A Calculus of Virtually Timed Ambients

Einar Broch Johnsen, Martin Steffen, and Johanna Beate Stumpf

University of Oslo, Oslo, Norway
{einarj,msteffen,johanbst}@ifi.uio.no

Abstract. A virtual machine, which is a software layer representing an execution environment, can be placed inside another virtual machine. As virtual machines at every level in a location hierarchy compete with other processes for processing time, the computing power of a virtual machine depends on its position in this hierarchy and may change if the virtual machine moves. These effects of nested virtualization motivate the calculus of virtually timed ambients, a formal model of hierarchical locations for execution with explicit resource provisioning, introduced in this paper. Resource provisioning in this model is based on virtual time slices as a local resource. To reason about timed behavior in this setting, weak timed bisimulation for virtually timed ambients is defined as an extension of bisimulation for mobile ambients. We show that the equivalence of contextual bisimulation and reduction barbed congruence is preserved by weak timed bisimulation. The calculus of virtually timed ambients is illustrated by examples.

1 Introduction

Virtualization technology enables the resources of an execution environment to be represented as a software layer, a so-called *virtual machine*. Application-level processes are agnostic to whether they run on such a virtual machine or directly on physical hardware. Since a virtual machine is a process, it can be executed on another virtual machine. Technologies such as VirtualBox, VMWare ESXi, Ravello HVX, Microsoft Hyper-V, and the open-source Xen hypervisor increasingly support running virtual machines inside each other in this way. This *nested virtualization*, originally introduced by Goldberg [12], is necessary to host virtual machines with operating systems which themselves support virtualization [5], such as Microsoft Windows 7 and Linux KVM. Nested virtualization has many uses, for example for end-user virtualization for guests, for development, and for deployment testing. Nested virtualization is also a crucial technology to support the hybrid cloud, as it enables virtual machines to migrate between different cloud providers [26].

To study the logical behavior of virtual machines in the context of nested virtualization, this paper develops a calculus of virtually timed ambients with explicit resource provisioning. Whereas previous work on process algebra with resources typically focusses on binary resources such as locks (e.g., [16, 20]) and previous work on process algebra with time typically focusses on time-outs

(e.g., [4,6,13,19,22]), time and resources in virtually timed ambients are *quantitative* notions: a process which gets *more resources* typically executes *faster*. Interpreting virtually timed ambients as locations for the deployment of processes, the resource requirements of processes executing at a location are matched by resources made available by the virtually timed ambient. The number of resources made available by the virtually timed ambient constitutes its computing power. This number is determined by the time slices it receives from its parent ambient. A virtually timed ambient that shares the time slices of its parent ambient with another process has less available time slices to execute its own processes. The model of resource provisioning in virtually timed ambients is inspired by Real-Time ABS [15], but extended to address nested virtualization in our calculus.

We call the corresponding time model for virtually timed ambients *virtual time*. Virtual time is provided to a virtually timed ambient by its parent ambient, similar to the time slices that an operating system provisions to its processes. When we consider many levels of nested virtualization, virtual time becomes a local notion of time which depends on a virtually timed ambient's position in the location hierarchy. Virtually timed ambients are mobile, reflecting that virtual machines may migrate between host virtual machines.

To formalize nested virtualization, notions of mobility and nesting are essential. The calculus of mobile ambients, originally developed by Cardelli and Gordon [8], captures processes executing at distributed locations in networks such as the Internet. Mobile ambients model both location mobility and nested locations, which makes this calculus well-suited as a starting point for our work. Combining these notions from the ambient calculus with the concepts of virtual time and resource provisioning, the calculus of virtually timed ambients can be seen as a model of nested virtualization, where different locations, barriers between locations, barrier crossing, and their relation to virtual time and resource provisioning are important, and where the number and position of virtually timed ambients available for processing tasks influences the overall processing time of a program. This allows the effects of, e.g., load balancing and scaling to be observed using weak timed bisimulation.

Contributions. To study the effects of nested virtualization, the main contributions of this paper can be summarized as follows:

- we define a calculus of *virtually timed ambients*, to the best of our knowledge the first process algebra capturing notions of virtual time and resource provisioning for nested virtualization;
- we define *weak timed bisimulation* for the calculus, and show that weak timed bisimulation is equivalent to reduction barbed congruence with time.

2 Virtually Timed Ambients

Mobile ambients [8] are located processes, arranged in a hierarchy which may change dynamically. Interpreting the location as a place of deployment, virtually timed ambients extend mobile ambients with notions of virtual time and

	n	name
	`tick`	virtual time slice
Global systems:		
$G ::=$	$\mathbf{0}$	inactive system
	$G \mid G$	parallel composition
	$n[\textsc{Source} \mid M]$	virtually timed root ambient with a source clock
Timed systems:		
$M, N ::=$	$\mathbf{0}$	inactive system
	$M \mid N$	parallel composition
	$(\nu n)M$	restriction
	$n[\textsc{Clock} \mid P]$	virtually timed ambient with a local clock
Timed processes:		
$P, Q ::=$	$\mathbf{0}$	inactive process
	$P \mid Q$	parallel composition
	$(\nu n)P$	restriction
	$!C.P$	replication
	$C.P$	prefixing
	$n[\textsc{Clock} \mid P]$	virtually timed ambient with a local clock
Timed capabilities:		
$C ::=$	**in** n	can enter n and adjust the local clock there
	out n	can exit n and adjust the local clock on the outside
	open n	can open n and adjust own local clock
	consume	consumes one resource

Table 1. Syntax of the virtually timed ambient calculus.

resource consumption. The timed behavior depends on the one hand on the *local* timed behavior, but on the other hand on the placement or deployment of the component in the hierarchical ambient structure. Virtually timed ambients use a notion of time which is *local* to each ambient, but at the same time *relative* to the computing power of the surrounding ambients.

Before considering the details of virtually timed ambients, we briefly recall the syntax and basic ideas of mobile ambients [8]. This syntax, and the semantics we consider, is based on [18] and largely unchanged compared to [8]. The main difference to [8] lies in a separation of processes into two levels: *processes* and *systems*. Systems characterize the outermost layer of an ambient structure. This distinction is used to simplify proofs of bisimulation in Sect. 3.

Mobile Ambients. Mobile ambients [8], originally introduced to represent "administrative domains" for processes, are defined as follows. The inactive process $\mathbf{0}$ does nothing. The parallel composition $P \mid Q$ allows both processes P and Q to proceed concurrently, where the binary operator $\mid$ is commutative and associative. The restriction operator $(\nu m)P$ creates a new and unique name with process P as its scope. In the calculus, the administrative domains for processes, called *ambients*, are represented by names. A process P located in an ambient named m is written $m[P]$. Ambients can be nested, and the nesting structure can change dynamically. A change of the nesting structure is specified by prefixing a process with a *capability*. There are three basic capabilities.

