refer to the set of all multisets as $\mathbb{M}$. (We assume that the type of elements in the multiset is clear from the context.) The operator $+$ and $-$ are overloaded to represent addition and subtraction of multisets.

$$P ::= \delta \mid p(t) \mid P \,;\, P \mid P \,||\, P \mid P \,|||\, P \mid P + P \mid$$
$$\sigma_t(P) \mid \mu X.P(X) \mid \int_{x_t \in T} P(x_t) \mid \partial_{Act}(P) \mid id : P$$
$$p \in (A \times (R \to I\!R^{\geq 0})) \cup \{\epsilon\},\ t \in I\!R^{\geq 0},\ Act \subseteq A,\ x_t \in V_t, T \subseteq I\!R^{\geq 0}, id \in I\!N$$

Fig. 2. Syntax of PARS, Part 1: Process Specification

The syntax of process specification in *PARS* is presented in Figure 2. It resembles a relative dense time process algebra (such as relative dense time ACP of [2]) with empty process ($\epsilon(0)$) and deadlock (δ). The main difference with such a theory is the attachment of resource requirements to basic actions (most process algebras abstract from resource requirements by assuming abundant availability of shared resources) and our interpretation of time as duration of action execution. Basic action $\epsilon(t)$ represents idling which lasts for t time and does not require any resource. Other basic actions $(a, \rho)(t)$ are pairs of actions from the set A together with the respective resource requirement function ρ and the timing t during which the resource requirement should be provided to the action. Thus, the time annotation t should be interpreted as a duration, corresponding to the time duration which action a is to be executed; in standard timed process algebras, time annotations are usually interpreted as (absolute or relative) points in time corresponding to the occurrence or completion of an action. Terms $P \,;\, P$, $P \,||\, P$, $P \,|||\, P$, $P + P$ represent sequential composition, abstract, and strict parallel composition, and nondeterministic choice, respectively. Abstract parallel composition refers to cases where the ordering (and possible preemption) of actions has to be decided by a scheduling strategy. Strict parallel composition is similar to standard parallel composition in timed process algebra in that it forces concurrent execution of the two operands. The deadline operator applied to process P in $\sigma_t(P)$ specifies that process P should terminate within t units of time or it will deadlock. Recursion is specified explicitly using the expression $\mu X.P(X)$ where free variable X may occur in process P and is bound by μX. The term $\int_{x_t \in T} P(x_t)$ specifies continuous choice of x_t (from the set of timing variables V_t) over set T. Similar to recursion, variable x_t is bound in term P by operator $\int_{x_t \in T}$. In this paper, we are only concerned with closed terms (processes that do not have free recursion or timing variables). To prevent process P from performing particular actions in some given set Act (in particular, to force synchronization among two parallel processes, see e.g., [2]), the encapsulation expression $\partial_{Act}(P)$ is used. Using the id : construct, process terms are decorated with identifiers (natural numbers, following the idea of [4]) which serve to group processes for scheduling purposes. Note that an atomic action is neither required to have an identifier, nor its identifier needs to be unique. Later on, in the semantics, a process identifier is augmented with a few estimations of performance measures of processes, namely relative deadline and worst-case execution time. Such a semantic identifer, in turn, is referenced by the scheduler specification domain in order to devise scheduling strategies. Precedence of binding among binary composition operators is ordered as $;, |||, ||, +$ where ; binds the strongest and + the weakest. Unary operators are followed by a pair of parentheses or they bind to the smallest possible term.

The operational semantics of process specification is given in Figure 3. States are process terms and the semantics has two types of transitions. First, time passage (by spending time on resources or idling) $\xrightarrow{M,t}$ where M is the multiset that represents the amount of resources required by the actions participating in the transition. Elements of M are of the form (ids, ρ), where ids is the set of identifiers related to the action having resource requirements ρ. The second type of transitions, $\xrightarrow{a}$, represent the completion of actions. These transitions occur when an action has used the resources it requires for the specified amount of time. We do not combine resource requirements of different actions, but keep them separate in a multiset, since they may be provided by different scheduling policies (based on their respective process identifiers). We use $\xrightarrow{\chi}$ as a shorthand for either of the two transitions. Predicate $P\surd$ refers to the possibility of successful termination of P. The semantics of process specification is the smallest transition relation (union of the time and action transition relations) and the smallest termination predicate satisfying the rules of Figure 3.

Rules **(I0)** and **(I1)** specify the transitions and termination options of idling processes. In rule **(I1)**, $\overline{0}$ is a shorthand for the function mapping all resources to zero. Rules **(A0)** and **(A1)** specify how an atomic action can spend time on its resources and after that commit. The semantics of sequential composition is captured by **(S0)**-**(S2)**. Abstract parallel composition is specified by **(P0)**-**(P4)** and strict parallel composition by **(SP0)**-**(SP3)**. In rule **(P0)**, $t \gg Q$ uses an auxiliary unary operator (called deadline shift) specifying that Q is getting t units of time closer to its deadlines. Semantics of this operator is as follows:

$$\textbf{(Sh0)}\frac{p(t') \xrightarrow{\chi} P'}{t \gg p(t') \xrightarrow{\chi} P'} \qquad \textbf{(Sh1)}\frac{t \leq t' \quad \sigma_{t'-t}(t \gg P) \xrightarrow{\chi} P'}{t \gg (\sigma_{t'}(P)) \xrightarrow{\chi} P'}$$

$$\textbf{(Sh2)}\frac{t \gg P \xrightarrow{\chi} P'}{t \gg (P \,;\, Q) \xrightarrow{\chi} P' \,;\, Q} \qquad \textbf{(Sh3)}\frac{P\surd \quad t \gg Q \xrightarrow{\chi} Q'}{t \gg (P \,;\, Q) \xrightarrow{\chi} Q'}$$

$$\textbf{(Sh4)}\frac{(t \gg P)\ Op_1\ (t \gg Q) \xrightarrow{\chi} P'}{t \gg (P\ Op_1\ Q) \xrightarrow{\chi} P'} \qquad \textbf{(Sh5)}\frac{Op_2\ (t \gg P) \xrightarrow{\chi} P'}{t \gg Op_2\ (P) \xrightarrow{\chi} P'}$$

$$\textbf{(Sh6)}\frac{t + t' \gg P \xrightarrow{\chi} P'}{t' \gg t \gg P \xrightarrow{\chi} P'} \qquad \textbf{(Sh7)}\frac{P\surd}{t \gg P\surd}$$

$$Op_1 \in \{||, |||, +\}, t, t' \in I\!R^{\geq 0} \qquad Op_2 \in \{\mu X., \int_{x_t \in T}, \partial_{Act}, id:\}$$

