Resource-Aware Virtually Timed Ambients

Einar Broch Johnsen, Martin Steffen, Johanna Beate Stumpf, Lars Tveito

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

Abstract. Virtually timed ambients is a calculus of nested virtualization, which models timing and resource consumption for hierarchically structured virtual machines. This structure may change dynamically to support load-balancing, migration, and scaling. This paper introduces resource-awareness for virtually timed ambients, which enables processes to actively query the system about the resources necessary for a task and to reconfigure accordingly. Technically we extend virtually timed ambients with context-expressions using modal logic operators, give a formal semantics for the extension, and define bisimulation for resource-aware virtually timed systems. The paper also provides a proof of concept implementation in Maude and a case study involving dynamic auto scaling.

1 Introduction

In cloud-computing, *horizontal scaling* describes scaling by adding more machines into the given pool of resources. Cloud-service providers offer different kinds of scaling policies that allow their clients to monitor applications and automatically adjust capacity to maintain steady performance at low costs. For example, Amazon EC2 Auto Scaling [1] allows to dynamically and automatically scale the virtual capacity up or down according to conditions defined by the client. This paper provides a formalization to support dynamic auto scaling via resource-awareness for virtually timed ambients.

The calculus of *virtually timed ambients* [11] is a calculus of explicit resource provisioning, based on mobile ambients [3], and has been used to model nested virtualization in cloud systems. Virtualization technology enables the resources of an execution environment to be represented as a software layer, a so-called virtual machine. *Dynamic nested virtualization*, first introduced in [7], is a crucial technology to support cloud systems, as it enables virtual machines to migrate between different cloud providers [22]. It is also necessary to host virtual machines with operating systems which themselves support virtualization [2], such as Microsoft Windows 7 and Linux KVM. The time model used to realize the resource provisioning for virtually timed ambients is called *virtual time*. The time of a virtually timed ambient proceeds in the context of its parental ambient and is relative to the parent's time progression, With nested levels of virtualization, virtual time becomes a *local* notion of time which depends on an ambient's position in the nesting structure. Virtually timed ambients are mobile, reflecting that virtual machines may migrate between host virtual machines. Observe that such migration affects the execution speed of processes in the migrating virtually timed ambient, as well as in the virtually timed ambient which is left, and in the virtually timed ambient which is entered.

Resource-awareness allows processes or programs to know about available resources and about resources necessary for a task, and react accordingly. For virtually timed ambients, resource awareness enables, e.g.,horizontal scaling, by adding more virtual machines to a server in the cloud. The notion of resource-aware virtually timed ambients is based on context-aware ambients (CCA) [21], which introduce context-guarded processes to enable context-awareness of mobile ambients. We enhance the given context expressions to cover the notions of timing and resources of virtually timed ambients and extend the theory of resource-aware virtually timed ambients by contextual bisimulation. We further provide a case study for modeling dynamic auto scaling on the cloud. Thus, we define a calculus to model explicit resource management in cloud computing.

Contributions. The main contributions of this paper are the following:

- we define and discuss a calculus of *resource-aware virtually timed ambients*;
- we define *weak timed context bisimulation* for resource-aware virtually timed ambients;
- we show the feasibility of virtually timed ambients as a modelling language for cloud computing with a *case study* of dynamic auto scaling on Amazon EC2 modelled in a prototype implementation of our calculus in the Maude rewriting system;
- all concepts are illustrated by *examples.*

To the best of our knowledge, this is the first implementation of resource awareness for mobile ambients in rewriting logic.

Paper overview. We introduce resource-aware virtually timed ambients in Section 3. Section 4 discusses the implementation and contains the case study, exemplifying dynamic auto scaling on the cloud. We discuss related work and conclude in Sections 5 and 6.

2 Virtually Timed Ambients

Virtually timed ambients [10,11] is a calculus of explicit resource provisioning, based on mobile ambients. Mobile ambients [3] are processes with a concept of location, arranged in a hierarchy which may change dynamically. *Virtually timed ambients* interpret these locations as places of deployment and extend mobile ambients with notions of virtual time and resource consumption. The timed behavior depends on the one hand on the *local* timed behavior, and on the other hand on the placement or deployment of the virtually timed ambient or the process in the hierarchical ambient structure. Virtually timed ambients combine timed processes and timed capabilities with the features of mobile ambients.

	n	name
	`tick`	virtual time slice
Timed processes:		
$P, Q ::=$	$\mathbf{0}$	inactive process
$\mid$	$P \mid Q$	parallel composition
$\mid$	$(\nu n)\, P$	restriction
$\mid$	$!C.P$	replication
$\mid$	$C.P$	prefixing
$\mid$	$n[\textsc{Sched} \mid \texttt{tick}^x \mid P]$	virtually timed ambient
Timed capabilities:		
$C ::=$	$\mathbf{in}\ n$	enter n and adjust the local scheduler there
$\mid$	$\mathbf{out}\ n$	exit n and adjust the local scheduler on the outside
$\mid$	$\mathbf{open}\ n$	open n and adjust own local scheduler
$\mid$	$\mathbf{c}$	consume a resource

Table 1. Syntax of virtually timed ambients, $x \in \mathbb{N}_0$.

Definition 1 (Virtually timed ambients). *The syntax of virtually timed ambients is given by the grammar in Table 1.*

Timed processes differ from mobile ambients in that each virtually timed ambient contains, besides possibly further (virtually timed) subambients, a *local scheduler*. In the sequel, we omit the qualification "timed" or "virtually timed", when speaking about processes, capabilities, or ambients when the context of virtually timed ambients is clear. In the calculus, *virtually timed ambients* are represented by names and time slices are written as `tick`. 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. Replication of processes is given as $!C.P$. A process P located in an virtually timed ambient named n is written $n[\textsc{Sched} \mid \texttt{tick}^x \mid P]$, where $\texttt{tick}^0 \equiv \mathbf{0}$. Ambients can be nested, and the nesting structure can change dynamically, this is specified by prefixing a process with a *capability* $C.P$. *Timed capabilities* extend the capabilities of mobile ambients by including a *resource consumption* capability $\mathbf{c}$ and by giving the *opening*, *exiting*, and *entering* capabilities of ambients a timed interpretation. These capabilities restructure the hierarchy of an ambient system, so the behavior of local schedulers and resource consumption changes, as these depend on the placement of the timed ambient in the hierarchy.

In a virtually timed ambient, the local scheduler triggers timed behavior and local resource consumption. Each time slice emitted by a local scheduler triggers the scheduler of a subambient or is consumed by a process as a resource in a preemptive, yet *fair* way, which makes system behavior sensitive to co-located virtually timed ambients and resource consuming processes.

Definition 2 (Local and root schedulers). *Let the sets unserved and served contain the names of virtually timed ambients as well as processes (these are represented directly, lacking names). A* local scheduler *is denoted by*

$$\textsc{Sched}_{speed}\{in, out, rest, unserved, served\},$$

where speed $\in \mathbb{Q}$ *relates externally received to internally emitted time slices; in* $\in \mathbb{N}$ *records the number of received time slices; out* $\in \mathbb{N}$ *records the numbers of time slices than can be distributed for each incoming time slice, while rest* $\in \mathbb{N}$ *records additional distributable time slices depending on the speed; and unserved contains local ambients with a positive speed and processes which are intended to receive one time slice in this round of the scheduling, while served contains processes scheduled for the next round.*

Root schedulers*, represented as* $\textsc{Sched}^{\dagger}\{in, out, 0, unserved, served\}$*, are local schedulers which do not need an input to distribute time slices and therefore have no defined speed.*

The semantics of virtually timed ambients is given as a reduction system, similar to the semantics of mobile ambients. The rules for structural congruence $P \equiv Q$ are equivalent to those for mobile ambients (and therefore omitted here). The *reduction* relation $P \rightarrowtriangle Q$ for virtually timed ambients makes use of *observables,* also known as *barbs.* Barbs, originally introduced for the π-calculus [16], capture a notion of immediate observability. In the ambient calculus, these observations concern the presence of a top-level ambient whose name is not restricted. Let $\widetilde{m}$ describe a tuple of names, then the observability predicate $\downarrow_n$ or "barb" is defined as follows:

Definition 3 (Barbs, from [14]). *Process P strongly barbs on n, written* $P{\downarrow_n}$*, if* $P \equiv (\nu\widetilde{m})(n[P_1] \mid P_2)$*, where* $n \notin \{\widetilde{m}\}$*.*

A process that does not contain ν-binders is said to be *ν-binder free.* By moving the ν-binders to the outside and only considering the inside of their scope, we can observe the bound ambients inside the scope of the ν-binders.