$P \rightarrowtriangle Q \Rightarrow (\nu n)P \rightarrowtriangle (\nu n)Q$	(R-Res)
$P \rightarrowtriangle Q \Rightarrow P \mid R \rightarrowtriangle Q \mid R$	(R-Par)
$P \rightarrowtriangle Q \Rightarrow n[P] \rightarrowtriangle n[Q]$	(R-Amb)
$P' \equiv P, P \rightarrowtriangle Q, Q \equiv Q' \Rightarrow P' \rightarrowtriangle Q'$	(R-Red $\equiv$)

Table 2. Reduction rules.

The input capability *in n* indicates the willingness of a process (respectively its containing ambient) to enter an ambient named n, running in parallel with its own ambient; e.g., $k[in\ n.P] \mid n[Q] \rightarrowtriangle n[k[P] \mid Q]$. The output capability *out n* enables an ambient to leave its surrounding (or parental) ambient n; e.g., $n[k[out\ n.P] \mid Q] \rightarrowtriangle k[P] \mid n[Q]$. The open capability *open n* allows an ambient named n at the same level as the capability to be opened; e.g., $k[open\ n.P \mid n[Q]] \rightarrowtriangle k[P \mid Q]$. The semantics is given as a reduction semantics which combines structural congruence with reduction rules.

Virtual Time and Local Clocks. Virtually timed ambients combine timed processes and timed capabilities with the features of the calculus for mobile ambients summarized above. In Table 1 we can see that in the calculus of virtually timed ambients every closed system of ambients must be contained in a root ambient with a *source clock* triggering the clocks of the local subambients recursively. Timed systems and processes are defined analogously to systems and processes in mobile ambients, with the difference that each virtually timed ambient contains a *local clock* and other virtually timed ambients or processes. The timed capabilities of virtually timed ambients extend the capabilities of mobile ambients with additional time management, explained below. In order to define computing power, a capacity **consume** for resource consumption, of processes is added.

Definition 1 (Virtually timed ambients). *Virtually timed ambients are defined by the syntax in Table 1.*

The semantics is given as a reduction semantics which combines structural congruence with reduction rules and can be found in Table 2 and Table 3. In Table 3 we make use of the notion of *observables*.

Definition 2 (Observables). *An* observable*, also known as a* barb*, is the presence of a top-level ambient whose name is not restricted. The observation predicate $P \downarrow n$ captures this observable. Thus, $P \downarrow n$ if $P \equiv (\nu \bar{m})(n[P_1] \mid P_2)$, where $n \notin \{\bar{m}\}$. We write $P \Downarrow n$ if there exists P' such that $P \Rightarrow P'$ and $P' \downarrow n$.*

To represent the outlined time model the local clock contained in each virtually timed ambient is responsible for triggering timed behavior and local resource consumption. Each time slice emitted by a local clock triggers the clock of one of its subambients in a round-robin way or is consumed by a process as a resource. This corresponds to a simple form of fair, *preemptive scheduling*, which makes the system's behavior sensitive to the number of co-located virtually timed ambients and resource consuming processes. Clocks have a *speed*, interpreted *relative*

$$\frac{\begin{array}{l}\textsc{Clock}_k = \textsc{Clock}\{c_k, (p_k, q_k), \{m,\ n\ \}, \emptyset\} \\ \textsc{Clock}_m = \textsc{Clock}\{c_m, (p_m, q_m), \{a_1, \ldots, a_k\}, \{a_{k+1}, \ldots a_n\}\} \\ \textsc{Clock}_n = \textsc{Clock}\{c_n, (p_n, q_n), \{b_1, \ldots, b_i\}, \{b_{i+1}, \ldots b_j\}\} \\ \textsc{Clock}_k^* = \textsc{Clock}\{c_k, (p_k, q_k), \{m\}, \emptyset\} \\ \textsc{Clock}_m^* = \textsc{Clock}\{c_m, (p_m, q_m), \{a_1, \ldots, a_k,\ n\ \}, \{a_{k+1}, \ldots a_n\}\} \\ \textsc{Clock}_n^* = \textsc{Clock}\{c_n, (p_n, q_n), \{b_1, \ldots, b_i\} \cup\ \overline{o}\ , \{b_{i+1}, \ldots b_j\}\}\end{array}}{\begin{array}{c} k[\textsc{Clock}_k \mid n[\textsc{Clock}_n \mid\ \textbf{in}\ m.P\ \mid Q] \mid m[\textsc{Clock}_m \mid R]] \\ \multimap k[\textsc{Clock}_k^* \mid m[\textsc{Clock}_m^* \mid R \mid n[\textsc{Clock}_n \mid P \mid Q]]]\end{array}}\ (\text{TR-In})$$

$$\frac{\begin{array}{l}\textsc{Clock}_k = \textsc{Clock}\{c_k, (p_k, q_k), \emptyset, \{m\}\} \\ \textsc{Clock}_m = \textsc{Clock}\{c_m, (p_m, q_m), \{a_1, a_2, \ldots, a_k,\ n\ \}, \{a_{k+1}, \ldots a_n\}\} \\ \textsc{Clock}_n = \textsc{Clock}\{c_n, (p_n, q_n), \{b_1, \ldots, b_i\}, \{b_{i+1}, \ldots b_j\}\} \\ \textsc{Clock}_k^* = \textsc{Clock}\{c_k, (p_k, q_k), \{\ n\ \}, \{m\}\} \\ \textsc{Clock}_m^* = \textsc{Clock}\{c_m, (p_m, q_m), \{a_1, a_2, \ldots, a_k\}, \{a_{k+1}, \ldots a_n\}\} \\ \textsc{Clock}_n^* = \textsc{Clock}\{c_n, (p_n, q_n), \{b_1, \ldots, b_i\} \cup\ \overline{o}\ , \{b_{i+1}, \ldots b_j\}\}\end{array}}{\begin{array}{c} k[\textsc{Clock}_k \mid m[\textsc{Clock}_m \mid n[\textsc{Clock}_n \mid\ \textbf{out}\ m.P\ \mid Q] \mid R]] \\ \multimap k[\textsc{Clock}_k^* \mid n[\textsc{Clock}_n \mid P \mid Q] \mid m[\textsc{Clock}_m^* \mid R]]\end{array}}\ (\text{TR-Out})$$