In the above semantics, the rules for sequential composition (**(Sh2)** and **(Sh3)**) are in line with the intuition that in scheduling theory only *ready* actions can take part in scheduling and other actions have to wait for their causal predecessors to commit. Function $\gamma(a, b)$ in rules **(P3)** and **(SP2)** specifies the result of a synchronized communication between a and b. The semantics of abstract parallel composition deviates from standard semantics of parallelism in

$$\textbf{(I0)}\,\frac{}{\epsilon(0)\surd}\qquad \textbf{(I1)}\,\frac{t' \leq t}{\epsilon(t) \xrightarrow{[(\emptyset,\overline{0})],t'} \epsilon(t-t')}$$

$$\textbf{(A0)}\,\frac{t' \leq t}{(a,\rho)(t) \xrightarrow{[(\emptyset,\rho)],t'} (a,\rho)(t-t')}\qquad \textbf{(A1)}\,\frac{}{(a,\rho)(0) \xrightarrow{a} \epsilon(0)}$$

$$\textbf{(S0)}\,\frac{P \xrightarrow{\chi} P'}{P\ ;\ Q \xrightarrow{\chi} P'\ ;\ Q}\qquad \textbf{(S1)}\,\frac{P\surd \quad Q \xrightarrow{\chi} Q'}{P\ ;\ Q \xrightarrow{\chi} Q'}\qquad \textbf{(S2)}\,\frac{P\surd \quad Q\surd}{P\ ;\ Q\surd}$$

$$\textbf{(P0)}\,\frac{P \xrightarrow{M,t} P'}{\begin{array}{c}P \mid\mid Q \xrightarrow{M,t} P' \mid\mid t \gg Q\\ Q \mid\mid P \xrightarrow{M,t} t \gg Q \mid\mid P'\end{array}}\qquad \textbf{(P1)}\,\frac{P \xrightarrow{a} P'}{\begin{array}{c}P \mid\mid Q \xrightarrow{a} P' \mid\mid Q\\ Q \mid\mid P \xrightarrow{a} Q \mid\mid P'\end{array}}\qquad \textbf{(P2)}\,\frac{P \xrightarrow{M,t} P' \quad Q \xrightarrow{M',t} Q'}{P \mid\mid Q \xrightarrow{M+M',t} P' \mid\mid Q'}$$

$$\textbf{(P3)}\,\frac{P \xrightarrow{a} P' \quad Q \xrightarrow{b} Q' \quad \gamma(a,b)=c}{P \mid\mid Q \xrightarrow{c} P' \mid\mid Q'}\qquad \textbf{(P4)}\,\frac{P\surd \quad Q\surd}{P \mid\mid Q\surd}$$

$$\textbf{(SP0)}\,\frac{P \xrightarrow{M,t} P' \quad Q \xrightarrow{M',t} Q'}{P \mid\mid\mid Q \xrightarrow{M+M',t} P' \mid\mid\mid Q'}\qquad \textbf{(SP1)}\,\frac{P \xrightarrow{a} P'}{\begin{array}{c}P \mid\mid\mid Q \xrightarrow{a} P' \mid\mid\mid Q\\ Q \mid\mid\mid P \xrightarrow{a} Q \mid\mid\mid P'\end{array}}$$

$$\textbf{(SP2)}\,\frac{P \xrightarrow{a} P' \quad Q \xrightarrow{b} Q' \quad \gamma(a,b)=c}{P \mid\mid\mid Q \xrightarrow{c} P' \mid\mid\mid Q'}\qquad \textbf{(SP3)}\,\frac{P\surd \quad Q\surd}{P \mid\mid\mid Q\surd}$$

$$\textbf{(C0)}\,\frac{P \xrightarrow{\chi} P'}{\begin{array}{c}P + Q \xrightarrow{\chi} P'\\ Q + P \xrightarrow{\chi} P'\end{array}}\qquad \textbf{(C1)}\,\frac{P\surd}{\begin{array}{c}P + Q\surd\\ Q + P\surd\end{array}}$$

$$\textbf{(D0)}\,\frac{P \xrightarrow{M,t} P' \quad t \leq t_0}{\sigma_{t_0}(P) \xrightarrow{M,t} \sigma_{t_0-t}(P')}\qquad \textbf{(D1)}\,\frac{P \xrightarrow{a} P'}{\sigma_{t_0}(P) \xrightarrow{a} \sigma_{t_0}(P')}\qquad \textbf{(D2)}\,\frac{P\surd}{\sigma_{t_0}(P)\surd}$$

$$\textbf{(E0)}\,\frac{P \xrightarrow{a} P' \quad a \notin Act}{\partial_{Act}(P) \xrightarrow{a} \partial_{Act}(P')}\qquad \textbf{(E1)}\,\frac{P \xrightarrow{M,t} P'}{\partial_{Act}(P) \xrightarrow{M,t} \partial_{Act}(P')}\qquad \textbf{(E2)}\,\frac{P\surd}{\partial_{Act}(P)\surd}$$

$$\textbf{(R0)}\,\frac{P[\mu X.P(X)/X] \xrightarrow{\chi} P'}{\mu X.P(X) \xrightarrow{\chi} P'}\qquad \textbf{(R1)}\,\frac{P[\mu X.P(X)/X]\surd}{\mu X.P(X)\surd}$$

$$\textbf{(CC0)}\,\frac{t_0 \in T \quad P(t_0) \xrightarrow{\chi} P'}{\int_{x_t \in T} P(x_t) \xrightarrow{\chi} P'}\qquad \textbf{(CC1)}\,\frac{t_0 \in T \quad P(t_0)\surd}{\int_{x_t \in T} P(x_t)\surd}$$

$$\textbf{(Id0)}\,\frac{P \xrightarrow{M,t} P'}{id : P \xrightarrow{M \oplus \widetilde{id},t} id : P'}\qquad \textbf{(Id1)}\,\frac{P \xrightarrow{a} P'}{id : P \xrightarrow{a} id : P'}\qquad \textbf{(Id2)}\,\frac{P\surd}{id : P\surd}$$

