Int J Inf Secur (2004) 2: 126–144 / Digital Object Identifier (DOI) 10.1007/s10207-004-0035-y

Using ambients to control resources

David Teller[1], **Pascal Zimmer**[2], **Daniel Hirschkoff**[1]

[1] LIP – ENS Lyon, France
e-mail: {David.Teller,Daniel.Hirschkoff}@ens-lyon.fr
[2] INRIA Sophia Antipolis, France
e-mail: Pascal.Zimmer@sophia.inria.fr

Published online: 29 June 2004 –

Abstract. Current software and hardware systems, being parallel and reconfigurable, raise new safety and reliability problems, and the resolution of these problems requires new methods. Numerous proposals aim at reducing the threat of bugs and preventing several kinds of attacks. In this paper, we develop an extension of the calculus of mobile ambients, named *controlled ambients*, that is suited for expressing such issues, specifically denial of service attacks. We present a type system for controlled ambients, which makes static resource control possible in our setting, and enhance it with a rich notion of resources.

Keywords: Global computing – Process algebra – Resource control – Type system

Introduction

The latest generation of computer software and hardware makes use of numerous new technologies in order to enhance flexibility or performance. Most current systems may be dynamically reconfigured or extended, allow parallelism or use it, and can communicate with other systems. This flexibility, however, induces the multiplication of subsystems and protocols. In turn, this multiplication greatly increases the possibility of bugs, the feasibility of attacks, and the sensitivity to possible breakdown of individual subsystems.

This paper presents a formalism for *resource control* in parallel, distributed, mobile systems called *controlled ambients* (CA). The calculus of CA is based on mobile ambients [5], extends safe ambients [18], and is equipped with a type system to express and verify resource control policies.

In the first section, we present our point of view on the problem of resource control. We provide motivations for using ambient calculi to represent the notion of resource in a distributed setting and claim that a specific calculus should be designed for the purpose of guaranteeing some control over the use of resources. In Sect. 2, we introduce our calculus of controlled ambients and explain why it matches our goals. We then develop in Sect. 3 a type system that uses the specifics of this language to make resource control possible; we prove its correctness (i.e., that it does indeed monitor the acquisition and release of resources) and use it to treat several examples. We discuss some refinements of our type system in Sect. 4 and then sketch a generalization of our type system, allowing us to guarantee properties that are not directly related to resource control (Sect. 5). We discuss related work and conclude in Sect. 6.

This paper is an extended version of [26], that contained no proof. With respect to that paper, the material presented in Sects. 3 and 4 is more detailed here, and the system of Sect. 5 is new.

Work supported by the European project FET – Global Computing.

1 Resource control

For the sake of the present study, we define a resource as an entity that may *at will* be acquired, used, and then released. We thus work with a rather broad notion of resource that encompasses ports, CPUs, computers, or RAM, but not time or, presumably, money. A *resource-controlled system* is a system in which no subsystem will ever require more resources than may be available.

In order to prevent problems such as denial of service attacks, we need a formalism making resource control possible. This formalism should in particular provide means to describe systems in terms of resource availability and resource requirements and should also support the description of concurrent and mobile computations. Lastly, the model should provide some kind of entity that can be regarded as a resource. We now present ambient

calculi and explain why they can be used for these purposes (see also Sect. 6 for a discussion of related works).

Ambient calculi. Ambient calculi are based on the notion of locality: each *ambient* is a site. In turn, any ambient may contain subambients, as well as *processes*, controlling the ambient's behavior through the use of *capabilities*. Capabilities let the structure of ambients evolve: $\mathsf{in}\ m$ and $\mathsf{out}\ m$ let an ambient move (resp. entering ambient m or leaving ambient m), while $\mathsf{open}\ m$ opens ambient m and releases its contents in the current ambient. This is expressed by the following reduction rules of the mobile ambients calculus [5] that describe the basic evolution steps (captured by relation $\longrightarrow$) of terms:

$$\begin{aligned} m[\mathsf{in}\ n.P \mid Q] \mid n[R] &\longrightarrow n[m[P \mid Q] \mid R] \\ &\quad m \textit{ entering ambient } n \\ n[m[\mathsf{out}\ n.P \mid Q] \mid R] &\longrightarrow m[P \mid Q] \mid n[R] \\ &\quad m \textit{ exiting ambient } n \\ \mathsf{open}\ m.P \mid m[Q] &\longrightarrow P \mid Q \textit{ opening ambient } m. \end{aligned}$$