$$\frac{\begin{array}{l}\textsc{Clock}_m = \textsc{Clock}\{c_m, (p_m, q_m), \{\ n\ \}, \emptyset\} \\ \textsc{Clock}_m^* = \textsc{Clock}\{c_m, (p_m, q_m), \{\ a_1, a_2, \ldots, a_n\ \} \cup\ \overline{o}\ , \emptyset\}\end{array}}{m[\textsc{Clock}_m \mid\ \textbf{open}\ n.P\ \mid n[\textsc{Clock}_n \mid R] \mid Q] \multimap m[\textsc{Clock}_m^* \mid P \mid R \mid Q]}\ (\text{TR-Open})$$

$$\frac{\begin{array}{l}\textsc{Clock}_m = \textsc{Clock}\{c_m, (p_m, q_m), \{a_1, a_2, \ldots, a_n\}, \emptyset\},\ p_m > 0 \\ \textsc{Clock}_m^* = \textsc{Clock}\{c_m, (p_m, q_m), \{a_1, a_2, \ldots, a_n,\ \textbf{consume}.P\ \}, \emptyset\}\end{array}}{m[\textsc{Clock}_m \mid\ \textbf{consume}.P\ \mid R] \multimap m[\textsc{Clock}_m^* \mid R]}\ (\text{TR-Res})$$

$$\frac{\begin{array}{l}\textsc{Clock} = \textsc{Clock}\{\ c\ , (p, q), \{a_1, a_2, \ldots, a_k\}, \{a_{k+1}, \ldots a_n\}\}, c + 1 < q \\ \textsc{Clock}^* = \textsc{Clock}\{\ c + 1\ , (p, q), \{a_1, a_2, \ldots, a_k\}, \{a_{k+1}, \ldots a_n\}\}\end{array}}{m[\ \texttt{tick}\ \mid \textsc{Clock} \mid R] \multimap m[\textsc{Clock}^* \mid R]}\ (\text{TR-Tick}_1)$$

$$\frac{\textsc{Clock} = \textsc{Clock}\{c, (p, q), \{a_1, a_2, \ldots, a_k\}, \{a_{k+1}, \ldots a_n\}\}, c + 1 = q}{m[\ \texttt{tick}\ \mid \textsc{Clock} \mid R] \multimap m[\ \mathrm{RR}(\textsc{Clock} \mid R)\ ]}\ (\text{TR-Tick}_2)$$

$$\frac{\textsc{Source} = \textsc{Source}\{0, (n, 0), \{a_1, a_2, \ldots, a_k\}, \{a_{k+1}, \ldots a_n\}\}}{\textsc{Source} \mid R \multimap\ \mathrm{RR}(\textsc{Source} \mid R)}\ (\text{TR-Source})$$

Table 3. Timed reduction rules for timed capabilities, where a_k and b_i are time consuming virtually timed ambients and processes in R and Q, respectively.

to the speed of the surrounding virtually timed ambient. The speed of a clock is given by the pair (p, q), where p is the number of local time slices emitted for a number q of time slices received from the surrounding ambient, $p, q \in \mathbb{N}^0$. Thus, time in a nested ambient is relative to the global time, and depends on the speed of the clocks of the ambients in which it is contained and on its number of siblings. The speed of the source clocks is defined as $(n, 0), n \in \mathbb{N}^0$, as the sources do not need any input, while for the speed of a local clock it holds that an input of $q = 0$ is only valid if $p = 0$, too. As those ambients with speed $(0, 0)$ do not require any times slices from their parental ambient and do not show any timed behavior, they are not considered *time consuming*. However, processes which are prefixed with the resource consumption capability **consume**.P are considered time consuming. Note, that mobile ambients can be represented as virtually timed ambients with a clock with speed $(0, 0)$.

Definition 3 (Local clocks). *A local clock contains a* counter *to record the number of received time slices, its own* speed*, and two* sets*:*

$$\textsc{Clock}\{counter, (p, q), \{a_1, a_2, \ldots, a_k\}, \{a_{k+1}, \ldots a_n\}\} \ .$$

The first set contains the names of time consuming processes running in the ambient as well as time consuming virtually timed subambients in the surrounding ambient which have not yet received a time slice in the current cycle and the second set those which have.

When a clock receives a time slice, denoted `tick`, from its surrounding ambient, one of the following actions occurs: If $counter + 1 < q$, then the clock records this time slice and continues waiting (i.e., $\textsc{Clock}\{counter \coloneqq counter + 1, (p, q), \{a_1, a_2, \ldots, a_k\}, \{a_{k+1}, \ldots a_n\}\}$); if $counter + 1 = q$, then the input number is reached, the counter is set to 0 and the clock emits time slices to p subambients of the first set and puts them in the second set (i.e., $\textsc{Clock}\{counter \coloneqq 0, (p, q), \{a_{p+1} \ldots, a_k\}, \{a_{k+1}, \ldots a_n, a_1, a_2, \ldots, a_p\}\}$). As soon as the first of the two sets is empty, the first and second set are switched. Thus, no ambient receives a second time slice before every other subambient has received the first one. In the sequel, we omit the representation of the counter and the sets of subambients. For a better overview in the examples we denote the speed of the clocks as superscript $\textsc{Clock}^{p,q}$. If an ambient is not time consuming, i.e, it has a clock with speed $(0, 0)$, we do not mention the clock. For actions which do not require time we assume *maximal progress*, meaning that actions which can be executed immediately will not be delayed.

Timed Capabilities. The timed capabilities **in** n, **out** n, and **open** n enable virtually timed ambients to move in a timed system. When moving virtually timed ambients, we must consider that the clocks need to know about their current subambients, therefore their list of subambients need to be adjusted.

We now explain the reduction rules for virtually timed ambients, which are given in Table 2 and Table 3. Observe that if we would not adjust the clocks then a moving subambient would not receive time slices from its new parental clock.