Definition 4 (Timed top-level ambients). *For a process P, let* $P_{\downarrow}$ *denote the sets of all* timed top-level ambients*:* $P_{\downarrow} = \{n \mid P \equiv (\nu\widetilde{m})P' \wedge P'$ *is ν-binder free* $\wedge\, P'{\downarrow_n} \wedge speed_n > 0\}$*.*

Timed capabilities. The reduction rules for virtually timed ambients are given in Tables 2 and 3. The timed capabilities **in** n, **out** n, and **open** n enable virtually timed ambients to move in the hierarchical ambient structure. The local schedulers need to know about the current subambients, so their lists of subambients need to be adjusted when virtually timed ambients move. Observe that without adjusting the schedulers, the moving subambient would not receive time slices from the scheduler in its new surrounding ambient. In TR-In and TR-Out, the schedulers of the old and new surrounding ambient of the moving ambient are updated by removing and adding, respectively, the name of the moving ambient, if it has a speed greater zero. The scheduler of the moving

$$\frac{\begin{array}{l}\textsc{Sdl}_k = \textsc{Sched}_{speed_k}\{in_k, out_k, rest_k, U_k, S_k\}, \quad n \in U_k \cup S_k \\ \textsc{Sdl}_m = \textsc{Sched}_{speed_m}\{in_m, out_m, rest_m, U_m, S_m\} \\ \textsc{Sdl}_n = \textsc{Sched}_{speed_n}\{in_n, out_n, rest_n, U_n, S_n\} \\ \textsc{Sdl}'_k = \textsc{Sched}_{speed_k}\{in_k, out_k, rest_k, U_k \setminus \{n\}, S_k \setminus \{n\}\} \\ \textsc{Sdl}'_m = \textsc{Sched}_{speed_m}\{in_m, out_m, rest_m, U_m, S_m \cup \{n\}\}, \text{ if } speed_n > 0 \text{ else } \textsc{Sdl}_m \\ \textsc{Sdl}'_n = \textsc{Sched}_{speed_n}\{in_n, out_n, rest_n, U_n, S_n \cup P_{\downarrow}\}\end{array}}{\begin{array}{c}k[\textsc{Sdl}_k \mid n[\textsc{Sdl}_n \mid \mathbf{in}\ m.P \mid Q] \mid m[\textsc{Sdl}_m \mid R] \mid U] \\ \rightarrowtriangle k[\textsc{Sdl}'_k \mid m[\textsc{Sdl}'_m \mid R \mid n[\textsc{Sdl}'_n \mid P \mid Q]] \mid U]\end{array}} \quad (\text{TR-In})$$

$$\frac{\begin{array}{l}\textsc{Sdl}_k = \textsc{Sched}_{speed_k}\{in_k, out_k, rest_k, U_k, S_k\}, \quad n \in U_m \cup S_m \\ \textsc{Sdl}_m = \textsc{Sched}_{speed_m}\{in_m, out_m, rest_m, U_m, S_m\} \\ \textsc{Sdl}_n = \textsc{Sched}_{speed_n}\{in_n, out_n, rest_n, U_n, S_n\} \\ \textsc{Sdl}'_k = \textsc{Sched}_{speed_k}\{in_k, out_k, rest_k, U_k, S_k \cup \{n\}\}, \text{ if } speed_n > 0 \text{ else } \textsc{Sdl}_k \\ \textsc{Sdl}'_m = \textsc{Sched}_{speed_m}\{in_m, out_m, rest_m, U_m \setminus \{n\}, S_m \setminus \{n\}\} \\ \textsc{Sdl}'_n = \textsc{Sched}_{speed_n}\{in_n, out_n, rest_n, U_n, S_n \cup P_{\downarrow}\}\end{array}}{\begin{array}{c}k[\textsc{Sdl}_k \mid m[\textsc{Sdl}_m \mid n[\textsc{Sdl}_n \mid \mathbf{out}\ m.P \mid Q] \mid R] \mid U] \\ \rightarrowtriangle k[\textsc{Sdl}'_k \mid n[\textsc{Sdl}'_n \mid P \mid Q] \mid m[\textsc{Sdl}'_m \mid R] \mid U]\end{array}} \quad (\text{TR-Out})$$

$$\frac{\begin{array}{l}\textsc{Sdl}_k = \textsc{Sched}_{speed_k}\{in_k, out_k, rest_k, U_k, S_k\}, \quad n \in U_k \cup S_k \\ \textsc{Sdl}'_k = \textsc{Sched}_{speed_k}\{in_k, out_k, rest_k, U_k \setminus \{n\}, S_k \setminus \{n\} \cup P_{\downarrow} \cup R\downarrow\}\end{array}}{k[\textsc{Sdl}_k \mid \mathbf{open}\ n.P \mid n[\textsc{Sdl}_n \mid R] \mid Q] \rightarrowtriangle k[\textsc{Sdl}'_k \mid P \mid R \mid Q]} \quad (\text{TR-}\textsc{Open})$$

$$\frac{\begin{array}{l}\textsc{Sdl}_m = \textsc{Sched}_{speed_k}\{in_m, out_m, rest_m, U_m, S_m\}, \quad speed_m > 0 \\ \textsc{Sdl}'_m = \textsc{Sched}_{speed_m}\{in_m, out_m, rest_m, U_m, S_m \cup \{\mathbf{c}\ .P\}\}\end{array}}{m[\textsc{Sdl}_m \mid \mathbf{c}\ .P \mid R] \rightarrowtriangle m[\textsc{Sdl}'_m \mid R]} \quad (\text{TR-}\textsc{Resource})$$

Table 2. Timed reduction rules for timed capabilities. Here, a blue backdrop marks the trigger of the reduction, red the changes in the schedulers, and green eventual constraints.

subambient is also updated as it needs to contain the barbs of the process that was hidden behind the movement capability. In TR-Open, the scheduler of the opening ambient itself is updated by removing the name of the opened ambient and adding the barbs of the processes inside this ambient as well as the barbs of the process hidden behind the open capability. The scheduler of the opened ambient is deleted. In TR-Resource, the time consuming process moves into the scheduler, where it awaits the distribution of a time slice as resource before it

$$\frac{\textsc{Sdl} = \textsc{Sched}_{speed}\{in, 0, 0, \emptyset, \emptyset\},\ \textsc{Sdl}' = \textsc{Sched}_{speed}\{in+1, 0, 0, \emptyset, \emptyset\},\ R \not\equiv \mathbf{c}\,.P \mid P'}{a[\,\texttt{tick} \mid \textsc{Sdl} \mid R] \to a[\textsc{Sdl}' \mid R]}\ \text{(RR-Empty)}$$

$$\frac{\begin{array}{l}\textsc{Sdl} = \textsc{Sched}_{speed}\{in, 0, 0, U, S\}, \qquad U \cup S \neq \emptyset\\ \textsc{Sdl}' = \textsc{Sched}_{speed}\{in+1, x, z, U, S\}, \qquad speed = x + \sum_{y=1}^{z} \frac{1}{b_y},\ b_y > 1\end{array}}{a[\,\texttt{tick} \mid \textsc{Sdl} \mid R] \to a[\textsc{Sdl}' \mid R]}\ \text{(RR-Tick)}$$

$$\frac{\begin{array}{l}\textsc{Sdl} = \textsc{Sched}_{speed}\{in, out, rest, \emptyset, S\}, \qquad R \not\equiv \mathbf{c}\,.P \mid P'\\ \textsc{Sdl}' = \textsc{Sched}_{speed}\{in, out, rest, S, \emptyset\}\end{array}}{a[\textsc{Sdl} \mid R] \to a[\textsc{Sdl}' \mid R]}\ \text{(RR-NewRound)}$$