In the terms above, $n[P]$ stands for process P running at site (or, equivalently, *ambient*) n, while $\mid$ denotes parallel composition of terms. Hence for instance, $n[P] \mid n'[P']$ represents two adjacent sites named n and n', with their corresponding contents P and P'. A capability can be used to prefix a term (as for instance in $\mathsf{open}\ m.P$), which results in a process liable to execute this capability when appropriate, as defined by the rules for $\longrightarrow$. When a capability is triggered, it is consumed by the corresponding reduction step. A more precise, formal definition of the syntax and semantics of ambients will be provided below when we present our calculus of controlled ambients.

To draw some analogies with real systems, the in and out primitives can represent the movement of data in a computer or in a network, while open could be used for cleaning memory, for reading data, or for loading programs into memory. As for ambients, they could stand for computers, programs, data, components, etc.

These correspondences open the way to a natural model of resource control, where each site may have a finite (or infinite) quantity of resources of a given category. Resources will be used for data, programs, etc. In other words, each ambient has a given *capacity* and each subambient *uses* a part of this capacity. Basically, *controlling resources means checking the number of* ***direct*** *subambients (according to the amount of resources these are using) that may be present in one ambient at any time.*

Note that we could have chosen different points of view and decided to take into account all subambients at all depths, or possibly only "leaf" ambients. We believe, however, that our approach is more general and flexible, which is the reason we chose it.

An example. We shall use as our main running example a cab protocol: the system consists of one city, n sites, and several cabs and clients. Cabs may be either "anywhere in the city" or in a precise site. Each client may be either in a given site or in a cab. Any client may call a cab, asking for a trip from one site to another site.

In this scenario, several nontrivial properties concerning the interaction among participants and the managment of resources may be expressed. Typically, we stipulate that if a cab is available, one (and only one) cab must come fetch the client and bring her to her destination. Moreover, if we consider the unique passenger seat of a cab as a resource, the system will be resource controlled if each cab contains at most one client at any time.

Figure 1 presents the cab protocol as written in the calculus of mobile ambients.[1] The *city* itself is an ambient, which may contain *site*s and *cab*s. Each site s is in turn an ambient, which may contain *client*s, and ambient movements are used to simulate the movements in the protocol (client entering a cab, cab moving from site to site, etc.). In order for this protocol to work, there must be at least one *cab* and each "*client from to*" declaration must be coherent, i.e., *from* must be the name of the site that hosts the *client* and *to* must be the name of some site.

To call a cab, the client sends a *call* ambient. This ambient then enters a cab, where it gets opened. Opening ambient *call* unleashes process

$$\mathsf{in}\ from.loading[\mathsf{out}\ cab.\mathsf{in}\ client]\,.$$

Therefore, after opening, the cab goes in *from* to meet its client and releases ambient *loading*. Once *loading* has been released, it enters *client*. As soon as the client opens *loading*, she knows that the cab is present and therefore that she may enter it. Consequently, the client enters the cab and releases ambient *trip*, which the cab in turn receives and opens. Once again, a process is unleashed: $\mathsf{out}\ from.\mathsf{in}\ to.unloading[\mathsf{in}\ c]$. This process moves the cab to its destination and releases another synchronization ambient, *unloading*, to tell the client she may get out. When the client receives this ambient, she opens it, leaves, and sends the last synchronization ambient *bye* to the *cab* to tell it it may leave.

Limitations. By examining the code of Fig. 1, one may see that several aspects of this implementation may lead to unwanted behaviors. The most visible flaw is the sending of ambient *bye*: if, for any reason, there are several *cab*s in the site, nothing guarantees that *bye* will reach the right *cab*. And if it does not, it may completely break the system by making one *cab* wait forever for its client to exit, although it already has left, while making the other *cab* leave its destination site with its unwilling *client*. In turn, the *client* may then get out of the cab almost anywhere.

Although this problem is partly due to the way this implementation has been designed, its roots are deeply nested within the calculus of mobile ambients itself. One

[1] As a matter of fact, we are not exactly using the original MA calculus since we work with a recursion operator (rec) instead of replication, which better suits our purposes.