RR: Round-based distribution function

input: $\textsc{Clock}\{counter, (p, q), \{a_1, a_2, \ldots, a_k\}, \{a_{k+1}, \ldots, a_n\}\} \mid R$
$S := \{a_1, a_2, \ldots, a_k\}$
$T := \{a_{k+1}, \ldots, a_n\}$
if $S = \emptyset$ **then**
return $\textsc{Clock}\{0, (p, q), \emptyset, \emptyset\}$
else
  while $p \geq |S|$ **do**
    for all $a_i \in S$ **do**
      if $a_i = \textbf{consume}.P$ **then** $S := S \setminus a_i$; $R := R \mid P$
        Let $P \equiv (\nu\widetilde{n})P'$, P' is ν-binder free, $\overline{o} = \{o \mid P' \downarrow o\}$:
        $S := S \cup \overline{o}$
      else $R := a_1 \mid \cdots \mid a_i[\texttt{tick} \mid \cdots] \mid \cdots \mid a_n$
      end if
    end for
    $p := p - |S|$; $S := S \cup T$; $T := \emptyset$
  end while
  Choose a subset $S' \subset S$ such that $|S'| = p$.
  for all $a_i \in S'$ **do**
    if $a_i = \textbf{consume}.P$ **then** $S' := S \setminus a_i$, $R := R \mid P$
      Let $P \equiv (\nu\widetilde{n})P'$, P' is ν-binder free, $\overline{o} = \{o \mid P' \downarrow o\}$:
      $S := S \cup \overline{o}$
    else $R := a_1 \mid \cdots \mid a_i[\texttt{tick} \mid \cdots] \mid \cdots \mid a_n$
    end if
  end for
  $S := S \setminus S'$; $T := T \cup S'$
end if
return $\textsc{Clock}\{0, (p, q), S, T\} \mid R$

In (TR-In) and (TR-Out), the clocks of the old and new parental ambient of the moving ambient have to be updated. Here we let $P \equiv (\nu\widetilde{n})P'$, where P' is ν-binder free, such that $\overline{o} = \{o \mid P' \downarrow o\}$. In (TR-Open) the clock of the opening ambient itself is updated. Note also that here the clock of the opened ambient is deleted. For virtually timed ambients with a clock with speed $(0, 0)$, the timed capabilities are equivalent to the capabilities for mobile ambients, as ambients, which are not time consuming, are not considered in the time management of the clocks. In (TR-Res) the time consuming process is moved into the clock, where it awaits the distribution of a time slice as resource before it can continue. This reduction can only happen in virtually timed ambients with $p > 0$, meaning ambients which actually emit resources. In (TR-Tick_1) the required number q_m of input time slices to trigger the local clock is not reached, thus the incoming time slice, denoted as `tick`, is only registered in the counter. In (TR-Tick_2) the local clock releases p time slices to its subambients and potentially to time consuming process. This is denoted with the function RR for round based distribution of time slices. The function takes as input the distributing Clock and distributes time slices `tick` to the subambients and processes in the given sets, thereby adjusting the sets of remaining and of served ambients. The source clocks Source can reduce without parental time slices as

given in (TR-Source). The following example shows the encoding of a system with a load balancer in virtually timed ambients.

Example 1. A system with a load balancer can be defined as follows:

load balancer system: $(\nu\ lb, a, b)\ lbs[\textsc{Clock}^{2,1} \mid lb \mid a \mid b]$

incoming request: $request[P.done\_signal \mid in\ lbs.enter\_signal.\mathbf{open}\ move]$

load balancer: $lb[!\mathbf{open}\ start.wait\_for\_enter.\mathbf{open}\ lock_a.$
$wait\_for\_enter.\mathbf{open}\ lock_b.start[] \mid !lock_a[x[]] \mid$
$!lock_b[y[]] \mid !(\mathbf{open}\ x.move[\mathbf{out}\ lb.\mathbf{in}\ request.\mathbf{in}\ a] \mid$
$\mathbf{open}\ y.move[\mathbf{out}\ lb.\mathbf{in}\ request.\mathbf{in}\ b])]$

ambient a: $a[\textsc{Clock}^{1,1} \mid !\mathbf{open}\ request.wait\_for\_done.done[\mathbf{out}\ a.\mathbf{out}\ lbs]]$

ambient b: $b[\textsc{Clock}^{1,1} \mid !\mathbf{open}\ request.wait\_for\_done.done[\mathbf{out}\ b.\mathbf{out}\ lbs]]$.

Here, the untimed load balancer creates a *move* ambient which moves incoming requests alternately into the virtually timed ambients a and b. For each time slice it receives from the source clock of the surrounding root ambient, the local clock of *lbs* distributes two time slices. Therefore, both subambients a and b receive one time slice. When a request has been executed, it releases an ambient *done* which emerges to the outside of the system and becomes observable.

Resource Consumption. Processes expend the processing power of the ambient they are contained in by consuming the local time slices as resources. Thus, time consuming processes and time consuming subambients in a virtually timed ambient compete for the same resource. The consumption of a computing resource is defined as the capability **consume**. An ambient with a higher local clock speed produces more time slices and therefore also more resources for each parental time slice, which in turn allows more work to be done for each parental time slice. We consider resource consumption by a request which was sent to the system of Example 1.

Example 2. Consider the virtually timed system with a load balancer from Example 1, with an incoming request.

$lbs[\cdots] \mid$
$request[\mathbf{consume}.\mathbf{consume}.done\_signal \mid \mathbf{in}\ lbs.enter\_signal.\mathbf{open}\ move]$

The request enters the system and is transferred by the load balancer into a, where it is opened during the reduction and awaits resource consumption. After one time signal of the source clock, the virtually timed ambient a emits one resource, which is consumed by the request:

$$a[\textsc{Clock}^{1,1} \mid !open\ request.wait\_for\_done.done[\mathbf{out}\ a.\mathbf{out}\ lbs] \mid wait\_for\_done.done[\mathbf{out}\ a.\mathbf{out}\ lbs] \mid \mathbf{consume}.done\_signal].$$

After another time signal from the source clock the ambient *done* can emerge:

$$a[\textsc{Clock}^{1,1} \mid !open\ request.wait\_for\_done.done[\mathbf{out}\ a.out\ lbs]] \mid done[\mathbf{out}\ lbs].$$