$$\frac{\begin{array}{l}out > 0,\ a_i \in U,\ a_i \equiv \mathbf{c}\,.P, \qquad \textsc{Sdl} = \textsc{Sched}_{speed}\{in, out, rest, U, S\}\\ \textsc{Sdl}' = \textsc{Sched}_{speed}\{in, out-1, rest, U \setminus \{a_i\} \cup P_{\downarrow}, S\}\end{array}}{a[\textsc{Sdl} \mid R] \to a[\textsc{Sdl}' \mid R \mid P]}\ \text{(RR-Tock}_{\text{1-consume}}\text{)}$$

$$\frac{\begin{array}{l}out > 0,\ a_i \in U, \qquad R \equiv a_i[\textsc{Sdl}_{a_i} \mid P'] \mid P, \qquad R' \equiv a_i[\textsc{Sdl}_{a_i} \mid \texttt{tick} \mid P'] \mid P\\ \textsc{Sdl} = \textsc{Sched}_{speed}\{in, out, rest, U, S\}\\ \textsc{Sdl}' = \textsc{Sched}_{speed}\{in, out-1, rest, U \setminus \{a_i\}, S \cup \{a_i\}\}\end{array}}{a[\textsc{Sdl} \mid R] \to a[\textsc{Sdl}' \mid R']}\ \text{(RR-Tock}_{\text{1-ambient}}\text{)}$$

$$\frac{\begin{array}{l}rest > 0,\ in \bmod b_{rest} = 0,\ a_i \in U,\ a_i \equiv \mathbf{c}\,.P,\ speed = x + \sum_{y=1}^{z} \frac{1}{b_y},\ b_y > 1\\ \textsc{Sdl} = \textsc{Sched}_{speed}\{in, out, rest, U, S\}\\ \textsc{Sdl}' = \textsc{Sched}_{speed}\{in, out, rest-1, U \setminus \{a_i\} \cup P_{\downarrow}, S\}\end{array}}{a[\textsc{Sdl} \mid R] \to a[\textsc{Sdl}' \mid R \mid P]}\ \text{(RR-Tock}_{\text{2-consume}}\text{)}$$

$$\frac{\begin{array}{l}rest > 0,\ in \bmod b_{rest} = 0,\ a_i \in U, \qquad speed = x + \sum_{y=1}^{z} \frac{1}{b_y},\ b_y > 1\\ R \equiv a_i[\textsc{Sdl}_{a_i} \mid P'] \mid P, \qquad R' \equiv a_i[\textsc{Sdl}_{a_i} \mid \texttt{tick} \mid P'] \mid P\\ \textsc{Sdl} = \textsc{Sched}_{speed}\{in, out, rest, U, S\}\\ \textsc{Sdl}' = \textsc{Sched}_{speed}\{in, out, rest-1, U \setminus \{a_i\}, S \cup \{a_i\}\}\end{array}}{a[\textsc{Sdl} \mid R] \to a[\textsc{Sdl}' \mid R']}\ \text{(RR-Tock}_{\text{2-ambient}}\text{)}$$

$$\frac{\begin{array}{l}rest > 0,\ in \bmod b_{rest} \neq 0,\ speed = x + \sum_{y=1}^{z} \frac{1}{b_y},\ b_y > 1\\ \textsc{Sdl} = \textsc{Sched}_{speed}\{in, out, rest, U, S\}, \quad \textsc{Sdl}' = \textsc{Sched}_{speed}\{in, out, rest-1, U, S\}\end{array}}{a[\textsc{Sdl} \mid R] \to a[\textsc{Sdl}' \mid R]}\ \text{(RR-Tock}_{\text{2-no action}}\text{)}$$

$$\frac{\textsc{Sdl}^{\dagger} = \textsc{Sched}^{\dagger}\{in, 0, 0, -, U, S\},\ \textsc{Sdl}^{\dagger}_{*} = \textsc{Sched}^{\dagger}\{in+1, 1, 0, -, U, S\}}{\textsc{Sdl}^{\dagger} \to \textsc{Sdl}^{\dagger}_{*}}\ \text{(RR-Root)}$$

Table 3. Reduction rules for fair, preemptive distribution of virtual time and resources, where $b_y \in \mathbb{N}$. A blue backdrop marks the reduction trigger and red the changes.

can continue. This reduction can only happen in virtually timed ambients with speed greater zero, meaning ambients which actually emit resources.

The RR-Tick and RR-Tock rules in Table 3 distribute time slices via the local schedulers. We want to enable the schedulers to distribute time slices as soon as possible. The ratio of output time slices to input time slices is defined by the *speed* $\in \mathbb{Q}$ of the scheduler. For example, for a speed of $^3/_2$ the first incoming `tick` should trigger one outgoing time slice and the second input should trigger two, emitting in total three time slices for two inputs. Thus, in order to implement a simple *eager scheduling strategy*, we make use of the so-called *Egyptian fraction decomposition* to decide the number of time slices to be distributed by a local scheduler for each input time slice `tick`. For every rational number $q \in \mathbb{Q}$ it holds that $q = x + \sum_{y=1}^{z} \frac{1}{b_y}$ for $x, b_y \in \mathbb{N}$, which is solvable in polynomial time. A greedy algorithm (e.g. [6]) yields the desirable property that a time slice is distributed as soon as possible. From this decomposition, it follows that for each input time slice the local scheduler with speed q will distribute x time slices, plus one additional time slice for every b_y-th input. In RR-Tick, the local scheduler receives a time slice, which it registers in the counter *in*. At the same time *out* and *rest* initiate the distribution of time slices depending on the Egyptian fraction decomposition of the speed of the scheduler. These steps of the time slice distribution are shown in the RR-Tock rules, which allow transferring a new `tick` to a timed subambient or using the time slice as a resource for a consume capability, which is waiting in the scheduler. The RR-Tock$_1$ rules concern the number x of time slices that are given out for every input time slice, while the RR-Tock$_2$ rules only allow to give out a time slice if the input step is a multiple of one of the fraction denominators b_y. This amounts to a concrete implementation of a fair scheduler where progress is uniform over the queue of timed subambients and time consuming processes. Once all waiting subambients and processes inside the set *unserved* have been served one time slice and are moved to the set *served*, either the rule RR-NewRound ensures that the next round of time slice distribution can begin, or, if the queue is empty, the rule RR-Empty is applied. This scheduling strategy ensures fairness in the competition for resources between processes, without enforcing a particular order in each round of the scheduler. The root scheduler Sched† reduces without time slices from surrounding ambients in RR-Root.

Example 1 (Virtually timed subambients, scheduling and resource consumption). The virtually timed ambient *cloud*, exemplifying a cloud server, emits one time slice for every time slice it receives, $\textsc{Sdl}_{cloud} = \textsc{Sched}_1\{0,0,0,\emptyset,\emptyset\}$. It contains two `tick` and is entered by a virtually timed subambient *vm*.

$$\begin{aligned} &\mathit{cloud}[\textsc{Sched}_1\{0,0,0,\emptyset,\emptyset\} \mid \texttt{tick} \mid \texttt{tick}] \\ &\mid \mathit{vm}[\textsc{Sched}_{3/4}\{0,0,0,\emptyset,\emptyset\} \mid \mathbf{in}\ \mathit{cloud}.\ \mathbf{c}\ .P] \end{aligned}$$