Message emitted by client *client* at site *from* to call a cab

$$call\,from\,client \triangleq call[\mathsf{out}\ client.\mathsf{out}\ from.\mathsf{in}\ cab.\mathsf{in}\ from.loading[\mathsf{out}\ cab.\mathsf{in}\ client]]$$

Instructions given by client *client* going from site *from* to site *to*

$$trip\,from\,to\,c \triangleq trip[\mathsf{out}\ client.\mathsf{out}\ from.\mathsf{in}\ to.unloading[\mathsf{in}\ c]]$$

The client itself, willing to go from *from* to *to*

$$client\,from\,to \triangleq (\nu c)c[call\ from\ c \mid \mathsf{open}\ loading.\mathsf{in}\ cab.trip\ from\ to\ c \mid \mathsf{open}\ unloading.\mathsf{out}\ cab.bye[\mathsf{out}\ c.\mathsf{in}\ cab.\mathsf{out}\ to]]$$

The cab and the city

$$cab \triangleq cab[\mathsf{rec}\ X.\mathsf{open}\ call.\mathsf{open}\ trip.\mathsf{open}\ bye.X]$$

$$city \triangleq city[cab \mid \cdots \mid cab \mid site_1[client\ site_1 site_i \mid client\ site_1\ site_j \mid \cdots] \mid \cdots \mid site_i[\cdots]]$$

Fig. 1. Cab protocol – first attempt

may notice that any malicious ambient may, at any time, enter the cab: in the calculus of mobile ambients, there is no such thing as a filtering of entries/exits. This lack of filtering and accounting is a security threat as well as an obstacle for resource control: for security, since it prevents modeling a system that could check and refuse entry to unwanted mobile code, and for control, since one cannot maintain any information about who is using which resources in a given ambient.

Toward a better control. Difficulties with security and control are largely due to the nature of capabilities in, out, and open. Actually, the way these capabilities are used seems too simplistic: in any real system, arrival or departure of data cannot happen without the consent of the acting subsystem, much less go unnoticed, not to mention that opening a program is not an event that can happen without the operating system having explicitly requested it. In practice, if a program wishes to receive network information, it must first "listen" on some communication port. If a binary file is to be loaded and executed, it must have some executable structure and some given entry point.

A calculus derived from mobile ambients is presented in [18]; in this calculus of *safe ambients*, three *cocapabilities* are introduced, which we will denote by $\overline{\mathsf{SAin}}$, $\overline{\mathsf{SAout}}$, and $\overline{\mathsf{SAopen}}$. When executed in m, capability $\overline{\mathsf{SAin}}\ m$ allows an ambient to enter m (by execution of capability $\mathsf{in}\ m$). Similarly, $\overline{\mathsf{SAout}}\ m$ allows an ambient to leave m using $\mathsf{out}\ m$, while $\overline{\mathsf{SAopen}}\ m$ allows m's parent to open m using $\mathsf{open}\ m$. These cocapabilities make synchronizations more explicit and considerably decrease the risk of security breaches. Getting back to the example above, a rewritten cab may thus easily refuse entry to parasites as long as it is not in any site or while it contains a client. Moreover, a form of resource control is indeed possible since an ambient having no more available resource may refuse entrance of new subambients.

However, in this model, ambients are not always warned when they receive or lose subambients by some kind of side effect: in safe ambients, when the process $h[m[n[\mathsf{out}\ m] \mid \overline{\mathsf{SAout}}\ m]]$ evolves to $h[m[\mathbf{0}] \mid n[\mathbf{0}]]$, h receives n from m but is not made aware of this. Moreover, while $\overline{\mathsf{SAin}}\ m$ serves as a warning for m that it will receive a new subambient, m does not know which one. Since a subambient representing static data and another one modeling some internal message will not occupy the same amount of resources, this model is probably not sufficient for our purposes.

Guan et al. [13] offer an alternative to these cocapabilities in order to further enhance systems' robustness: in this formalism, $\overline{\mathsf{in}}\ m$ does not allow *entering* m but rather *m to enter*. This approach solves one of our problems: identifying incoming data. Controlled ambients, which will be presented in the next section, may be considered as a development of [13] toward even greater robustness as well as resource control. Let us also mention [19], where a different mechanism for the $\overline{\mathsf{SAout}}$ cocapability w.r.t. [18] is introduced. Our proposal subsumes the solutions of [19] and [18].