$(\nu\widetilde{m})(m[\textsc{Clock} \mid \textbf{in}\ n.P \mid Q] \mid M), m \in \widetilde{m}$ $\xrightarrow{*.\texttt{enter\_}n} (\nu\widetilde{m})(n[m[(\textsc{Clock} \mid P) \mid Q] \mid \circ\ ] \mid M)$	(Enter Shh)
$(\nu\widetilde{m})(k[\textsc{Clock} \mid \textbf{in}\ n.P \mid Q] \mid M), k \notin \widetilde{m}$ $\xrightarrow{k.\texttt{enter\_}n} (\nu\widetilde{m})(n[k[(\textsc{Clock} \mid P) \mid Q] \mid \circ\ ] \mid M)$	(Enter)
$(\nu\widetilde{m})(m[\textsc{Clock} \mid \textbf{out}\ n.P \mid Q] \mid M), m \in \widetilde{m}$ $\xrightarrow{*.\texttt{exit\_}n} (\nu\widetilde{m})(m[(\textsc{Clock} \mid P) \mid Q] \mid n[M \mid \circ\ ])$	(Exit Shh)
$(\nu\widetilde{m})(k[\textsc{Clock} \mid \textbf{out}\ n.P \mid Q] \mid M), k \notin \widetilde{m}$ $\xrightarrow{k.\texttt{exit\_}n} (\nu\widetilde{m})(k[(\textsc{Clock} \mid P) \mid Q] \mid n[M\ \mid \circ\ ])$	(Exit)
$(\nu\widetilde{m})(k[(\textsc{Clock} \mid P)] \mid M), k \notin \widetilde{m}$ $\xrightarrow{k.\texttt{enter\_}n} (\nu\widetilde{m})(k[\textsc{Clock}^* \mid n[\circ] \mid P] \mid M)$	(Co-Enter)
$(\nu\widetilde{m})(k[(\textsc{Clock} \mid P)] \mid M)$ $\xrightarrow{n.\texttt{open\_}k} n[\circ \mid (\nu\widetilde{m})(P \mid M)]$	(Open)
$(\nu\widetilde{m})(k[\textsc{Clock} \mid Q] \mid M), k \notin \widetilde{m}$ $\xrightarrow{k.\texttt{tick}} (\nu\widetilde{m})(k[\textsc{Clock} \mid \texttt{tick} \mid Q] \mid M)$	(Tick)

Table 4. Rules for timed labeled transition systems, where in (Co-Enter) given $\textsc{Clock} = \textsc{Clock}\{c_k, (p_k, q_k), \{a_1, \ldots, a_k\}, \{a_{k+1}, \ldots a_n\}\}$ the updated clock is denoted by $\textsc{Clock}^* = \textsc{Clock}\{c_k, (p_k, q_k), \{a_1, \ldots, a_k, n\}, \{a_{k+1}, \ldots a_n\}\}$ as seen in Table 3.

3 Comparing Virtually Timed Ambients

When comparing virtually timed ambients, e.g., in terms of bisimulation, we need to consider time as a factor. For this purpose, we define a labeled transition system which contains an observable action capturing global time steps.

Weak Bisimulation for Virtually Timed Ambients. The reduction semantics from Sect. 2 captures the behavior of *closed* or global systems. To define bisimulation, we first formalize an *open* version of the operational semantics, using a labelled transition relation. To express interaction with a surrounding *context* in the open setting, transitions will have *labels* which capture interaction with an environment.

Definition 4 (Labels). *Let the set of labels Lab, with typical element α, be given as follows:* $\alpha \in Lab ::= \tau \mid k.\texttt{enter\_}n \mid k.\texttt{exit\_}n \mid k.\overline{\texttt{enter}}\texttt{\_}n \mid n.\texttt{open\_}k \mid *.\texttt{exit\_}n \mid *.\texttt{enter\_}n \mid k.\texttt{tick}$*, where k and n represent names of ambients. The* internal label *is τ, the rest are called* observable *labels. We refer to labels of the forms $*.\texttt{exit\_}n$ and $*.\texttt{enter\_}n$ as* anonymous *and other labels as* non-anonymous*, and let the* untimed *labels exclude the tick labels.*

The definition of the labels is based on the formalization for mobile ambients in [18]. In the rules (Enter) and (Exit), an ambient k enters, respectively exits, from an ambient n provided by the environment. The rules (Enter Shh) and (Exit Shh) model the same behavior for ambients with private names. In the rule (Co-Enter), an ambient n provided by the environment enters

an ambient k of the process. In the rule (Open), the environment provides an ambient n in which the ambient k of the process is opened. In the rule (Tick), $M \xrightarrow{k.\mathtt{tick}} M'$ expresses that the top-level ambient k of the system M receives one time slice `tick` from the source clock. Note that the transition semantics contains the symbol $\circ$ as a placeholder variable for the body of the context ambient, containing an arbitrary process and an arbitrary local clock. The process $P := \circ$ must be instantiated in the bisimulation. The replacement of the placeholder by a process and local clock is written as $P \bullet (\textsc{Clock} \mid Q)$ and defined as expected. The reduction semantics of a process can be encoded in the labelled transition system, because a reduction step can be seen as an interaction with an empty context. We are interested in bisimulations that abstract from τ-actions and use the notion of *weak actions*; let $\Rightarrow$ denote the reflexive and transitive closure of $\xrightarrow{\tau}$, let $\overset{\alpha}{\Rightarrow}$ denote $\Rightarrow\xrightarrow{\alpha}\Rightarrow$, and let $\overset{\hat{\alpha}}{\Rightarrow}$ denote $\Rightarrow$ if $\alpha = \tau$ and $\overset{\alpha}{\Rightarrow}$ otherwise. An example of a system consuming one parental `tick` and performing the subsequent τ-actions is given below:

Example 3. We reconsider Example 2. After one time signal of the source clock, the subambient a emits one resource, which is consumed by the request:

$$a[P'] := a[\textsc{Clock}^{1,1} \mid !\mathbf{open}\ \mathit{request}.\mathit{wait\_for\_done}.\mathit{done}[\mathbf{out}\ a.\mathbf{out}\ \mathit{lbs}] \mid \mathit{wait\_for\_done}.\mathit{done}[\mathbf{out}\ a.\mathbf{out}\ \mathit{lbs}] \mid \mathbf{consume}.\mathit{done\_signal}].$$

After another time signal from the source clock and some internal τ steps the ambient *done* can emerge, thus here it holds that:

$$\mathit{lbs}[\textsc{Clock}^{2,1} \mid \mathit{lb} \mid a[P'] \mid b] \xrightarrow{\mathit{lbs}.\mathtt{tick}} \mathit{lbs}[\textsc{Clock}^{2,1} \mid \mathtt{tick} \mid \mathit{lb} \mid a[P'] \mid b] \Rightarrow \mathit{lbs}[\textsc{Clock}^{2,1} \mid \mathit{lb} \mid a \mid b] \mid \mathit{done}[].$$

Considering timed systems as defined in Table 1, we can now define weak timed bisimulation as follows:

Definition 5 (Weak timed bisimulation). *A symmetric relation $\mathcal{R}$ over timed systems is a* weak timed bisimulation *if $M\ \mathcal{R}\ N$ implies:*