The ambient *vm* exemplifies a virtual machine containing a resource consuming task, where $\textsc{Sdl}_{vm} = \textsc{Sched}_{3/4}\{0,0,0,\emptyset,\emptyset\}$. The Egyptian fraction decomposition of the speed yields $^3/_4 = 0 + {^1/_2} + {^1/_4}$ meaning that there is no time slice given out for every incoming time slice, but one time slice for every second incoming time slice, and one for every fourth. The process reduces as follows:

$$\begin{aligned}
&\rightarrowtriangle \mathit{cloud}\,[\textsc{Sched}_1\{0,0,0,\emptyset,\mathit{vm}\} \mid \texttt{tick} \mid \texttt{tick} \\
&\quad \mid \mathit{vm}[\textsc{Sched}_{3/4}\{0,0,0,\emptyset,\emptyset\} \mid \mathbf{c}\,.P]] && (\text{TR-}\textsc{In}) \\
&\rightarrowtriangle \mathit{cloud}\,[\textsc{Sched}_1\{0,0,0,\mathit{vm},\emptyset\} \mid \texttt{tick} \mid \texttt{tick} \\
&\quad \mid \mathit{vm}[\textsc{Sched}_{3/4}\{0,0,0,\emptyset,\emptyset\} \mid \mathbf{c}\,.P]] && (\text{RR-}\textsc{NewRound}) \\
&\rightarrowtriangle \mathit{cloud}\,[\textsc{Sched}_1\{0,0,0,\mathit{vm},\emptyset\} \mid \texttt{tick} \mid \texttt{tick} \\
&\quad \mid \mathit{vm}[\textsc{Sched}_{3/4}\{0,0,0,\emptyset,\mathbf{c}\,.P\} \mid \mathbf{0}]] && (\text{TR-}\textsc{Resource}) \\
&\rightarrowtriangle \mathit{cloud}\,[\textsc{Sched}_1\{0,0,0,\mathit{vm},\emptyset\} \mid \texttt{tick} \mid \texttt{tick} \\
&\quad \mid \mathit{vm}[\textsc{Sched}_{3/4}\{0,0,0,\mathbf{c}\,.P,\emptyset\} \mid \mathbf{0}]] && (\text{RR-}\textsc{NewRound})\ .
\end{aligned}$$

Here the ambient *vm* enters the ambient *cloud* and is registered in the scheduler. Furthermore, the resource consuming process in *vm* is registered. In the next steps the time slices move into the scheduler of the *cloud* ambient and are distributed further down in the hierarchy.

$$\begin{aligned}
&\rightarrowtriangle \mathit{cloud}\,[\textsc{Sched}_1\{1,1,0,\mathit{vm},\emptyset\} \mid \texttt{tick} \\
&\quad \mid \mathit{vm}[\textsc{Sched}_{3/4}\{0,0,0,\mathbf{c}\,.P,\emptyset\} \mid \mathbf{0}]] && (\text{RR-}\textsc{Tick}) \\
&\rightarrowtriangle \mathit{cloud}\,[\textsc{Sched}_1\{1,0,0,\emptyset,\mathit{vm}\} \mid \texttt{tick} \\
&\quad \mid \mathit{vm}[\textsc{Sched}_{3/4}\{0,0,0,\mathbf{c}\,.P,\emptyset\} \mid \texttt{tick}]] && (\text{RR-}\textsc{Tock}_{\textsc{1-ambient}}) \\
&\rightarrowtriangle \mathit{cloud}\,[\textsc{Sched}_1\{2,0,0,\mathit{vm},\emptyset\} \\
&\quad \mid \mathit{vm}[\textsc{Sched}_{3/4}\{0,0,0,\mathbf{c}\,.P,\emptyset\} \mid \texttt{tick}]] && (\text{RR-}\textsc{NewRound}) \\
&\rightarrowtriangle \mathit{cloud}\,[\textsc{Sched}_1\{2,1,0,\mathit{vm},\emptyset\} \\
&\quad \mid \mathit{vm}[\textsc{Sched}_{3/4}\{0,0,0,\mathbf{c}\,.P,\emptyset\} \mid \texttt{tick}]] && (\text{RR-}\textsc{Tick}) \\
&\rightarrowtriangle \mathit{cloud}\,[\textsc{Sched}_1\{2,0,0,\emptyset,\mathit{vm}\} \\
&\quad \mid \mathit{vm}[\textsc{Sched}_{3/4}\{0,0,0,\mathbf{c}\,.P,\emptyset\} \mid \texttt{tick} \mid \texttt{tick}]] && (\text{RR-}\textsc{Tock}_{\textsc{1-ambient}}) \\
&\rightarrowtriangle \mathit{cloud}\,[\textsc{Sched}_1\{2,0,0,\mathit{vm},\emptyset\} \\
&\quad \mid \mathit{vm}[\textsc{Sched}_{3/4}\{0,0,0,\mathbf{c}\,.P,\emptyset\} \mid \texttt{tick} \mid \texttt{tick}]] && (\text{RR-}\textsc{NewRound})\ .
\end{aligned}$$

Now the ambient *vm* can use the time signals to enable resource consumption.

$$\begin{aligned}
&\rightarrowtriangle \mathit{cloud}\,[\textsc{Sched}_1\{2,0,0,\mathit{vm},\emptyset\} \\
&\quad \mid \mathit{vm}[\textsc{Sched}_{3/4}\{1,0,1,\mathbf{c}\,.P,\emptyset\} \mid \texttt{tick}]] && (\text{RR-}\textsc{Tick}) \\
&\rightarrowtriangle \mathit{cloud}\,[\textsc{Sched}_1\{2,0,0,\mathit{vm},\emptyset\} \\
&\quad \mid \mathit{vm}[\textsc{Sched}_{3/4}\{1,0,0,\mathbf{c}\,.P,\emptyset\} \mid \texttt{tick}]] && (\text{RR-}\textsc{Tock}_{\textsc{2-no action}}) \\
&\rightarrowtriangle \mathit{cloud}\,[\textsc{Sched}_1\{2,0,0,\mathit{vm},\emptyset\} \\
&\quad \mid \mathit{vm}[\textsc{Sched}_{3/4}\{2,0,1,\mathbf{c}\,.P,\emptyset\} \mid \mathbf{0}]] && (\text{RR-}\textsc{Tick}) \\
&\rightarrowtriangle \mathit{cloud}\,[\textsc{Sched}_1\{2,0,0,\mathit{vm},\emptyset\} \\
&\quad \mid \mathit{vm}[\textsc{Sched}_{3/4}\{2,0,0,P_{\downarrow},\emptyset\} \mid P]] && (\text{RR-}\textsc{Tock}_{\textsc{2-consume}})
\end{aligned}$$

Note that as the calculus is non-deterministic, the reduction rules can be applied in arbitrary order, making several outcomes possible.

Context:		**Context expressions:**	
$E ::= \mathbf{0}$	nil	$\kappa ::= \textsc{True}$	true
$\mid \odot$	hole	$\mid \odot$	hole
$\mid n[E]$	location	$\mid \neg\kappa$	negation
$\mid E \mid P$	parallel composition	$\mid \kappa_1 \mid \kappa_2$	parallel composition
		$\mid \kappa_1 \wedge \kappa_2$	conjunction
		$\mid n[\kappa]$	location
		$\mid \oplus\kappa$	spatial next modality
		$\mid \lozenge_{(speed,s)}\kappa$	somewhere modality
		$\mid \diamond_{x@n}\kappa$	sometime modality
		$\mid \exists x.\kappa$	existential quantification
		$\mid \mathbf{c}$	consumption

Table 4. Syntax of contexts and context expressions

3 Resource-Aware Virtually Timed Ambients

We now consider *context-guarded actions* for the calculus of virtually timed ambients, building on properties of context aware ambients [21].

Definition 5 (Resource-aware virtually timed ambients). *The syntax of resource-aware virtually timed ambients is given by the grammar in Table 1 together with the process*

$$\kappa.P \text{ (context-guarded process)},$$

where κ is a context expression. The semantics of resource-aware virtually timed ambients is given by the reduction rules in Tables 2 and 3 and the rule

$$\frac{E \vDash \kappa}{E(\kappa?P) \rightharpoonup E(P)} \qquad \text{(TR-Context)}.$$

A context-guarded process $\kappa?P$ has to fulfil a context requirement before it can be reduced, meaning that a guard is removed when it is satisfied by the environment. The context model is given in Table 4, where E denotes a context or environment and $\mathbf{0}$ is the empty context. Ambient names and processes are defined as in Table 1. The symbol $\odot$ is the hole context, showing the position of a process in the surrounding context. A ground context is defined to be a normal process with no holes. Multi-hole contexts are omitted.