Embedding resource control. In Sect. 3, we equip our language with a type system for resource control. Basically, the type of an ambient carries two pieces of information:

- Its *capacity* – how many resources the ambient offers to its subambients;
- Its *weight* – how many resources it requires from its parent ambients.

The type system allows one to statically divide the available resources between parallel processes and check that resources will be controlled along movements and openings of ambients.

2 The language of controlled ambients

2.1 Syntax and semantics

In CA, each movement is subject to a three-way synchronization between the moving ambient, the ambient welcoming a new subambient, and the ambient letting a sub-

ambient go. As for the opening of an ambient, it is triggered by a synchronization between the opener and the ambient being opened. These forms of synchronization are somewhat reminiscent of early versions of Seal [28]. Interaction is handled using *cocapabilities*: $\overline{\mathsf{in}_\uparrow}$, $\overline{\mathsf{out}_\uparrow}$, $\overline{\mathsf{in}_\downarrow}$, $\overline{\mathsf{out}_\downarrow}$, and $\overline{\mathsf{open}}$.

$\overline{\mathsf{in}_\uparrow}\ m$	the *up coentry*, welcomes m coming from a subambient;
$\overline{\mathsf{in}_\downarrow}\ m$	the *down coentry*, welcomes m coming from the parent ambient;
$\overline{\mathsf{out}_\uparrow}\ m$	the *up coexit*, allows m to leave the current ambient by exiting it;
$\overline{\mathsf{out}_\downarrow}\ m$	the *down coexit*, allows m to leave by entering a subambient;
$\overline{\mathsf{open}}\ \{m, h\}$	the *coopening*, allows the parent ambient h to open the current ambient m.

Note that the direction tags $\uparrow$ and $\downarrow$ are not strictly necessary for resource control. We added them since we found they eased the task of specification in mobile ambients. We will return to the use of these annotations in Sect. 2.3.

The syntax of controlled ambients is presented in Fig. 2. Suppose we have two infinite sets of term variables, denoted by uppercase letters (X, Y), and of names, denoted by lowercase letters $(m, n, h, x, \dots)$. Name binders (input and restriction) are decorated with some type of information that will be made explicit in the next section. While several proposals for mobile ambient calculi use replication, infinite behavior is represented using recursion in CA. This is mostly due to the fact that recursion allows for an easier specification of loops, especially in the context of resource consumption. Note also that, compared to the original calculus of mobile ambients, we restrict ourselves to communication of ambient names only and do not handle communicated capabilities.

The null process $\mathbf{0}$ does nothing. Process $M.P$ is ready to execute M, then to proceed with P. $P \mid Q$ is the parallel composition of P and Q. $m[P]$ is the definition of an ambient with name m and contents P. The process $(\nu n : A)P$ creates a new, private name n, then behaves as P. The recursive construct $\mathsf{rec}\ X.P$ behaves like P in which occurrences of X have been replaced by $\mathsf{rec}\ X.P$. Process $(n : A)Q$ is ready to accept a message, then to proceed with Q with the actual message replacing the formal parameter n. $\langle m \rangle$ is the asynchronous emission of a message m. In most cases, we omit the terminal $\mathbf{0}$ process. We say that a process is *prefixed* if it is of the form $M.P$, $\mathsf{rec}\ X.P$ or $(x : A)P$.

The operational semantics of CA is defined in two steps. Structural congruence, written $\equiv$, is defined as the least congruence relation that contains α-equivalence (capture-free renaming of bound names) and satisfying the laws of Fig. 3. Two processes are deemed equal by $\equiv$ when they only differ by some elementary syntactical manipulations. Reduction $(\longrightarrow)$ is defined by the rules of Fig. 4. The first three rules specify movement and opening in CA as described informally above: note the three-way synchronization for the movement rules and the role of the direction tags in cocapabilities. The other reduction rules are standard: they describe communication in ambients and recursion unfolding and express the fact that reduction can occur anywhere in nonprefixed contexts and that $\longrightarrow$ is defined modulo $\equiv$. We let $\longrightarrow^*$ stand for the reflexive transitive closure of $\longrightarrow$.

2.2 Examples of CA programming

We now provide a few examples to illustrate the use of controlled ambients. We omit in the examples given below type annotations in restrictions; these will be made explicit in the next section.