- *if $M \xrightarrow{\alpha} M'$, $\alpha \in \{\tau, k.\mathtt{enter}\_n, k.\mathtt{exit}\_n, k.\overline{\mathtt{enter}}\_n, n.\mathtt{open}\_k, k.\mathtt{tick}\}$, then there is a system N' such that $N \overset{\hat{\alpha}}{\Rightarrow} N'$ and for all clocks* Clock *and processes P it holds that $M' \bullet (\textsc{Clock} \mid P)\ \mathcal{R}\ N' \bullet (\textsc{Clock} \mid P)$;*
- *if $M \xrightarrow{*.\mathtt{enter}\_n} M'$ then there is a system N' such that $N \mid n[\circ] \Rightarrow N'$ and for all clocks* Clock *and processes P it holds that $M' \bullet (\textsc{Clock} \mid P)\ \mathcal{R}\ N' \bullet (\textsc{Clock} \mid P)$;*
- *if $M \xrightarrow{*.\mathtt{exit}\_n} M'$ then there is a system N' such that $n[\circ \mid N] \Rightarrow N'$ and for all clocks* Clock *and processes P it holds that $M' \bullet (\textsc{Clock} \mid P)\ \mathcal{R}\ N' \bullet (\textsc{Clock} \mid P)$.*

Systems M and N are *weakly timed bisimilar*, written $M \approx^t N$, if $M\ \mathcal{R}\ N$ for some weak timed bisimulation $\mathcal{R}$. If two systems are weakly timed bisimilar in

a timed setting where we observe the ticking of the source clock, then it follows from the definition of weak timed bisimulation that they are weakly bisimilar in a setting where we do not observe the ticking of the clocks and instead interpret all `tick`-actions as τ-actions.

Lemma 1 (Consistency). *$M \approx^t N$ implies that M and N are weakly bisimilar, $M \approx N$.*

Note that for virtually timed ambients which are not time consuming, i.e. with a speed of $(0, 0)$, weak timed bisimulation and weak bisimulation coincide.

Example 4. We compare the behavior of the system *lbs* from Example 1 with a second system called lbs_2, which is defined as follows:

load balancer system: $(\nu\ lb, a)\ lbs_2[\textsc{Clock}^{1,1} \mid lb \mid a]$
incoming request: $request[P.done\_signal \mid \textbf{in}\ lbs_2.enter\_signal.\textbf{open}\ move]$
load balancer: $lb[\ !wait\_\ for\ \_enter.move[\textbf{out}\ lb.\textbf{in}\ request.\textbf{in}\ a]]$
ambient a: $a[\textsc{Clock}^{2,1} \mid !\textbf{open}\ request.wait\_\ for\ \_done.done[\textbf{out}\ a.\textbf{out}\ lbs_2]]$

In contrast to *lbs*, system lbs_2 only contains one virtually timed ambient, which receives all requests. If we do not observe time, the systems behave the same, $lbs \approx lbs_2$, as they both answer requests by emitting an observable *done*-signal. However, the systems are not weakly timed bisimilar, $lbs \not\approx^t lbs_2$.

Reduction Barbed Congruence. Honda and Yoshida's method [14] can be used to define weak reduction barbed congruence for mobile ambients [18]. This approach can be extended to virtually timed ambients.

Definition 6 (Reduction barbed congruence over timed systems). Reduction barbed congruence over timed systems $\simeq_s$ *is the largest symmetrical relation over timed systems which is preserved by all system contexts, reduction closed and barb preserving.*

Theorem 1. *Weak timed bisimilarity and reduction barbed congruence coincide over timed systems .*

We first introduce some concepts related to reduction barbed congruence over timed systems before approaching the proof.

Definition 7. *A* context *is a process with a hole. A* system context *is a context that transforms systems into systems. System contexts are generated by the following grammar: $\mathcal{C}[-] := - \mid (\mathcal{C}[-] \mid M) \mid (M \mid \mathcal{C}[-]) \mid (\nu n)\mathcal{C}[-] \mid n[\mathcal{C}[-] \mid P] \mid n[P \mid \mathcal{C}[-]]$, where M is an arbitrary system and P is an arbitrary process.*

Definition 8. *A relation $\mathcal{R}$ is* preserved by system contexts *if $M\ \mathcal{R}\ N$ implies $\mathcal{C}[M]\ \mathcal{R}\ \mathcal{C}[N]$ for all system contexts $\mathcal{C}[-]$.*

Theorem 2. *Weak timed bisimilarity is preserved by system contexts.*

Proof. We extend the proof of Merro and Zappa Nardelli [18] for the $k.\texttt{tick}$ action. As the $k.\texttt{tick}$ action can be seen as a special case of the $k.\overline{\texttt{enter\_n}}$ action, the same proof method can be used here.

Definition 9. *A relation $\mathcal{R}$ over processes is* barb preserving *if $P \;\mathcal{R}\; Q$ and $P \downarrow n$ implies $Q \Downarrow n$.*

Weak timed bisimilarity carries over the properties of being reduction closed and barb-preserving from weak bismilarity for mobile ambients [18]. Thus, weak timed bisimilarity is contained in reduction barbed congruence, $\approx^t \subseteq \simeq_s$. To show that the other direction holds as well, we need to define a system context that observes the action $k.\texttt{tick}$. Contexts to observe the other actions of the labeled transition system are defined in [18]. Again we can consider $k.\texttt{tick}$ as a special case of the $k.\overline{\texttt{enter\_n}}$ action and can therefore define the context equivalently:

$$\mathcal{C}_{\texttt{tick}}[-] = (\nu a, b)a[\textbf{in}\ n.\texttt{tick}[\textbf{out}\ a.b[\textbf{out}\ \texttt{tick}.\textbf{out}\ n.\mathit{done}[\textbf{out}\ b]]]] \mid - \ .$$

Note that that all top level ambients of the system which is entered in the context will receive the time via τ-actions. With this context, we can adjust Lemma 3.8 of Merro and Zappa Nardelli [18] to virtually timed ambients.