Definition 6 (Context evaluation, from [21]). *Let E_1 and E_2 be contexts. The evaluation of context E_1 at context E_2, denoted $E_1(E_2)$, is the context obtained by replacing the hole in E_1 (if any) by E_2 as follows*

$$E_1(E_2) = \begin{cases} E_1 & \text{if } E_1 \text{ is a ground context,} \\ E_1\{\odot \leftarrow E_2\} & \text{otherwise,} \end{cases}$$

where $E_1\{\odot \leftarrow E_2\}$ is the substitution of E_2 for $\odot$ in E_1.

Context expressions are defined in Table 4. We enhance the context expressions for context-aware ambients from [21] with a *consumption* formula, stating the existence of consume capabilities in a process, as well as resource-aware *sometime* and *somewhere* modalities capturing the number of resources consumed in a certain ambient during the reduction, and the relative speed and number of siblings of the target ambient, respectively. To expose these numbers in reductions, we define a *labeled reduction relation.* While $\rightarrowtriangle$ refers to all reduction steps in virtually timed ambients, we denote by $\xrightarrow{\texttt{tick}}$ the steps of the (RR-Tick) rule, i.e., the internal reductions in the schedulers enabling timed reduction of processes. All other reduction steps are marked by $\xrightarrow{\tau}$.

Definition 7 (Tick-reduction). *$P \xrightarrow{\texttt{tick}} P'$ iff $P \mid \texttt{tick} \rightarrow P'$. We write $\xrightarrow{\texttt{tick}^x}$ if x time signals* tick *are used; i.e., $P \xrightarrow{\texttt{tick}^x} P'$ iff $P \mid \texttt{tick} \mid \cdots \mid \texttt{tick} \rightarrow^* P'$, where the number of time signals* tick *is x. The weak version of this reduction is defined as $P \xRightarrow{\texttt{tick}^x} P'$ iff $P(\xrightarrow{\tau}^* \xrightarrow{\texttt{tick}} \xrightarrow{\tau}^*)^x P'$, where $\xrightarrow{\tau}^*$ describes the application of an arbitrary number of τ-steps.*

The relation $\xRightarrow{\texttt{tick}^x}_n$ captures the number of resources used inside an ambient n inside a process.

Definition 8 (Tick-reduction inside an ambient). *$P \xRightarrow{\texttt{tick}^x}_n P'$ iff $P \rightarrowtriangle^* P'$ and there exists Q, Q' such that $P \downarrow^* n[Q]$, $P' \downarrow^* n[Q']$ and $Q \xRightarrow{\texttt{tick}^x} Q'$.*

Lastly, we define *accumulated speed* [10] based on the eager distribution strategy for time slices. The accumulated speed $accum\{m\}_P \in \mathbb{Q}$ in a subambient m which is part of a process P, is the relative speed of the ambient m with respect to the speed of the parental ambient and the siblings of m.

Definition 9 (Accumulated speed). *Let $speed_k \in \mathbb{Q}$ and $children(k)$ denote the speed and number of children of a virtually timed ambient* k. *Let* m *be a timed subambient of a process P, the name* parent *denoting the direct parental ambient of* m*, and* C *the path of all parental ambients of* m *up to the level of P. The accumulated speed for preemptive scheduling in a subambient* m *up to the level of the process P is given by*

$$\begin{aligned} accum\{m\}_P &= speed_m \cdot {}^1\!/_{children(parent)} \cdot speed_{parent} \\ &= speed_m \cdot \prod_{k \in C} {}^1\!/_{children(k)} \cdot \prod_{k \in C} speed_k \end{aligned}$$

Schedulers distribute time slices preemptively, as child processes get one time slice at a time in iterative rounds. Consequently, an ambient's accumulated speed is influenced by both the speed and the number of children n of the parental ambient. Thus, scheduling is not only *path sensitive* but also *sibling sensitive.*

The formal semantics for context expressions is given by the satisfaction relations in Table 5. The spatial reduction relation $\downdownarrows$, which describes the option to go exactly one step deep into the nesting of ambients, is defined as follows.

$E \vDash \textsc{True}$		
$E \vDash \odot$	iff	$E = \odot$
$E \vDash \neg\kappa$	iff	$E \nvDash \kappa$
$E \vDash \kappa_1 \mid \kappa_2$	iff	exist E_1, E_2, such that $E = E_1 \mid E_2$ and $E_1 \vDash \kappa_1$ and $E_2 \vDash \kappa_2$
$E \vDash \kappa_1 \wedge \kappa_2$	iff	$E \vDash \kappa_1$ and $E \vDash \kappa_2$
$E \vDash n[\kappa]$	iff	exist E', such that $E = n[E']$ and $E' \vDash \kappa$
$E \vDash \oplus\kappa$	iff	exist E', such that $E \downharpoonleft\downharpoonright E'$ and $E' \vDash \kappa$
$E \vDash \Diamond_{(speed,s)}\kappa$	iff	exists E', E'', n s.t. $(E \equiv n[\textsc{Sdl} \mid E'] \mid E'' \vee E \downharpoonleft\downharpoonright^* n[\textsc{Sdl} \mid E'])$
		$\wedge E' \vDash \kappa \wedge \mathit{accum}\{n\}_E \geq \mathit{speed} \wedge \lvert U_{\textsc{Sdl}} \cup S_{\textsc{Sdl}} \rvert \leq s$
$E \vDash \diamond_{x@n}\kappa$	iff	exist E', such that $E \xRightarrow{\mathtt{tick}^y}_n E'$, $y \leq x$ and $E' \vDash \kappa$
$E \vDash \exists x.\kappa$	iff	exist n, such that $E \vDash \kappa\{x \leftarrow n\}$
$E \vDash \mathbf{c}$	iff	exist E', E'', E''', such that $E \downarrow^* E'$ and $E' \equiv E''.\ \mathbf{c}\ .E'''$

Table 5. Satisfaction relation for context expressions

Definition 10 (Spatial reduction). *$E \downharpoonleft\downharpoonright E'$ iff there exist a name n and context E'' such that $E = (n[E'] \mid E'')$ and $\downharpoonleft\downharpoonright^*$ is the reflexive and transitive closure.*

Thus, the *spatial next modality* $\oplus$ is satisfied if and only if the expression following it is satisfied after stepping one level down in the context. The *consumption* expression **c** is satisfied by any context which contains a consumption capability anywhere inside. A context E satisfies the *sometime modality* if and only if it can reduce to a context satisfying the formula, while using less than x resources in the ambient n in the reduction. Lastly, the *somewhere modality* is satisfied if and only if there exists a subcontext of E satisfying the formula and the relative speed in the sublocation containing the context is greater or equal the given *speed* and the sublocation has less or equal than s timed subambients.

We use the virtually timed system from Example 1 to show the meaning of some context expressions.

Example 2 (Context expressions and context-guards). It holds that

$$\mathit{cloud}[\textsc{Sdl}_{\mathit{cloud}} \mid \mathtt{tick} \mid \mathtt{tick} \mid \mathit{vm}[\textsc{Sdl}_{\mathit{vm}} \mid \mathbf{c}\ .\mathbf{0}]] \vDash \diamond_{2@\mathit{vm}}\neg\mathbf{c}.$$

This means that two resources are used in the virtual timed ambient before the consume capability reduces. This reduction can be seen in Example 1. Further, it holds that

$$\mathit{cloud}[\textsc{Sdl}_{\mathit{cloud}} \mid \odot \mid \mathtt{tick} \mid \mathtt{tick} \mid \mathit{vm}[\textsc{Sdl}_{\mathit{vm}} \mid \mathbf{c}\ .P]] \vDash \oplus(\mathit{vm}[\textsc{True}] \mid \odot \mid \textsc{True})$$

or, omitting the hole,

$$\mathit{cloud}[\textsc{Sdl}_{\mathit{cloud}} \mid \mathtt{tick} \mid \mathtt{tick} \mid \mathit{vm}[\textsc{Sdl}_{\mathit{vm}} \mid \mathbf{c}\ .P]] \vDash \oplus(\mathit{vm}[\textsc{True}] \mid \textsc{True})$$

as there is an ambient named *vm* directly under the top level in the system. Using the context expression as context-guard we can define the following process

$$\mathit{cloud}[\textsc{Sdl}_{\mathit{cloud}} \mid \oplus\mathit{vm}[\textsc{True}]?\ \mathbf{open}\ \mathit{vm} \mid \mathtt{tick} \mid \mathtt{tick} \mid \mathit{vm}[\textsc{Sdl}_{\mathit{vm}} \mid \mathbf{c}\ .P]],$$

which aims to open the subambient *vm* if it is the only process on the top level in the system. As this is true after the ticks have been moved into the scheduler, the guard is removed and the process reduces to $\mathit{cloud}[\textsc{Sdl}_{\mathit{cloud}} \mid \mathbf{c}\ .P]$.