$a, b, c \in A,\ t, t' \in I\!\!R^{>0}, t_0 \in I\!\!R^{\geq 0}, \chi \in (I\!\!M \times I\!\!R^{>0}) \cup A$

Fig. 3. Semantics of *PARS*, Part 1: Process Specification

timed process algebras in that it allows for asynchronous spending of time by the two parties (rule **(P0)**). This reflects that depending on availability of resources and due to scheduling, concurrent execution of tasks can be preempted and serialized at any moment of time. Components not spending time on resources do not participate (actively) in a time transition. Rules **(C0)-(C1)** provide a semantics for nondeterministic choice. Given our interpretation of time, the choice operator does not have the property of time-determinism (which states that passage of time cannot determine choices): Starting to spend time on an action reveals the choice in the same way executing an action determines the choice in untimed process algebras. The deadline operator is defined by **(D0)-(D2)**. There is no rule for $\sigma_0(P)$ when P can only do a time step. This means that this process deadlocks (i.e., missing a deadline results in deadlock). Encapsulation is defined in rules **(E0)-(E2)** stating that the encapsulation operator prevents process P from performing actions in Act. Rules **(R0)-(R1)** and **(CC0)-(CC1)** specify the semantics of recursion and continuous choice. Note that in the semantics of continuous choice, the choice is made as soon as the process term starts making a transition. Rules **(Id0)-(Id2)** specify the semantics of id by adding the semantic identifier $\widetilde{id}$ to the multiset in the transition, where $\widetilde{id}$ is the tuple $(id, Dl(P), WCET(P))$ consisting of the syntactic id, (an estimation of) the deadline, and the worst-case execution time of P. We omit detailed definitions of Dl and $WCET$. They are defined in [14] using structural induction on process terms. Other performance measures can extend or replace this notion of semantic identifier. In semantic rule **(Id0)**, $\oplus$ stands for adding a semantic identifier to the set of identifiers of each resource-requirement function in the multiset.

The standard notion of strong bisimulation is not a congruence with respect to the operators defined in the process language. The problem lies particularly in the interaction between deadlines and abstract parallel composition. In [14], it is shown that strong bisimulation is a congruence with respect to a restricted subset of the language without the deadline operator. Also, there we define a notion of deadline-sensitive bisimulation that is a congruence. To show how the process specification language is to be used, next we specify a few common patterns from scheduling literature [5].

Example 1 (Periodic and Aperiodic Tasks) First, we specify a periodic task, consisting of an atomic action a requiring a single CPU and 100 units of memory during its computation time of t, and with period of t'.

$P_1 = \mu X.((a, \{CPU \mapsto 1, Mem \mapsto 100\})(t) \;|||\; \epsilon(t') \;;\; X)$

Note that the computation time of the periodic task may be larger than the period (which means that any feasible scheduler must allow for task parallelism). Now, suppose that the exact computation time of a is not known. However, we know that the computation time is within a (possibly infinite) interval I, then the periodic task is specified as follows.

$P_2 = \mu X.(\int_{x_t \in I}(a, \{CPU \mapsto 1, Mem \mapsto 100\})(x_t) \;|||\; \epsilon(t') \;;\; X)$

In the remainder, we use syntactic shorthand $p(I)$ instead of $\int_{x_t \in I} p(x_t)$.

Aperiodic tasks follow a similar pattern with the difference that instead of computation time, their period of arrival is not known:

$S = \mu X.((b, \{CPU \mapsto 1\})(t) \mid\mid\mid \epsilon([0, \infty)) \; ; X)$
If the process specification of the system consists of periodic user level tasks and aperiodic system level tasks (e.g., system interrupts) that are to be scheduled with different policies, the specification goes as follows:
$\mathit{SysProc} = \mathit{System} : (S) \mid\mid \mathit{User} : (P_2)$
where *System* and *User* are distinct integer id's for these two types of tasks.

Example 2 (Portable Tasks) Suppose that the task a can run on different platforms, either on a dual-processor machine on which it will take 2 units of time and 100 units of memory (during those 2 time units) or on a single processor for which it will require 4 units of time and 70 units of memory (over the 4 time units). Then it is specified as follows:
$P = (a, \{CPU \mapsto 2, Mem \mapsto 100\})(2) + (a, \{CPU \mapsto 1, Mem \mapsto 70\})(4)$

3 Scheduler Specification

The second part of system specification in *PARS* is about scheduler specification. In this part, we model availability of resources and the strategy to grant these resources to processes requiring them. This is done by using predicates referring to properties of processes eligible for receiving the resources. The syntax of scheduler specification (Sc) is similar to process specification and is specified in Figure 4. Basic actions of schedulers are predicates (*Pred*) mentioning appropriate processes to be provided with resources and the amount of resources available ($\overline{\rho}$) during the specified time (t). The predicate can refer to the syntactic identifier, deadline or worst-case execution time of processes. In the syntax of *Pred*, Id is a variable from the set of semantic identifiers V_i (with a distinguished member $\underline{\mathit{Id}}$ and typical members $Id_0, Id_1, etc.$). To refer to the specific process receiving the provided resources, we use $\underline{\mathit{Id}}$ and to refer to the processes in its context we use other members of V_i. Following the structure of a semantic identifier, Id is a tuple containing syntactic identifier ($Id.id$), deadline ($Id.Dl$) and execution time ($Id.WCET$). As in the process language, the language for predicates can be extended to other metrics of processes.