Lemma 2. *Let $\alpha \in \{k.\texttt{enter\_}n, k.\texttt{exit\_}n.k.\overline{\texttt{enter\_n}}, n.\texttt{open\_k}, n.\texttt{tick}\}$ and let M be a system. For all clocks* Clock *and processes P, if $M \xrightarrow{\alpha} M'$ then $\mathcal{C}_\alpha[M] \bullet (\textsc{Clock} \mid P) \Rightarrow \simeq_s (M' \bullet (\textsc{Clock} \mid P)) \mid \mathit{done}[]$.*

Proof. We will only consider $\alpha = n.\texttt{tick}$. All other cases are detailed in [18]. Let P be a process. We know that $M \xrightarrow{n.\texttt{tick}} M'$. Then $M' \equiv n[\texttt{tick} \mid Q] \mid M''$.

$$\begin{aligned}
&\mathcal{C}_{\texttt{tick}}[M] \bullet (\textsc{Clock} \mid P)\\
\equiv&((\nu a, b)a[\textbf{in}\ n.\texttt{tick}[\textbf{out}\ a.b[\textbf{out}\ \texttt{tick}.\textbf{out}\ n.\mathit{done}[\textbf{out}\ b]])]] \mid M)\\
&\bullet (\textsc{Clock} \mid P)\\
\rightarrowtriangle\ &((\nu a, b)n[a[\texttt{tick}[\textbf{out}\ a.b[\textbf{out}\ \texttt{tick}.\textbf{out}\ n.\mathit{done}[\textbf{out}\ b]]]] \mid Q] \mid M'')\\
&\bullet (\textsc{Clock} \mid P)\\
\rightarrowtriangle\ &((\nu a, b)n[a[] \mid \texttt{tick}[b[\textbf{out}\ \texttt{tick}.\textbf{out}\ n.\mathit{done}[\textbf{out}\ b]]] \mid Q] \mid M'') \bullet (\textsc{Clock} \mid P)\\
\rightarrowtriangle\ &((\nu a, b)n[a[] \mid \texttt{tick} \mid Q] \mid b[\mathit{done}[\textbf{out}\ b]] \mid M'') \bullet (\textsc{Clock} \mid P)\\
\rightarrowtriangle\ &((\nu a, b)n[a[] \mid \texttt{tick} \mid Q] \mid b[] \mid \mathit{done}[] \mid M'') \bullet (\textsc{Clock} \mid P)\\
\equiv&(M' \bullet (\textsc{Clock} \mid P)) \mid \mathit{done}[].
\end{aligned}$$

To prove the correspondence between actions α and their contexts $\mathcal{C}_\alpha[-]$, we have to prove the converse of the lemma above as well. The proof of this result uses particular contexts $\texttt{spy}_\alpha\langle i, j, -\rangle$ as a technical tool to guarantee that the process P provided by the environment does not perform any action. We define the context $\texttt{spy}_{\texttt{tick}}\langle i, j, -\rangle := (i[] \mid -) \oplus (j[] \mid -)$, where i, j are fresh ambient names and $\oplus$ is an encoding of internal choice in the ambient calculus. We can now adjust the proof of Lemma 3.12 in [18], making use of Lemmas 3.9, 3.10 and 3.11 in [18].

Lemma 3. *Let* $\alpha \in \{k.\texttt{enter\_}n, k.\texttt{exit\_}n.k.\overline{\texttt{enter\_n}}, n.\texttt{open\_k}, n.\texttt{tick}\}$ *and let* M *be a system. Let* i, j *be fresh names for* M*. For all processes* P *with* $\{i,j\} \cap fn(P) = \emptyset$*, if* $C_\alpha[M] \bullet (\textsc{Clock} \mid \textit{spy}_\alpha\langle i,j,P\rangle) \Rightarrow\equiv N \mid \textit{done}[]$ *and* $N \Downarrow_{i,j}$ *there exists a system* M' *such that* $M \stackrel{\alpha}{\Rightarrow} M'$ *and* $M' \bullet (\textsc{Clock} \mid \textit{spy}_\alpha\langle i,j,P\rangle) \simeq_s N$*.*

Proof. We consider the case of $\alpha = n.\texttt{tick}$, all other cases are detailed in [18].

$$
\begin{aligned}
&\mathcal{C}_\alpha[M] \bullet (\textsc{Clock} \mid \texttt{spy}_\alpha\langle i,j,P\rangle)\\
&\equiv (\nu a,b)a[\textbf{in}\ n.\texttt{tick}[\textbf{out}\ a.b[\textbf{out}\ \texttt{tick}.\textbf{out}\ n.\textit{done}[\textbf{out}\ b]]]] \mid M\\
&\quad \bullet (\textsc{Clock} \mid \texttt{spy}_\alpha\langle i,j,P\rangle)\\
&\rightarrowtriangle (\nu a,b)n[a[] \mid \texttt{tick} \mid Q] \mid b[] \mid M''\\
&\quad \bullet (\textsc{Clock} \mid \texttt{spy}_\alpha\langle i,j,P\rangle) \mid \textit{done}[] \bullet (\textsc{Clock} \mid \texttt{spy}_\alpha\langle i,j,P\rangle)\\
&\equiv M' \bullet (\textsc{Clock} \mid \texttt{spy}_\alpha\langle i,j,P\rangle) \mid \textit{done}[] \bullet (\textsc{Clock} \mid \texttt{spy}_\alpha\langle i,j,P\rangle)\\
&\equiv N \mid \textit{done}[]\ .
\end{aligned}
$$

We conclude that $M \stackrel{n.\texttt{tick}}{\Longrightarrow} M'$. It holds that $M' \bullet (\textsc{Clock} \mid \texttt{spy}_\alpha\langle i,j,P\rangle) \simeq_s N$.

It now follows using Theorem 3.14 in [18] that reduction barbed congruence is contained in weak timed bisimilarity, i.e. $M \approx^t N$ iff $M \simeq_s N$.

4 Related Work

As our work is building on the ambient calculus, we focus on timed process algebras based on the closely related π-calculus [23], which originated from CCS. Related work building on ACP and CSP can be found in, e.g., [3, 4, 22].

A special action for time was introduced in an early timed extension [19] of CCS, without committing to a discrete or continuous time domain. A related idling action that needs exactly one time unit to be processed is proposed in [13], where time is discrete and processes synchronized via a global clock. The notion of local time proposed for CCS in [24] is closer to our local clocks, but uses a time-out oriented model. Timers, which are introduced to express the possibility of a time-out and are controlled by a global clock, have been studied for mobile ambients [1,2,10]. Modeling time-out is a straightforward extension of our work. However, the high-level idea of these works is very different from ours: they all focus on speed as the *duration* of processes, while in our approach with local clocks speed describes the *processing power* of a virtually timed ambient.

Cardelli and Gordon defined a labeled transition system for mobile ambients [9], but no bisimulation. A bisimulation relation for a restricted version of mobile ambients, called mobile safe ambients, is defined in [17] and provides the basis for later work. Barbed congruence for the same fragment of mobile ambients is defined in [25]. It is shown in [11] that name matching reduction barbed congruence and bisimulation coincide in the π-calculus. A bisimulation relation with contextual labels for the ambient calculus is defined in [21], but this approach is not suitable for providing a simple proof method. A labelled bisimulation for mobile ambients is defined by Merro and Nardelli [18], who prove

that this bisimulation is equivalent to reduction barbed congruence and develop up-to-proof techniques. The weak timed bisimulation defined in this paper is a conservative extension of this approach.