Weak timed context bisimulation. We define *weak timed context bisimulation* for resource-aware ambients, which extends the definition of weak bisimulation for virtually timed ambients [11] by treating the context-guarded processes as τ actions in the timed labelled transition system and adding notions of context bisimulation [19,20] to the bisimulation relation.

The following definitions make use of the notion of *timed systems*, which are special processes without capabilities on the outermost level.

Definition 11 (Timed systems). *Timed systems are given as follows:*

$$\begin{aligned} M, N ::= {} & \mathbf{0} \\ \mid {} & M \mid N \\ \mid {} & (\nu n)M \\ \mid {} & n[P], \end{aligned}$$

where P is a timed process as given in Table 1.

The behavior of a timed system interacting with its environment is given as a transmission system with transition labels.

Definition 12 (Labels). *Let the set of labels Lab, with typical element α, be given as follows:*

$$\begin{aligned} \alpha \in \mathit{Lab} ::= {} & \tau \\ \mid {} & k.\mathtt{enter\_}n \mid k.\mathtt{exit\_}n \mid k.\overline{\mathtt{enter\_}}n \mid n.\mathtt{open\_}k \\ \mid {} & *.\mathtt{exit\_}n \mid *.\mathtt{enter\_}n \\ \mid {} & k.\mathtt{tick} \end{aligned}$$

where k and n represent names of ambients. The label τ is called the internal label*, the rest are called* observable *labels. We refer to labels of the forms $*.\mathtt{exit\_}n$ and $*.\mathtt{enter\_}n$ as* anonymous *and other labels as* non-anonymous*, and let the* untimed *labels exclude the $k.\mathtt{tick}$ label.*

Note that the **c** capability does not represent an interaction with an environment but an internal action and is therefore not captured by a separate observable label apart from τ.

Definition 13 (Timed labeled transitions). *The observable steps $M \xrightarrow{\alpha} M'$ of the timed labeled transition semantics for timed systems is given by the rules of Table 6. For internal behavior, τ-steps are the result of reduction steps, i.e., $M \rightarrowtriangle M'$ implies $M \xrightarrow{\tau} M'$.*

The untimed labels, which record the system-environment interactions (i.e., ambient movements induced by capabilities), coincide with the labels from the untimed case of mobile ambients [14]. In 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 rule Co-Enter, an ambient n, provided by the environment, enters an ambient k of the process. In rule Open, the environment provides an

$$(\nu\widetilde{m})(m[\textsc{Sdl} \mid \mathbf{in}\ n.P \mid Q] \mid M), m \in \widetilde{m}$$
$$\xrightarrow{*.\mathtt{enter}\_n} (\nu\widetilde{m})(n[m[(\textsc{Sdl} \mid P) \mid Q] \mid \circ\ ] \mid M) \qquad (\textsc{Enter Shh})$$
$$(\nu\widetilde{m})(k[\textsc{Sdl} \mid \mathbf{in}\ n.P \mid Q] \mid M), k \notin \widetilde{m}$$
$$\xrightarrow{k.\mathtt{enter}\_n} (\nu\widetilde{m})(n[k[(\textsc{Sdl} \mid P) \mid Q] \mid \circ\ ] \mid M) \qquad (\textsc{Enter})$$
$$(\nu\widetilde{m})(m[\textsc{Sdl} \mid \mathbf{out}\ n.P \mid Q] \mid M), m \in \widetilde{m}$$
$$\xrightarrow{*.\mathtt{exit}\_n} (\nu\widetilde{m})(m[(\textsc{Sdl} \mid P) \mid Q] \mid n[M \mid \circ\ ]) \qquad (\textsc{Exit Shh})$$
$$(\nu\widetilde{m})(k[\textsc{Sdl} \mid \mathbf{out}\ n.P \mid Q] \mid M), k \notin \widetilde{m}$$
$$\xrightarrow{k.\mathtt{exit}\_n} (\nu\widetilde{m})(k[(\textsc{Sdl} \mid P) \mid Q] \mid n[M \ \mid \circ\ ]) \qquad (\textsc{Exit})$$
$$(\nu\widetilde{m})(k[(\textsc{Sdl}_k \mid P)] \mid M), k \notin \widetilde{m}$$
$$\xrightarrow{k.\overline{\mathtt{enter}}\_n} (\nu\widetilde{m})(k[\textsc{Sdl}^*_k \mid n[\circ] \mid P] \mid M) \qquad (\textsc{Co-Enter})$$
$$(\nu\widetilde{m})(k[(\textsc{Sdl} \mid P)] \mid M)$$
$$\xrightarrow{n.\mathtt{open}\_k} n[\ \circ \mid (\nu\widetilde{m})(P \mid M)] \qquad (\textsc{Open})$$
$$(\nu\widetilde{m})(k[\textsc{Sdl} \mid Q] \mid M), k \notin \widetilde{m}$$
$$\xrightarrow{k.\mathtt{tick}} (\nu\widetilde{m})(k[\textsc{Sdl} \mid \mathtt{tick} \mid Q] \mid M) \qquad (\textsc{Tick})$$

Table 6. Rules for timed labeled transition systems, where in (Co-Enter) given $\textsc{Sdl}_k = \textsc{Sched}_{speed_k}\{in, out, rest, unserved, served\}$ the updated scheduler is denoted by $\textsc{Sdl}^*_k = \textsc{Sched}_{speed_k}\{in, out, rest, unserved \cup \{n\}, served\}$ if $speed_n > 0$ as described in Table 2.

ambient n in which the ambient k of the process is opened. In rule Tick, the transition $M \xrightarrow{k.\mathtt{tick}} M'$ expresses that the top-level ambient k of the system M receives one time slice `tick` from the root scheduler on the global level.

The post-configurations after the transitions contain the symbol $\circ$, which is used as placeholder variable. The labels, which capture interaction with the environment, carry partial information about the "data" exchanged with the environment. For example, label $k.\mathtt{enter}\_n$ carries information about the identity k of the ambient being entered, which is contained in the system, as well as the identity of the entering ambient named n, which, before the step, is still part of the environment. If the enter-label conceptually indicates that some arbitrary ambient $n[R \mid \textsc{Sdl}]$ enters the system as an effect of executing the **in** n-capability, then the name n is mentioned as part of the label but its "body" $R \mid \textsc{Sdl}$ is not. We want to relate the actions of the two systems by a notion of bisimulation. Intuitively, if one system does a transition where $n[R \mid \textsc{Sdl}]$ enters, the second system must be able to exhibit the same transition, i.e., have the "same" ambient entering without breaking their (bi)simulation relationship. In principle, the second system can simulate the first doing a step where an ambient $n[R]$ enters, with the body $S \equiv R \mid \textsc{Sdl}$. To achieve that (without overburdening the labels by interpreting them up-to structural congruence $\equiv$), the definition uses the placeholder $\circ$ and requires preservation of the relationship for all *instantiations* of the placeholders for both systems by the same body (cf. Definition 14 below). The substitution of the placeholder by a pair consisting of a process and its local scheduler, is written as $P \bullet (\textsc{Sdl} \mid Q)$ 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 $\Longrightarrow$ denote the reflexive and transitive closure of $\xrightarrow{\tau}$, let $\overset{\alpha}{\Rightarrow}$ denote $\Longrightarrow\xrightarrow{\alpha}\Longrightarrow$, and let $\overset{\hat{\alpha}}{\Rightarrow}$ denote $\Longrightarrow$ if $\alpha = \tau$ and $\overset{\alpha}{\Rightarrow}$ otherwise.