P	::=	$\mathbf{0}$	null process	M	::=	$\mathsf{in}\ m$	enter m
	|	$M.P$	capability		|	$\mathsf{out}\ m$	leave m
	|	$m[P]$	ambient		|	$\mathsf{open}\ m$	open m
	|	$P_1 \mid P_2$	parallel composition		|	$\overline{\mathsf{in}_\uparrow}\ m$	m may enter upwards
	|	$(\nu n : A)P$	restriction		|	$\overline{\mathsf{in}_\downarrow}\ m$	m may enter downwards
	|	$\mathsf{rec}\ X.P$	recursion		|	$\overline{\mathsf{out}_\uparrow}\ m$	m may leave upwards
	|	X	process variable		|	$\overline{\mathsf{out}_\downarrow}\ m$	m may leave downwards
	|	$(n : A)P$	abstraction		|	$\overline{\mathsf{open}}\ \{m, h\}$	h may open m
	|	$\langle m \rangle$	message emission				

Fig. 2. Controlled ambients – syntax

$$P \equiv P \mid \mathbf{0} \qquad P \mid Q \equiv Q \mid P \qquad P \mid (Q \mid R) \equiv (P \mid Q) \mid R$$
$$(\nu n : A)\,\mathbf{0} \equiv \mathbf{0} \qquad (\nu n : A)(\nu m : B)\,P \equiv (\nu m : B)(\nu n : A)\,P$$
$$(\nu n : A)\,(P \mid Q) \equiv ((\nu n : A)\,P) \mid Q \quad \text{if } n \notin fn(Q)$$
$$(\nu n : A)\,m[P] \equiv m[(\nu n : A)\,P] \quad \text{if } n \neq m$$

Fig. 3. Controlled ambients – structural congruence

$$m[\mathsf{in}\ n.P \mid Q] \mid n[\overline{\mathsf{in}_\downarrow}\ m.R \mid S] \mid \overline{\mathsf{out}_\downarrow}\ m.T \longrightarrow n[m[P \mid Q] \mid R \mid S] \mid T$$

$$n[m[\mathsf{out}\ n.P \mid Q] \mid \overline{\mathsf{out}_\uparrow}\ m.R \mid S] \mid \overline{\mathsf{in}_\uparrow}\ m.T \longrightarrow m[P \mid Q] \mid n[R \mid S] \mid T$$

$$h[\mathsf{open}\ m.P \mid Q \mid m[\overline{\mathsf{open}}\ \{m,h\}.R \mid S]] \longrightarrow h[P \mid Q \mid R \mid S]$$

$$\langle n \rangle \mid (x:A)P \longrightarrow P\{x \leftarrow n\}$$

$$\mathsf{rec}\ X.P \longrightarrow P\{X \leftarrow \mathsf{rec}\ X.P\}$$

$$\frac{P \longrightarrow Q}{(\nu n:A)\,P \longrightarrow (\nu n:A)\,Q} \qquad \frac{P \longrightarrow Q}{R \mid P \longrightarrow R \mid Q} \qquad \frac{P \longrightarrow Q}{n[P] \longrightarrow n[Q]}$$

$$\frac{P \equiv Q \quad Q \longrightarrow R \quad R \equiv S}{P \longrightarrow S}$$

Fig. 4. Controlled ambients – reduction

The examples we discuss focus on the issue of resource control. In particular, we do not address here questions related to interference-freeness or to the behavioral properties of the processes we introduce, as in, e.g., [3, 18].

Renaming. Since movements in controlled ambients require full knowledge about the name of moving ambients (also in cocapabilites, which is not the case in safe ambients), renaming often turns out to be useful in order to comply with some protocols. One may write the renaming of ambient a to b as follows:

$$a\,be\,b.P \triangleq b[\mathsf{out}\ a.\overline{\mathsf{in}_\downarrow}\ a.\mathsf{open}\ a] \mid \overline{\mathsf{out}_\uparrow}\ b.\mathsf{in}\ b.\overline{\mathsf{open}}\ \{a,b\}.P\,.$$

We then have $\overline{\mathsf{in}_\uparrow}\ b.\overline{\mathsf{out}_\downarrow}\ a \mid a[a\ be\ b.P] \longrightarrow^* b[P]$. This important example is also characteristic of controlled ambients since $\overline{\mathsf{in}_\uparrow}\ b.\overline{\mathsf{out}_\downarrow}\ a$ illustrates a particular programming discipline: a's parent ambient must accept the replacement of a by b. This means that, at any time, the father ambient knows its own contents, that is, both the number of subambients and their names.