A couple of new operators are added to the ones in the process specification language. The preemptive precedence operator $\rhd$ gives precedence to the right-hand-side term (with the possibility of the right-hand side taking over the execution of the left-hand side at any point). Continuous preemptive precedence $\rangle\!\!\rangle_{x_t \in T}$ gives precedence to the least possible matching of $x_t \in T$. To be more precise, a continuous precedence operator generates a symbolic transition system with all possible $t \in T$, but, when confronted with a process, allows a transition with a particular t' if the processes confronted with cannot make a transition with $t'' \in T \wedge t'' < t'$. The non-preemptive counter-parts of precedence operators $\rhd^{\mathrm{n}}$ and $\rangle\!\!\rangle^{\mathrm{n}}_{x_t \in T}$ have the same intuition but they do not allow taking over of one side if the other side has already decided to start. Timing variables bound by continuous choice or continuous precedence operators can be used in predicates (as timing constants) and in process timings.

$$\begin{aligned} Sc \quad &::= \delta \mid s(t) \mid Sc \,;\, Sc \mid Sc \mid\mid Sc \mid Sc \mid\mid\mid Sc \mid Sc + Sc \mid \\ &\qquad \int_{x_t \in T} Sc(x_t) \mid Sc \rhd Sc \mid Sc \rhd^{\mathrm{n}} Sc \mid \mathop{|\!\rangle}_{x_t \in T} Sc(x_t) \mid \mathop{|\!\rangle}^{\mathrm{n}}_{x_t \in T} Sc(x_t) \mid \mu X.Sc(X) \\ \mathit{Pred} &::= \mathit{Id.id}\ \mathit{Op}_1\ \mathit{Num} \mid \mathit{Id.Dl}\ \mathit{Op}_1\ \mathit{time} \mid \mathit{Id.WCET}\ \mathit{Op}_1\ \mathit{time} \mid \mathit{Pred}\ \mathit{Op}_2\ \mathit{Pred} \end{aligned}$$

$$\mathit{Op}_1 ::= < \ \mid\ = \ \mid\ > \qquad \mathit{Op}_2 ::= \wedge \ \mid \vee$$

$$s \in (\mathit{Pred} \times (R \to I\!R^{\leq 0})) \cup \{\epsilon\}, t \in I\!R^{\geq 0}, x_t \in V_t, \mathit{time} \in V_t \cup I\!R^{\geq 0}, \mathit{Id} \in V_i, \mathit{Num} \in I\!N$$

Fig. 4. Syntax of *PARS*, Part 2: Scheduler Specification

The semantics of schedulers is presented in Figure 5. It induces a symbolic transition system that has predicates indicating resource grants on its labels. At this level, we assume no information about resource requiring processes that the scheduler is to be confronted with. Thus, the resource grant predicates specify the criteria that processes receiving resources should satisfy and the criteria they should falsify. The latter can be used to state that a process should not be able to perform higher priority transitions. The transition relation is of the form $\overset{M,t}{\rightarrow}$, where M is a multiset containing predicates about processes that can receive a certain amount of resources during time t. Elements of M are of the form $(\mathit{pred}, \mathit{npred}, \overline{\rho})$ where *pred* is the positive predicate that the process receiving resources should satisfy, *npred* is the negative predicate that it should falsify and $\overline{\rho}$ is the function representing the amount of different resources offered.

$$\textbf{(ScA0)}\frac{}{(p,\rho)(0)\surd} \qquad \textbf{(ScA1)}\frac{t' \leq t}{(p,\rho)(t) \overset{[(p,\mathit{false},\rho)],t'}{\rightarrow} (p,\rho)(t-t')}$$

$$\textbf{(Pr0)}\frac{P \overset{M,t}{\rightarrow} P'}{P \rhd Q \overset{M \vee_{\mathrm{neg}} en(Q),t}{\rightarrow} P' \rhd Q} \qquad \textbf{(Pr1)}\frac{Q \overset{M,t}{\rightarrow} Q'}{P \rhd Q \overset{M,t}{\rightarrow} P \rhd Q'} \qquad \textbf{(Pr2)}\frac{P\surd \quad Q\surd}{P \rhd Q\surd}$$

$$\textbf{(CPr0)}\frac{t' \in T \quad P(t') \overset{M,t}{\rightarrow} P'(t')}{\mathop{|\!\rangle}_{x_t \in T} P(x_t) \overset{M \vee_{\mathrm{neg}} (en(P(x_t)) \wedge x_t \in \lfloor T \rfloor_{t'}),t}{\rightarrow} \mathop{|\!\rangle}_{x_t \in T} P'(x_t)} \qquad \textbf{(CPr1)}\frac{t' \in T \quad P(t')\surd}{\mathop{|\!\rangle}_{x_t \in T} P(x_t)\surd}$$

$$\textbf{(NPr0)}\frac{P \overset{M,t}{\rightarrow} P'}{P \rhd^{\mathrm{n}} Q \overset{M \vee_{\mathrm{neg}} en(Q),t}{\rightarrow} P'} \qquad \textbf{(NPr1)}\frac{Q \overset{M,t}{\rightarrow} Q'}{P \rhd^{\mathrm{n}} Q \overset{M,t}{\rightarrow} Q'} \qquad \textbf{(NPr2)}\frac{P\surd \quad Q\surd}{P \rhd^{\mathrm{n}} Q\surd}$$

$$\textbf{(NCPr0)}\frac{t' \in T \quad P(t') \overset{M,t}{\rightarrow} P'}{\mathop{|\!\rangle}^{\mathrm{n}}_{x_t \in T} P(x_t) \overset{M \vee_{\mathrm{neg}} (en(P(x_t)) \wedge x_t \in \lfloor T \rfloor_{t'}),t}{\rightarrow} P'} \qquad \textbf{(NCPr1)}\frac{t' \in T \quad P(t')\surd}{\mathop{|\!\rangle}^{\mathrm{n}}_{x_t \in T} P(x_t)\surd}$$

$$t \in I\!R^{>0},\ t' \in I\!R^{\geq 0}$$

Fig. 5. Semantics of *PARS*, Part 2: Scheduler Specification

Rules **(ScA0)** and **(ScA1)** specify the semantics of atomic scheduler actions. Rule **(ScA1)** shows that a scheduler can provide its resources if the requiring process satisfies its predicate. The negative predicate of a basic scheduler is set to false (which is by default falsified). Rules **(Pr0)**-**(Pr2)** specify the semantics for the precedence operator. In these rules, $M \vee_{\mathrm{neg}}$ *pred* stands for adding *pred* as a disjunction to all negative predicates in M. Enabledness of a process term is used as a negative predicate to assure that a lower priority process cannot take over a higher priority one. The notion of enabledness is defined as follows. ($P \rightarrow$ stands for the possibility of performing a transition $P \xrightarrow{\chi} P'$ for some P' and χ. Moreover $P \nrightarrow$ stands for its negation.)