A process algebra with resources as primitives is studied in [16], in which priorities are used to make processes sensitive to scheduling. A similar approach with explicit scheduling is studied in [20]. In contrast to these works, scheduling in our approach is determined by the implementation of the resource distribution. A core language for defining cloud services and their deployment on cloud platforms is introduced in [7] to enable statically safe service composition and custom implementations of cloud services. However, in contrast to our approach, time and performance are not taken into consideration.

5 Concluding Remarks

We believe that virtualization opens for new and interesting foundational models of computation by explicitly emphasizing deployment and resource allocation. This paper introduces virtually timed ambients, a formal model of hierarchical locations of execution with explicit resource provisioning. In the proposed model resource provisioning is based on virtual time, a local notion of time reminiscent of time slices for virtual machines in the context of nested virtualization. This way, the computing power of an ambient depends on its location in the deployment hierarchy. To reason about timed behavior in this setting, we define weak timed bisimulation for virtually timed ambients as a conservative extension of bisimulation for mobile ambients, and show that the equivalence of bisimulation to reduction barbed congruence is preserved by this extension.

The calculus of virtually timed ambients opens up opportunities for further interesting research questions. One line of research is in statically controlling resource management, for example by means of behavioral types. Another line of research is in dynamically controlling resource management by means of resource awareness. This line of work is suggested by examples in this paper such as load balancers but could be enhanced by reflective resource capabilities allowing a process to influence its own deployment similar to virtualization APIs found in the context of cloud computing.

References

1. B. Aman and G. Ciobanu. Mobile ambients with timers and types. In *Proc. 4th International Colloquium on Theoretical Aspects of Computing (ICTAC'07)*, *LNCS* 4711, pages 50–63. Springer, 2007.
2. B. Aman and G. Ciobanu. Timers and proximities for mobile ambients. In *Proc. 2nd International Symposium on Computer Science in Russia (CSR'07)*, *LNCS* 4649, pages 33–43. Springer, 2007.
3. J. C. M. Baeten and J. A. Bergstra. Real Time Process Algebra. Technical Report CS-R 9053, Centrum voor Wiskunde en Informatica (CWI), 1990.
4. J. C. M. Baeten and C. A. Middelburg. *Process Algebra with Timing*. Monographs in Computer Science. An EATSC series. Springer, 2002.

5. M. Ben-Yehuda, M. D. Day, Z. Dubitzky, M. Factor, N. Har'El, A. Gordon, A. Liguori, O. Wasserman, and B. Yassour. The Turtles project: Design and implementation of nested virtualization. In *Proc. 9th Symposium on Operating Systems Design and Implementation (OSDI 2010)*, pages 423–436. USENIX, 2010.
6. M. Berger. *Towards Abstractions for Distributed Systems*. PhD thesis, University of London, Imperial College, Nov. 2004.
7. O. Bračevac, S. Erdweg, G. Salvaneschi, and M. Mezini. CPL: A core language for cloud computing. In *Proc. 15th Intl. Conf. on Modularity*, pages 94–105. ACM, 2016.
8. L. Cardelli and A. D. Gordon. Mobile ambients. *Theoretical Comput. Sci.*, 240(1):177–213, 2000.
9. L. Cardelli and A. D. Gordon. Equational properties of mobile ambients. *Mathematical Structures in Computer Science*, 13(3):371–408, 2003.
10. G. Ciobanu. Interaction in time and space. *ENTSC*, 203(3):5–18, May 2008.
11. C. Fournet and G. Gonthier. A hierarchy of equivalences for asynchronous calculi. *Journal of Logic and Algebraic Programming*, 63(1):131 – 173, 2005.
12. R. P. Goldberg. Survey of virtual machine research. *IEEE Computer*, 7(6):34–45, 1974.
13. M. Hennessy and T. Regan. A process algebra for timed systems. *Information and Computation*, 117(2):221–239, 1995.
14. K. Honda and N. Yoshida. On reduction-based process semantics. *Theoretical Computer Science*, 151(2):437–486, 1995.
15. E. B. Johnsen, R. Schlatte, and S. L. Tapia Tarifa. Integrating deployment architectures and resource consumption in timed object-oriented models. *Journal of Logical and Algebraic Methods in Programming*, 84(1):67–91, 2015.
16. I. Lee, A. Philippou, and O. Sokolsky. Resources in process algebra. *Journal of Logic and Algebraic Programming*, 72(1):98 –122, 2007.
17. M. Merro and M. Hennessy. A bisimulation-based semantic theory of safe ambients. *ACM Transactions on Programming Languages and Systems*, 28(2):290–330, 2006.
18. M. Merro and F. Zappa Nardelli. Behavioral theory for mobile ambients. *Journal of the ACM*, 52(6):961–1023, Nov. 2005.
19. F. Moller and C. Tofts. A temporal calculus of communicating systems. In *Proc. CONCUR, LNCS* 458, pages 401–415. Springer, 1990.
20. M. R. Mousavi, M. A. Reniers, T. Basten, and M. R. V. Chaudron. PARS: A process algebraic approach to resources and schedulers. In *Process Algebra for Parallel and Distributed Processing*. Chapman and Hall/CRC, 2008.
21. M. Murakami. Congruent bisimulation equivalence of ambient calculus based on contextual transition system. In *Proc. 7th Intl. Symp. on Theoretical Aspects of Software Engineering (TASE 2013)*, pages 149–152. IEEE Computer Society, 2013.
22. X. Nicollin and J. Sifakis. The algebra of timed processes, ATP: Theory and application. *Information and Computation*, 114(1):131–178, 1994.
23. D. Sangiorgi and D. Walker. *The Pi-Calculus: A Theory of Mobile Processes*. Cambridge University Press, 2001.
24. I. Satoh and M. Tokoro. A timed calculus for distributed objects with clocks. In *Proc. 7th European Conference on Object-Oriented Programming (ECOOP'93)*, pages 326–345. Springer, 1993.
25. M. Vigliotti and I. Phillips. Barbs and congruences for safe mobile ambients. *ENTCS*, 66(3):37 – 51, 2007.
26. D. Williams, H. Jamjoom, and H. Weatherspoon. The Xen-Blanket: Virtualize once, run everywhere. In *Proc. 7th European Conference on Computer Systems (EuroSys'12)*, pages 113–126. ACM, 2012.