Safe ambient cocapabilities. As mentioned above, safe ambients [18] introduce another kind of cocapabilities, similar to ours, though weaker.

We concentrate here on the $\overline{\mathsf{SAin}}$ cocapability (the case of $\overline{\mathsf{SAout}}$ being symmetrical). Its semantics is defined by

$$a[\mathsf{in}\ b.P \mid Q] \mid b[\overline{\mathsf{SAin}}\ b.R \mid S] \longrightarrow b[R \mid S \mid a[P \mid Q]]\,.$$

By carrying on the idea behind renaming, we can approximate the specifics of this cocapability in CA. In other words, $a[\mathsf{in}\ b.P \mid Q] \mid b[\overline{\mathsf{SAin}}\ b.R \mid S]$ may be written

$$\begin{aligned}(\nu m,n)\big(a\big[\,&\overline{\mathsf{out}_\uparrow}\ m.\mathsf{in}\ b.(P \mid n[\mathsf{out}\ a.\overline{\mathsf{open}}\ \{n,b\}] \mid \overline{\mathsf{out}_\uparrow}\ n)\\ &\mid Q \mid m[\mathsf{out}\ a.\mathsf{in}\ b.\overline{\mathsf{open}}\ \{m,b\}.\overline{\mathsf{in}_\downarrow}\ a]\,\big]\\ \mid\ & b[\overline{\mathsf{in}_\downarrow}\ m.\mathsf{open}\ m.\overline{\mathsf{in}_\uparrow}\ n.\mathsf{open}\ n.R \mid S]\\ &\mid \overline{\mathsf{in}_\uparrow}\ m.\overline{\mathsf{out}_\downarrow}\ m.\overline{\mathsf{out}_\downarrow}\ a\big)\,.\end{aligned}$$

As specified, this expression reduces to $b[R \mid S \mid a[P \mid Q]]$. We use here two auxiliary ambients m and n to simulate the $\overline{\mathsf{SAin}}$ cocapability. Initially, ambient b does not know name a, so the role of m is to bring this knowledge into b in order for it to be able to execute the CA cocapability $\overline{\mathsf{in}_\downarrow}\ a$ (which is carried in m). Ambient n is used as a synchronization device in order to block the execution of R as long as a is not inside b. As was the case for renaming, the father must accept the transaction with $\overline{\mathsf{in}_\uparrow}\ m.\overline{\mathsf{out}_\downarrow}\ m.\overline{\mathsf{out}_\downarrow}\ a$. This entails in particular the father ambient's being aware of the presence of a.

Firewall. We revisit the firewall example of [5] and consider a system f protected by a firewall. Only agents knowing the password g are allowed in f. This may be modeled as

$$Agent\ P\ Q \triangleq agent[\mathsf{in}\ g.\overline{\mathsf{in}_\downarrow}\ entered.\mathsf{open}\ entered.P \mid Q]$$

$$\begin{aligned}System \quad \triangleq\ & (\nu f)\,f\big[\mathsf{rec}\ X.\big(\\ &\quad g[\mathsf{out}\ f.\overline{\mathsf{in}_\downarrow}\ agent.\mathsf{in}\ f.\overline{\mathsf{open}}\ \{g,f\}]\\ &\mid \overline{\mathsf{out}_\uparrow}\ g.\overline{\mathsf{in}_\downarrow}\ g.\mathsf{open}\ g.\\ &\qquad (entered[\mathsf{in}\ agent.\overline{\mathsf{open}}\ \{entered, agent\}]\\ &\qquad\qquad \mid \overline{\mathsf{out}_\uparrow}\ entered.X)\\ &\big)\big]\ \mid\ \mathsf{rec}\ Y.\overline{\mathsf{in}_\uparrow}\ g.\overline{\mathsf{out}_\downarrow}\ agent.\overline{\mathsf{out}_\downarrow}\ g.Y\,.\end{aligned}$$

This specification behaves as follows: *system* receives *agent* and then recovers its original structure thanks to rec. The structure of g guarantees that, at any time, g may only contain one *agent*. On the other hand, *system* may contain any number of *agent*s. This system implements two authentications: first, the agent must be named *agent* – it will not enter f by accident. Second, it must know the password. Note that this is not the firewall described in the original paper on mobile ambients [5], which relied on the secrecy of three keys. This version uses only one key and takes advantage of the synchronization mechanism to execute correctly.