$$en((pred, \rho)(t)) \doteq pred$$

$$en(P\ ;Q) \doteq \begin{cases} en(P) \vee en(Q) & \mathit{if}\ P\surd \wedge P \rightarrow \\ en(P) & \mathit{if}\ \neg(P\surd) \\ en(Q) & \mathit{if}\ P\surd \wedge P \nrightarrow \end{cases}$$

$$en(P \mid\mid Q) \doteq en(P + Q) \doteq en(P \rhd Q) \doteq en(P \rhd^{\mathrm{n}} Q) \doteq en(P) \vee en(Q)$$

$$en(\textstyle\int_{x_t \in T} P(x_t)) \doteq en(|\!\rangle_{x_t \in T} P(x_t)) \doteq en(|\!\rangle^{\mathrm{n}}_{x_t \in T} P(x_t)) \doteq x_t \in T \wedge en(P(x_t))$$

Rules **(CPr0)-(CPr1)** present the semantic rules for the continuous precedence operators. In rule **(CPr0)**, expression $\lfloor T \rfloor_t$ is defined as $\{t' | t' \in T \wedge t' < t\}$. Note that in both preemptive precedence operators, the possibility of other options (lower or higher priority processes) always remains after making a transition. This allows for preempting or changing the resource provision at any point of time based on the processes that the scheduler is confronted with. Rules **(NPr0)-(NPr2)** and **(NCPr0)-(NCPr1)** specify the semantics of non-preemptive precedence operators. We omit the semantic rules for operators shared with process specification since they are analogous to those specified in the process specification semantics. Apart from action transition rules such as **(P1)** and **(P3)** that are absent in the semantics of schedulers, the rest of the rules in Figure 3 remain intact for this semantics.

Example 3 Consider the process specification of Example 1, where the system consists of two types of processes: User processes and system processes. Suppose that our execution platform can provide two processors and 200 units of memory. System processes have priority over user processes (in using CPUs). The following scheduler is the first attempt to specify our scheduling strategy:

$$\begin{aligned} Sch_{Mem} &= (true, \{Mem \mapsto -200\})([0, \infty)) \\ PrSch_{CPU0} &= (\underline{Id}.id = User, \{CPU \mapsto -2\})([0, \infty)) \rhd \\ &\qquad (\underline{Id}.id = System, \{CPU \mapsto -2\})([0, \infty)) \\ Sch_0 &= Sch_{Mem} \mid\mid\mid PrSch_{CPU0} \end{aligned}$$

The above specification generates a transition system that allows arbitrary time transitions providing both CPUs and 200 units of memory with negative predicate $false$ to system processes (meaning that that there is no process that can take over a system process). However, according to rule **(Pr0)**, for transitions providing CPU to user processes, the predicate $t \in [0, \infty) \wedge \underline{Id}.id = System$ is added as a negative predicate. Intuitively, this should mean that CPUs are provided to a user process if no system process is able to take that transition. However, this would prevent the user process from gaining access to its CPU requirement even if only a single CPU is used by a system process (thus, one CPU can be wasted without any reason). The following scheduler specification solves this problem by separating the scheduling process of the two CPUs:

$$\begin{aligned} PrSch_{CPU1} &= (\underline{Id}.id = User, \{CPU \mapsto -1\})([0, \infty)) \rhd \\ &\qquad (\underline{Id}.id = System, \{CPU \mapsto -1\})([0, \infty)) \\ Sch_1 &= PrSch_{CPU1} \mid\mid\mid PrSch_{CPU1} \mid\mid\mid Sch_{Mem} \end{aligned}$$

Part of the symbolic transition system of scheduler Sch_1 is depicted in Figure 6. Of course, part of the intuitive explanation given above remains to be formalized by the semantics of applying schedulers to processes where resources are provided to actual tasks (i.e., the symbolic transition of a scheduler is matched with an actual transition of a process).

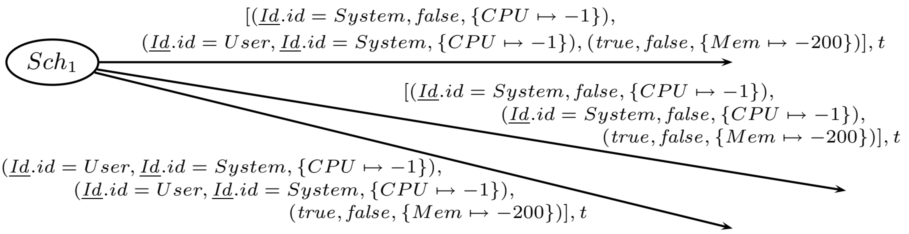
Sch_1
$[(\underline{Id}.id = System, false, \{CPU \mapsto -1\}),$
$(\underline{Id}.id = User, \underline{Id}.id = System, \{CPU \mapsto -1\}), (true, false, \{Mem \mapsto -200\})], t$
$[(\underline{Id}.id = System, false, \{CPU \mapsto -1\}),$
$(\underline{Id}.id = System, false, \{CPU \mapsto -1\}),$
$(true, false, \{Mem \mapsto -200\})], t$
$[(\underline{Id}.id = User, \underline{Id}.id = System, \{CPU \mapsto -1\}),$
$(\underline{Id}.id = User, \underline{Id}.id = System, \{CPU \mapsto -1\}),$
$(true, false, \{Mem \mapsto -200\})], t$

Fig. 6. Part of the transition system of scheduler Sch_1 in Example 3

Example 4 (Specifying Scheduling Strategies) To illustrate the scheduler specification language, we specify a few generic single-processor scheduling strategies. *Non-preemptive Round-Robin Scheduling*: Consider a scheduling strategy where a single processor is going to be granted to processes non-preemptively in an increasing order of process identifiers (from 0 to n). The following scheduler specifies the round-robin strategy.