Definition 14 (Weak timed context bisimulation). *A symmetric relation $\mathcal{R}$ over timed systems is a* weak timed context bisimulation *if $M \mathrel{\mathcal{R}} N$ and $M \xrightarrow{\alpha} M'$, $\alpha \in \{k.\texttt{enter}\_n, k.\texttt{exit}\_n, k.\overline{\texttt{enter}}\_n, n.\texttt{open}\_k, *.\texttt{exit}\_n, *.\texttt{enter}\_n, k.\texttt{tick}, \tau\}$ implies:*

1. *If α is a non-anonymous label, then $N \overset{\hat{\alpha}}{\Rightarrow} N'$ for some N', such that such that for all schedulers* Sched *and processes P it holds that $E[M' \bullet (\textsc{Sched} \mid P)] \mathrel{\mathcal{R}} E[N' \bullet (\textsc{Sched} \mid P)]$, for each context E.*
2. *For anonymous labels:*
   - *(a) If $\alpha = *.\texttt{enter}\_n$, then $N \mid n[\circ] \Longrightarrow N'$ for some N', such that for all schedulers* Sched *and processes P it holds that $E[M' \bullet (\textsc{Sched} \mid P)] \mathrel{\mathcal{R}} E[N' \bullet (\textsc{Sched} \mid P)]$, for each context E.*
   - *(b) If $\alpha = *.\texttt{exit}\_n$, then $n[\ \circ \mid N] \Longrightarrow N'$ for some N', such that for all schedulers* Sched *and processes P it holds that $E[M' \bullet (\textsc{Sched} \mid P)] \mathrel{\mathcal{R}} E[N' \bullet (\textsc{Sched} \mid P)]$, for each context E.*

The preservation of bisimilarity by system contexts follows from this definition:

Theorem 1. *Weak timed context bisimilarity is preserved by system contexts.*

The given bisimulation relation is a congruence. Furthermore, the relation coincides with reduction barbed congruence, defined as the largest relation which is preserved by all constructs of the language, by the internal steps of the reduction semantics, and by so-called barbs, which are simple observables of terms.

Definition 15 (Reduction barbed congruence over timed systems). Reduction barbed congruence over timed systems *is the largest symmetrical relation over timed systems which is preserved by all system contexts, is reduction closed and barb preserving.*

We can show that the bisimulation relation coincides with reduction barbed congruence by following the proof given in [11].

Theorem 2. *Weak timed context bisimulation and reduction barbed congruence over resource-aware virtually timed systems coincide.*

4 Implementation and Case Study

We implement resource-aware virtually timed ambients in the rewriting logic system Maude [5,17]. Rewriting logic is a flexible semantic and logical framework which can be used to represent a wide range of systems with *low representational distance* [15]. Rewriting logic embeds *membership equational logic*, which lets a specification or program contain both equations and rewrite rules. When executing a Maude specification, rewrite steps are applied to normal forms in the equational logic. Both equations and rewrite rules may be *conditional*, meaning that specified conditions must hold for the rule or equation to apply.

The calculus of virtually timed ambient and a modal logic model checker for virtually timed ambients have been implemented in Maude [12]. The timed reduction rules (Tables 2 and 3) are represented as rewrite rules and modal logic formulas are built from operator declarations in Maude. We now extend this implementation with guarded processes and the corresponding reduction rule, and with replication and restricted names, thereby allowing non-unique names for ambients [1]. The syntax of resource-aware virtually timed ambients, given in Table 3, is represented by Maude terms, constructed from operators:

```
op zero : -> VTA [ctor] .
op _|_ : VTA VTA -> VTA [id: zero assoc comm] .
op _._ : Capability VTA -> VTA .
op _[_|_] : Name Scheduler VTA -> VTA .
op !_ : VTA -> VTA .
op !<_>_ : Names VTA -> VTA .
op _?_ : Formula VTA -> VTA [frozen (2)] .
```

The correlation between our formal definition and the Maude specification is easy to see. The operator `zero` represents the inactive process. Parallel composition has the algebraic properties of being associative, commutative and having `zero` as identity element. Concatenation is represented by a dot. Virtually timed ambients are represented by a name followed by brackets containing a scheduler and processes. Replication is represented by an exclamation mark and context-guarded processes by a question mark. The `frozen` attribute prevents subterms behind the guard from being rewritten before the guard has been resolved.

The prototype implementation currently covers a *negation free fragment* of the logic. Context expressions (defined in Table 4) are implemented explicitly as modal logic formulas, their duals have been implemented as necessary for the case study. We explain the implementation of the reduction rules by the rewrite rule for context-guards, corresponding to TR-Context. The guards express global properties, which make it necessary to capture the entire environment. This is achieved by wrapping the top-level ambient in brackets `op {_} : VTA -> VTA`, which syntactically distinguish the top-level ambient from subambients. Using these brackets, we can express that a rewrite rule may only be applied at the global level. Guards are resolved by invoking the given modal logic model checker for virtually timed ambients during execution:

```
crl [RemoveGuard] : { P } => { removeGuard(P, G) }
  if G, Gs := findGuards(P) /\
     removeGuardedProcess(P, G) |= G => true .
```

[1] The full source code for the calculus and the case study is available at: `https://github.com/larstvei/Check-VTA/tree/resource-aware`

Here, `findGuards(P)` provides the set of all active guards found in the process, and some guard `G` is arbitrarily selected. Using the satisfaction relation `|=` the model checker is invoked on the top-level ambient, where the operation `removeGuardedProcess(P, G)` removes the guard together with the process behind it and thus yields the environment of the guarded process. If the environment satisfies the guard, the guard is removed by `removeGuard(P, G)`.

A case study of dynamic auto scaling on Amazon EC2. In the following, we show how resource-aware virtually timed ambients can model dynamic auto scaling of Amazon EC2 instances, based on the Auto Scaling User Guide by Amazon Web Services [1]. An auto scaling group is a collection of EC2 instances, which are essentially virtual machines, illustrated in Fig. 1. The user can specify the minimum and maximum number of instances in an auto scaling group, and auto scaling ensures that the given group never goes below or above these bounds. By specifying scaling policies the user enables auto scaling to adjust the number of instances depending on the demand on the application.

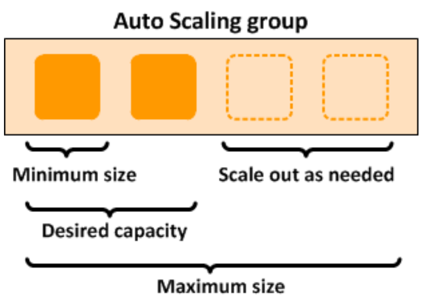

Fig. 1. Example of an auto scaling group as given in [1].

We model a cloud server as a top-level ambient with a scheduler `sdl('asg)`, an auto scaling group `asg`, a garbage bin `garbage`, and a number of requests, demanding resources. A minimal example of scaling can be given by using two requests `request(2)`, each expecting two resources:

```
op example : -> VTA .
eq example =
    { 'cloud[sdl('asg) | asg | request(2) | request(2) | garbage] } .
```

The expressions `asg`, `garbage` and `request(2)` reduce to resource-aware virtually timed ambients, containing other ambients and processes. For example, `request(K)` is an ambient containing an empty scheduler `sdl`, movement capabilities and a number `K` of consume capabilities, representing load on the machine.

```
eq request(K) =
    'request[sdl | in('asg) . open('move) . zero | consumes(K)] .
```

The ambient asg, which models the auto scaling group, manages the virtual machines and dynamically scales depending on the load. A request may enter the asg where an idle virtual machine seizes it or, if no virtual machine is idle and the maximal number of virtual machines is not reached, the asg scales up and produces a new virtual machine to handle the request. Scaling up is achieved by means of replicated ambients with restricted names, representing new virtual machines, protected by a scaling guard which realizes the scaling policy:

```
eq scalingGuard(MIN, MAX) =
    (+) someone('asg[<> 0 MIN someone('isRegistry[True]) \/
                     someone('request[True]) /\
                     <> 0 MAX someone('isRegistry[True]) /\
                     no-one(NotConsume /\ (+) someone('isVM[True]))]) .
```