Cab. Figure 5 presents a CA version of the cab protocol from Sect. 1. We do not give definitions for the city or for

Message emitted by client

$call\ from \triangleq call[\mathsf{out}\ client.\mathsf{out}\ from.$
$\quad \mathsf{in}\ cab.\overline{\mathsf{open}}\ \{call, cab\}.\mathsf{in}\ from.\overline{\mathsf{in}_\downarrow}\ client]$

Instructions given by client

$trip\ from\ to \triangleq$
$\quad trip[\mathsf{out}\ client.\overline{\mathsf{open}}\ \{trip, cab\}.\mathsf{out}\ from.$
$\quad\quad \mathsf{in}\ to.arrived[\overline{\mathsf{open}}\ \{arrived, cab\}.end[\overline{\mathsf{open}}\ \{end, cab\}.\mathsf{out}\ to]]]$

The client

$client\ from\ to \triangleq client[call\ from \mid \overline{\mathsf{out}_\uparrow}\ call.\mathsf{in}\ cab.trip\ from\ to$
$\quad \mid \overline{\mathsf{out}_\uparrow}\ trip.\mathsf{out}\ cab]$

The cab

$ONE\_TRIP.P \triangleq \overline{\mathsf{in}_\downarrow}\ call.\mathsf{open}\ call.\overline{\mathsf{in}_\uparrow}\ trip.\mathsf{open}\ trip.$
$\quad \mathsf{open}\ arrived.\overline{\mathsf{out}_\uparrow}\ client.\mathsf{open}\ end.P$

$cab \triangleq cab[\mathsf{rec}\ X.\overline{\mathsf{in}_\downarrow}\ call.\mathsf{open}\ call.\overline{\mathsf{in}_\uparrow}\ trip.\mathsf{open}\ trip.\mathsf{open}\ arrived.$
$\quad \overline{\mathsf{out}_\uparrow}\ client.\mathsf{open}\ end.X]$

Fig. 5. Cab protocol – CA-style (see Fig. 1)

the sites, which only need to contain all movement authorizations, in addition to *client*s and *cab*s. ONE_TRIP is a macro. Using cocapabilities, synchronizations in CA are easier than in both mobile ambients and atomic ambients. Additionally, the system is not subject to the interferences we have presented: only *client*s may enter the cab, not just any "parasite" ambient that happens to contain capability $\mathsf{in}\ cab$. Similarly, sites only welcome *client*s, *cab*s, and *call*s.

Note that, in this version, all clients must be named *client* in order to enter a *cab*. One could use renaming or the approximation of $\overline{\mathsf{SAin}}$ to relax this constraint (see above).

Additionally, controlled ambients permit the control of resources such as available space in cabs. As opposed to the mobile ambients version, we can easily check that the cab may contain at most *only one passenger* and possibly an auxiliary ambient *call*, *trip*, *arrived*, or *end*. These properties will be expressed formally using our type system in Sect. 3. Note that we could also have expressed the cab protocol in safe ambients, also avoiding grave interferences. In this example, the main benefit of controlled ambients is related to the properties that the type system allows us to establish.

2.3 Benefits

We believe that the formalism of controlled ambients is more reasonable than mobile ambients, safe ambients, or robust ambients. More reasonable insofar as the implementation of movements in ambient calculi suggests this kind of three-way synchronization. To illustrate our claim, let us consider the following transition in mobile ambients:

$$h[m[\mathsf{in}\ n] \mid n[\mathbf{0}]] \longrightarrow h[n[m[\mathbf{0}]]]\,.$$

As shown in [10, 22], a practical implementation of this rule requires that h must be aware of the presence of n, no matter how n may have entered h. More generally, the execution of this movement will involve a synchronization between n (who is actually present), m (who looks for n), and h (who knows about the presence of m and n). Similarly, the opening of ambient m by ambient h requires some complex synchronization between m and h in order to recover all processes and subambients of m within h and update the presence registers of h. A prototype implementation has been developed [11] in order to experiment with CA-like synchronization.