$$Sch_{NP-RR} = \mu X.((\underline{Id} = n, \{CPU \mapsto 1\})[0, \infty) \rhd^{\mathrm{n}} \ldots \rhd^{\mathrm{n}}$$
$$((\underline{Id} = 1, \{CPU \mapsto 1\})[0, \infty) \rhd^{\mathrm{n}} (\underline{Id} = 0, \{CPU \mapsto 1\})[0, \infty))) \,;\, X$$

Rate Monotonic Scheduling: Consider the following process specifications of several periodic tasks:

$$SysProc = P_0 \mid\mid P_1 \mid\mid \ldots \mid\mid P_n$$
$$P_i = \mu X.(2i + 1) :((a_i, \rho_i)(t)) \mid\mid\mid (2i) : (\epsilon(t')) \,;\, X$$

The following scheduler specifies the preemptive rate monotonic strategy, where processes with the shortest period (the highest rate) have priority:

$$RMSch(i, x_t) =$$
$$(\underline{Id}.id = 2i + 1 \wedge Id_0 = 2i \wedge Id_0.WCET = x_t, \{CPU \mapsto 1\})([0, \infty))$$
$$RMSch = \rangle_{x_t \in I\!R^{\geq 0}} RMSch(0, x_t) + \ldots + RMSch(n, x_t)$$

Scheduler process $RMSch(i, t)$ specifies that the process receiving CPU should have an odd identifier (thus, being an action) and its corresponding period should have worst-case execution time t. Process $RMSch$ states that the processes with the least period have precedence over the others.

4 Applying Schedulers to Processes

Scheduled systems are processes resulting from application of a scheduler to processes. Syntax of scheduled systems is presented in Figure 7. In this syntax, P and Sc refer to the syntactic classes of processes and schedulers presented in the previous sections. Term $\langle\!\langle Sys \rangle\!\rangle_{Sc}$ denotes applying scheduler Sc to the system Sys and $\partial_{Res}(Sys)$ is used to close a system specification and prevent it from requiring resources in Res.

$$Sys ::= P \mid \langle\!\langle Sys \rangle\!\rangle_{Sc} \mid Sys \,;\, Sys \mid Sys \parallel Sys \mid Sys \mid\mid\mid Sys \mid Sys + Sys \mid$$
$$\partial_{Res}(Sys) \mid \sigma_t(Sys) \mid \mu X.Sys(X) \mid id : Sys$$

Fig. 7. Syntax of *PARS*, Part 3: Syntax of Scheduled Systems

The semantics of the new operators for scheduled systems is defined in Figure 8. The type of labels in the transition relation is the same as that of the transition relation in the process specification semantics of Figure 3 (hence, multisets in time transitions are resource requirement multisets). Since a process is a system by definition, all semantic rules of Figure 3 carry over to the semantics of systems. It should be understood that the variables ranging over process terms in Figure 3, are in this case ranging over the more general class of system terms.

$$\textbf{(Sys0)}\,\frac{Sys \overset{M,t}{\to} Sys' \quad Sc \overset{M',t}{\to} Sc' \quad M'' \in apply(Sys, M, M')}{\langle\!\langle Sys \rangle\!\rangle_{Sc} \overset{M'',t}{\to} \langle\!\langle Sys' \rangle\!\rangle_{Sc'}}$$

$$\textbf{(Sys1)}\,\frac{Sys \overset{a}{\to} Sys'}{\langle\!\langle Sys \rangle\!\rangle_{Sc} \overset{a}{\to} \langle\!\langle Sys' \rangle\!\rangle_{Sc}} \qquad \textbf{(Sys2)}\,\frac{Sys\surd}{\langle\!\langle Sys \rangle\!\rangle_{Sc}\surd}$$

$$\textbf{(ER0)}\,\frac{Sys \overset{M,t}{\to} Sys' \quad \forall_{(ids,\rho)\in M, r\in Res}\,\rho(r) = 0}{\partial_{Res}(Sys) \overset{M,t}{\to} \partial_{Res}(Sys')}$$

$$\textbf{(ER1)}\,\frac{Sys \overset{a}{\to} Sys'}{\partial_{Res}(Sys) \overset{a}{\to} \partial_{Res}(Sys')} \qquad \textbf{(ER2)}\,\frac{Sys\surd}{\partial_{Res}(Sys)\surd}$$

$$t \in \mathbb{R}^{>0},\ a \in A,\ Res \subseteq R$$

Fig. 8. Semantics of *PARS*, Part 3: Applying Schedulers to Processes

The application operator $\langle\!\langle Sys \rangle\!\rangle_{Sc}$ is defined by semantic rules **(Sys0)-(Sys2)**. In **(Sys0)**, the operator $apply : Sys \times \mathbb{M} \times \mathbb{M} \to \mathbb{P}\,(\mathbb{M})$ is meant to apply a multiset of resource providing predicates (third parameter) to a multiset of resource requiring tasks (second parameter) originated from a system (first parameter). The formal definition of this operator is the smallest function satisfying the following constraint:

$$\forall_{m\in M'}\ \forall_{M''\in applyTask(S,M,m,\emptyset)}\, apply(S, M'', M' - [m]) \subseteq apply(S, M, M')$$

In this statement, *applyTask* (defined below) is meant to provide the set of possible outcomes of applying a single resource providing task m to the resource requiring multiset M (the forth parameter of *applyTask* is used to keep track of resource requiring tasks checked so far to receive the provided resource). This statement means that the application of a scheduler task to a multiset of process tasks is done by taking an arbitrary scheduler task and applying it to the multiset of process tasks and starting over with the rest of scheduler tasks. The function *applyTask* is the smallest function satisfying the following constraints:

$$applyTask(S, \emptyset, m, M) \doteq \{\emptyset\}$$

$$N + N' \in applyTask(S, [(ids, \rho)] + M, (pred, npred, \overline{\rho}), M'), \text{ where}$$

$$\begin{cases} \mathit{if}\ \ pred(ids, M + M' + [(ids, \rho)]) \wedge \\ \quad \neg engage(S, M + M' + [(ids, \rho)], (pred, npred, \overline{\rho})) \\ \qquad N = [(ids, max(\overline{0}, \rho + \overline{\rho})] \wedge \\ \qquad N' \in applyTask(S, M, (pred, npred, min(\overline{0}, \overline{\rho} + \rho)), M' + [(ids, \rho)]) \\ \mathit{otherwise} \\ \qquad N = [(ids, \rho)] \\ \qquad N' \in applyTask(S, M, (pred, npred, \overline{\rho}), M' + [(ids, \rho)]) \end{cases}$$