The guard checks the number of virtual machines, their load and the existence of a request. The formula someone(F) is introduced to capture a recurring pattern in the case study, namely the satisfaction of a formula by one ambient in the process. The dual is expressed by no-one(F). The guard uses the somewhere modality and is satisfied if there are less then the minimal number of subambients in a 'registry ambient (marked by the subambient 'isRegistry) which contains a subambient for every active virtual machine. It is also satisfied if there exists a request inside the auto scaling group, the maximal number of machines is not reached, and there is no idle virtual machine (marked by the property NotConsume and the subambient 'isVM). Idle virtual machines move into the garbage ambient, if the number of virtual machines is not below the minimum. By running the example in Maude, we can see how the scaling process and the virtual machines react dynamically to the load on the auto scaling group.

```
Maude> frew example .
result VTA:
{'cloud[sched 0{0,0,0,'asg,none}
  | 'asg[sched 1{10,0,0,'vm1,none}
    | ...
    | !< 'vm > (open('scaling_lock) . scalingGuard ? scalingProcess)
    | 'vm1[...]]
  | 'garbage[sched 0{0,0,0,'vm0,none}
    | 'vm0[...] | ...]]}
```

Initially, the auto scaling group produces a virtual machine 'vm0[...], in accordance with the scaling policy which requires at least one running instance. The first request is handled by 'vm0[...], and a new virtual machine 'vm1[...] is produced and handles the second request. Once 'vm0[...] has resolved its request, it moves itself into the garbage ambient. The second virtual machine 'vm1[...] is prevented from deleting itself, due to the scaling policy. The model autonomously creates virtual machines to deal with incoming requests and scales back down when the machines are not needed anymore.

5 Related Work

The calculus of virtually timed ambients, first introduced in [10], is based on mobile ambients [3]. Mobile ambients model both location mobility and nested locations, and capture processes executing at distributed locations in networks such as the Internet. Gordon proposed a simple formalism for virtualization (without notions of timing or resources) loosely based on mobile ambients in [8]. The calculus of virtually timed ambients [10,11] stays closer to the syntax of the original mobile ambient calculus, while at the same time including notions of time and explicit resource provisioning. Our notion of resource provisioning extends work on deployment models in ABS [9] to additionally cover nested virtualization and the capabilities of mobile ambients. Resource-awareness for virtually timed ambients draws on the Calculus of Context Aware Ambients [21] which introduces context-guarded processes to enable context-awareness of mobile ambients. The context expressions in this paper are adapted to cover the timing and resource aspects of virtually timed ambients.

Cardelli and Gordon defined a labeled transition system for their mobile ambients [4], but no bisimulation. A bisimulation relation for a fragment of mobile ambients, called mobile safe ambients, is defined in [13] and provides the basis for later work. A labelled bisimulation for mobile ambients is defined by Merro and Nardelli [14], who prove that this bisimulation is equivalent to reduction barbed congruence and develop up-to-proof techniques. The weak timed bisimulation defined in [11] is a conservative extension of this approach, which is extended further in this paper using notions of context bisimulation developed in [19,20].

In [12] we use the Maude [5] system to implement a model checker, exploiting the low representational distance which distinguishes Maude [15]. The reduction rules for mobile ambients as well as a type system have been implemented in Maude in [18]. In contrast, our implementation focuses on capturing the timed reduction rules of virtually timed ambients as well as the modal formulas to define guards and resource-awareness.

6 Concluding Remarks

Virtualization opens for new and interesting foundational models of computation by explicitly emphasizing deployment and resource management. The calculus of virtually timed ambients is a formal model of hierarchical locations of execution with explicit resource provisioning. Resource provisioning for virtually timed ambients is based on virtual time, a local notion of time reminiscent of time slices provisioned by an operating system to virtual machines in the context of nested virtualization. This paper introduces resource-awareness for virtually timed ambients, which enables horizontal scaling. We define weak timed context bisimulation for resource-aware virtually timed ambients as an extension of bisimulation for mobile ambients. We implement the calculus in the rewriting logic system Maude and illustrate its use by a case study of dynamic auto scaling. Future work aims to develop optimization strategies for resource-aware scaling as well as a notion of higher order resources.

References

1. Amazon Web Services. Auto scaling user guide. `http://docs.aws.amazon.com/autoscaling/latest/userguide/as-dg.pdf`. Accessed: 2017-06-21.
2. 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 *Proceedings 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2010)*, pages 423–436. USENIX Association, 2010.
3. L. Cardelli and A. D. Gordon. Mobile ambients. *Theoretical Computer Science*, 240(1):177–213, 2000.
4. L. Cardelli and A. D. Gordon. Equational properties of mobile ambients. *Mathematical Structures in Computer Science*, 13(3):371–408, 2003.
5. M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. L. Talcott, editors. *All About Maude — A High-Performance Logical Framework, How to Specify, Program and Verify Systems in Rewriting Logic*, volume 4350 of *Lecture Notes in Computer Science*. Springer, 2007.
6. Fibonacci. Greedy algorithm for Egyptian fractions. Available at `https://en.wikipedia.org/wiki/Greedy_algorithm_for_Egyptian_fractions`.
7. R. P. Goldberg. Survey of virtual machine research. *IEEE Computer*, 7(6):34–45, 1974.
8. A. D. Gordon. V for virtual. *Electronic Notes in Theoretical Computer Science*, 162:177–181, 2006.
9. E. B. Johnsen, R. Schlatte, and S. L. Tapia Tarifa. Integrating deployment architectures and resource consumption in timed object-oriented models. *Journal of Logic and Algebraic Methods in Programming*, 84(1):67–91, 2015.
10. E. B. Johnsen, M. Steffen, and J. B. Stumpf. A calculus of virtually timed ambients. In P. James and M. Roggenbach, editors, *Postproceedings of the 23rd International Workshop on Algebraic Development Techniques (WADT 2016)*, volume 10644 of *Lecture Notes in Computer Science*, pages 88–103. Springer, 2017.
11. E. B. Johnsen, M. Steffen, and J. B. Stumpf. Virtually timed ambients: A calculus of nested virtualization. *Journal of Logical and Algebraic Methods in Programming*, 94:109 – 127, 2018.
12. E. B. Johnsen, M. Steffen, J. B. Stumpf, and L. Tveito. Checking modal contracts for virtually timed ambients, 2018. Submitted for publication. Available at `www.ifi.uio.no/~johanbst/MLCheckingVTAS.pdf`.
13. 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.
14. M. Merro and F. Zappa Nardelli. Behavioral theory for mobile ambients. *Journal of the ACM*, 52(6):961–1023, 2005.
15. J. Meseguer. Twenty years of rewriting logic. *Journal of Logic and Algebraic Programming*, 81(7-8):721–781, 2012.
16. R. Milner and D. Sangiorgi. Barbed bisimulation. In W. Kuich, editor, *Proceedings of ICALP '92*, volume 623 of *Lecture Notes in Computer Science*, pages 685–695. Springer, 1992.
17. P. C. Ölveczky. *Designing Reliable Distributed Systems – A Formal Methods Approach Based on Executable Modeling in Maude.* Undergraduate Topics in Computer Science. Springer, 2018.
18. F. Rosa-Velardo, C. Segura, and A. Verdejo. Typed mobile ambients in Maude. *Electronic Notes in Theoretical Computer Science*, 147(1):135 – 161, 2006. Proceedings of the 6th International Workshop on Rule-Based Programming.

19. D. Sangiorgi. Bisimulation for higher-order process calculi. *Information and Computation*, 131(2):141 – 178, 1996.
20. D. Sangiorgi and D. Walker. *The Pi-Calculus: A Theory of Mobile Processes.* Cambridge University Press, 2001.
21. F. Siewe, H. Zedan, and A. Cau. The calculus of context-aware ambients. *Journal of Computer and System Sciences*, 77(4):597 – 620, 2011.
22. D. Williams, H. Jamjoom, and H. Weatherspoon. The Xen-Blanket: Virtualize once, run everywhere. In *Proceedings 7th European Conference on Computer Systems (EuroSys'12)*, pages 113–126. ACM, 2012.