The above expression states that if we pick a resource requiring task (ids, ρ) which satisfies the positive predicate (specified by $pred(ids, M+M'+[(ids, \rho)])$) and the tasks in its context (including the picked task itself) cannot satisfy the negative predicate ($\neg engage(S, M+M'+[(ids, \rho)], (pred, npred, \overline{\rho}))$) then we can grant the resources to this task and continue feeding the remaining tasks with the remaining resources. Otherwise, we leave this resource requiring task and proceed with the remaining tasks. In this expression, $min(\overline{0}, \overline{\rho}+\rho)$ and $max(\overline{0}, \rho+\overline{\rho})$ are point-wise minimum and maximum of $\overline{\rho}(r) + \rho(r)$ with 0, respectively. The predicate $pred(\widetilde{id}, M)$ means that there exists a mapping from variables in V_i (set of id variables) to the semantic identifiers in M (particularly mapping $\underline{\mathit{Id}}$ to a member of *ids*) which satisfies *pred*. The predicate *engage* is formally defined as follows:

$$engage(S, M, (pred, npred, \overline{\rho})) \doteq \exists_{M', t, S', \underset{\sim}{ids}, \rho, \widetilde{id}} S \overset{M', t}{\rightarrow} S' \wedge M \subseteq M' \wedge$$
$$(ids, \rho) \in M' \wedge \widetilde{id} \in ids \wedge npred(\widetilde{id}, M') \wedge \exists_{r \in R} \rho(r) > 0 \wedge \overline{\rho}(r) < 0$$

This predicate checks if S can perform a transition with a resource requiring multiset M' that firstly, contains M (thus, extending the same group of tasks), secondly, there exists a task identifier $\widetilde{id}$ in it that can satisfy the negative predicate ($npred(\widetilde{id}, M')$, defined in the same way as $pred(\widetilde{id}, \ldots)$) and the corresponding task can potentially use the resources offered by $\overline{\rho}$. To summarize, it checks for existence of a higher priority task in the context that can possibly consume the resources offered by the scheduler.

Note that application of a scheduler to a system does not necessarily satisfy all resource requirements of the system. Since the transition system of a scheduled system is itself a process specification transition system, several schedulers can be applied to a system in a distributed (using parallel composition of several schedulers) or hierarchical (using several levels of application operator) fashion in order to satisfy all its requirements.

Rules **(ER0)-(ER2)** represent preventing the system from requiring resources of a certain type by using an encapsulation operator on a given set of resources (similar to the encapsulation construct for actions).

Example 5 Consider the following process specification and the two different Earliest Deadline First (EDF) schedulers:

$$\begin{aligned}
Proc &= 1 : (\sigma_1(a, \{CPU \mapsto 1, Mem \mapsto 100\}(1))) \mid\mid \\
&\qquad 2 : (\sigma_2(b, \{CPU \mapsto 1, Mem \mapsto 100\}(2))) \\
EDF_1 &= \mu X.({|\!\rangle}_{x_t \in I\!R^{\geq 0}} \ (\underline{Id}.Dl = x_t)\{CPU \mapsto -2, Mem \mapsto -200\}(2)) \ ; X \\
EDF_2 &= \mu X.({|\!\rangle}_{x_t \in I\!R^{\geq 0}} \ ((\underline{Id}.Dl = x_t)\{CPU \mapsto -1, Mem \mapsto -100\}(2)) \mid\mid\mid \\
&\qquad {|\!\rangle}_{x_t \in I\!R^{\geq 0}}((\underline{Id}.Dl = x_t)\{CPU \mapsto -1, Mem \mapsto -100\}(2))) \ ; X
\end{aligned}$$

In the system $\partial_{\{CPU,Mem\}}(\langle\!\langle Proc \rangle\!\rangle_{EDF_1})$, the scheduler should start providing all available resources to task 1 for one unit of time, thus wasting one CPU and 100 units of memory. After that, available resources will be given to process 2. However, the process misses its deadline, since it needs 2 units of time to compute while its deadline has been shifted to 1 already. In contrast, system $\partial_{\{CPU,Mem\}}(\langle\!\langle Proc \rangle\!\rangle_{EDF_2})$ allows for a successful run. In this case, at the first time unit each of the two processes can receive a CPU and 100 units of memory. This is due to the fact that after providing the required resources of process 1 by one of the basic schedulers, the other scheduler may assign its resources to process 2. It follows from the semantics that after applying one resource offer to process 1 the whole process cannot engage in a resource interaction with a deadline of less than 2 and thus process 2 can receive its required resources.

This example helps us to realize that although scheduling policies such as earliest deadline first are assumed to be well-defined scheduling policies, formalizing their definition shows that different flavors of them may exist in practice (especially with respect to multiple resources), some of which may perform better than others for different systems.

5 Related Work

Several theories of process algebra with resources have been proposed recently. Our approach is mainly based on dense time ACSR of [3]. ACSR [13,12] is a process algebra enriched with priorities and specification of resources. Several extensions to ACSR have been proposed over time for which [13] provides a summary. The main shortcoming of this process algebra is the absence of an explicit scheduling concept. In ACSR, scheduling strategy is coded by means of priorities inside the process specification domain. Due to lack of a resource provision model, some other restrictions are also imposed on resource demands of processes. For example, two parallel processes are not allowed to call for one resource or they deadlock.

Our work has also been inspired by [4]. There, a process algebraic approach to resource modelling is presented and application of scheduling to process terms is investigated. This approach has an advantage over that of ACSR in that scheduling is separated from the process specification domain. However, firstly, there is no structure or guideline to define schedulers in this language (as [13] puts it, the approach looks like defining a new language semantics for each scheduling strategy) and secondly, the scheduling is restricted to a single resource (single CPU) concept.

Scheduling algebra of [17] defines a process algebra that has processes with interval timing. Computing the possible start time of tasks (so-called *anchor* *points*) is the only aspect of scheduling that is taken into account and it abstracts from resource requirements/provisions.

RTSL of [10] defines a discrete-time process algebra for scheduling analysis of single processor systems. The only shared resource in this process algebra is the single CPU. The restriction of tasks, in this approach, to sequential processes makes the language less expressive than ours (for example, in the process language a periodic task whose computation time may be larger than its period cannot be specified). Also, coding the scheduling policy in terms of a priority function may make specification of scheduling more cumbersome (similar to [4]).

Timed automata, as a well-known specification method for timed systems, has been extended to cover the notion of resources and scheduling as well (see [9], for example). Papers [16] and [11] are examples of an extension of untimed models with resources.

Asynchrony in timed parallel composition (interleaving of relative timed-transitions) has been of little interest in timed process algebras. Semantics of parallel composition in ATP [15] and different versions of timed-ACP [2], timed-CCS [6,7] and timed-CSP [8] all enforce synchronization of timed transitions such that both parallel components evolve concurrently in time. The *cIPA* of [1] is among the few timed process algebras that contain a notion of timed asynchrony. In this process algebra, non-synchronizing actions are forced to make asynchronous (interleaving) time transitions and synchronizing actions are specified to perform synchronous (concurrent) time transition. This distinction is not necessary in our framework, since non-synchronizing actions may find enough resources to execute in true concurrency and synchronizing actions may be forced to make interleaving time transitions due to the use of shared resources (e.g., scheduling two synchronizing actions on a single CPU).

6 Conclusion

In this paper, we propose an approach to integrate the separate specifications of real-time behavior (including aspects such as duration of actions, causal dependencies, synchronization) and scheduling strategy in an integrated and uniform process algebraic formalism. This allows for formalizing scheduling algorithms and benefiting from them in process algebraic design of systems as independent specification entities that influence the real-time behavior of the system. Our technical contribution to the current real-time and/or resource-based process algebraic formalisms can be summarized as defining a dense and asynchronous timed process algebra for resource consuming processes, providing a (similar) process algebraic language with basic constructs for defining resource providing processes (schedulers with multiple resources) and defining application of schedulers to processes in an algebraic fashion.

The theory presented in this paper can be completed/extended in several ways. Among those, axiomatizing *PARS* is one of the most important ones. As it can be seen in this paper, the three phases of specifications share a major part of the semantics; thus bringing the three levels of specification closer (for example, allowing for multi-level scheduling of a resource or allowing resource consuming schedulers) can be beneficial. Furthermore, applying the proposed theory in practice calls for simplification (e.g., to discrete time), optimization for implementation, tooling and experimenting in the future.

References

1. L. Aceto and D. Murphy. Timing and causality in process algebra. *Acta Informatica*, 33(4):317–350, 1996.
2. J. C. M. Baeten and C. A. Middelburg. *Process Algebra with Timing.* EATCS Monographs. Springer-Verlag, Berlin, Germany, 2002.
3. P. Brémond-Grégoire and I. Lee. A process algebra of communicating shared resources with dense time and priorities. *Theoretical Computer Science*, 189(1–2):179–219, 1997.
4. M. Buchholtz, J. Andersen, and H. H. Loevengreen. Towards a process algebra for shared processors. In *Proceedings of MTCS'01*, *Electronic Notes in Theoretical Computer Science*, 52(3), 2002.
5. G. C. Buttazzo. *Hard Real-Time Computing Systems.* Kluwer Academic Publishers, Boston, MA, USA, 1997.
6. F. Corradini, D. D'Ortenzio, and P. Inverardi. On the relationships among four timed process algebras. *Fundamenta Informaticae*, 38(4):377–395, 1999.
7. M. Daniels. Modelling real-time behavior with an interval time calculus. In J. Vytopil, editor, *Proceedings of FTRTF'91*, volume 571 of *Lecture Notes in Computer Science*, pages 53–71. Springer-Verlag, Berlin, Germany, 1991.
8. J. Davies and S. Schneider. A brief history of Timed CSP. *Theoretical Computer Science*, 138(2):243–271, Feb. 1995.
9. E. Fersman, P. Pettersson, and W. Yi. Timed automata with asynchronous processes: Schedulability and decidability. In *Proceedings of TACAS 2002*, volume 2280 of *Lecture Notes in Computer Science*, pages 67–82. Springer-Verlag, Berlin, Germany, 2002.
10. A.N. Fredette and R. Cleaveland. RTSL: A language for real-time schedulability analysis. In *Proceedings of the Real-Time Systems Symposium*, pages 274–283. IEEE Computer Society Press, Los Alamitos, CA, USA, 1993.
11. P. Gastin and M.W. Mislove. A truly concurrent semantics for a process algebra using resource pomsets. *Theoretical Computer Science*, 281:369–421, 2002.
12. I. Lee, J.-Y. Choi, H.H. Kwak, A. Philippou, and O. Sokolsky. A family of resource-bound real-time process algebras. In *Proceedings of FORTE'01*, pages 443–458. Kluwer Academic Publishers, Aug. 2001.
13. I. Lee, A. Philippou, and O. Sokolsky. A general resource framework for real-time systems. In *Proceedings of the Monterey Workshop, Venice, Italy*, 2002.
14. M. Mousavi, M. Reniers, T. Basten and M. Chaudron. *PARS*: A process algebra with resource and schedulers. Technical report, Department of Computer Science, Eindhoven University of Technology, 2003. To appear.
15. X. Nicollin and J. Sifakis. The algebra of timed processes ATP: theory and application. *Information and Computation*, 114(1):131–178, Oct. 1994.
16. M. Núñez and I. Rodríguez. PAMR: A process algebra for the management of resources in concurrent systems. In *Proceedings of FORTE'01*, pages 169–185. Kluwer Academic Publishers, 2001.
17. R. van Glabbeek and P. Rittgen. Scheduling algebra. In A. M. Haeberer, editor, *Proceedings of AMAST'99*, volume 1548 of *Lecture Notes in Computer Science*, pages 278–292. Springer-Verlag, Berlin, Germany, 1999.