Controlled ambients are also more realistic as modeling tools. When a system receives information, it must be by some action of his: the operating system "listens" on a device, the configuration server waits for a request by "listening" on some given TCP/IP port, etc. Unfortunately, this listening behavior is not rendered at all by mobile ambients and only in half of the cases by safe ambients. Similarly, a system is liable to request several kinds of information and to sort them according to their origin: the OS is able to differentiate data read on a disk from data read on the network or on the keyboard, while software may listen on several communication ports, for example. We can easily model such phenomena in CA and, if necessary, take into account situations where some part of the system (such as the network connection itself) accepts data without listening explicitly for them, using renaming and infinite loops of cocapabilities.

Additionally, the use of two cocapabilities, one triggering a continuation in the source space, the other one triggering a continuation in the target space, is very important for dynamic resource control. Using this mechanism, it is easy to write systems in which individual components may react to their resources being exhausted or replenished. Cocapabilities also serve as a basis for the

development of our method for static resource control, as will be explained in the next section.

3 Typing controlled ambients

This section is devoted to the presentation of a type system for resource control in controlled ambients. We first describe the system and its properties and then show the kind of information it is liable to check using some examples.

3.1 Type system

Type judgments. The grammar for types is given in Fig. 6 and includes entries for the types of ambients, processes, and messages ($\overline{\mathbb{N}}$ stands for $\mathbb{N} \cup \{\infty\}$).

Typing environments, denoted by Γ, are lists of associations of the form $x : A$ (for ambient names) or $X : U$ (for process variables). We write $\Gamma(x) = A$ (resp. $\Gamma(X) = U$) to represent the fact that environment Γ associates A (resp. U) with x (resp. X). $\Gamma, x : A$ stands for the extension of Γ with the association $x : A$, possibly hiding some previous binding for x (and similarly for $\Gamma, X : U$).

The typing judgment for ambient names is of the form

$$\Gamma \vdash n : \mathrm{CA}_{\mathrm{AM}}(s, e)[T]$$

and expresses the fact that under assumptions Γ, n is the name of an ambient of *capacity* s and *weight* e and within which messages carrying information of type T may be exchanged. The capacity s represents the amount of space (or of *resources*) available for subambients within n, while e is the number of resources this ambient is occupying in its surrounding ambient. Note that, while an ambient may have an infinite capacity ($s = \infty$), it cannot manipulate infinitely many resources ($e < \infty$). Moreover, if we decide to impose $e \geq s$ in ambient types, we may develop an analysis close to what is done in [7], where the weight of an ambient takes into account the weight of all its subambients, *at any depth.* The type T for messages captures the kind of names being exchanged within n, like Cardelli and Gordon's *topics of conversation* [6], augmented with a piece of information t that represents a higher bound on the effect of exchanging messages within n (we shall return to this below).

The typing judgment for processes is written

$$\Gamma \vdash P : \mathrm{CA}_{\mathrm{PR}}(t)[T]\,,$$

meaning that, according to Γ, P is a process that may use up to t resources and take part in conversations (that is, emit and receive messages) having type T.

A	::=	$\mathrm{CA}_{\mathrm{AM}}(s, e)[T]$	$s \in \overline{\mathbb{N}}, e \in \mathbb{N}$	ambient types
U	::=	$\mathrm{CA}_{\mathrm{PR}}(t)[T]$	$t \in \overline{\mathbb{N}}$	process types
T	::=	Ssh		message types
	|	t, A	$t \in \overline{\mathbb{N}}$	

Fig. 6. Types

Typing rules. The rules defining the typing judgments are given on Fig. 7. We now comment on them. While typing (subjective) movements has no effect from the point of view of resources (rules T-IN and T-OUT), the rules T-COIN and T-COOUT, for the cocapabilities (where δ denotes a direction tag that can be $\uparrow$ or $\downarrow$), express the meaning of t in $\mathrm{CA}_{\mathrm{PR}}(t)[T]$, according to the weight e of the moving ambient. Note that the number t of resources allocated to the process must remain positive after decreasing (rule T-COOUT). This is made possible by the subtyping property of the system (Lemma 1), together with rules T-NIL, T-AMB, ... , which allow one to allocate any number of resources to an inert process (inert from the point of view of the current ambient). This mechanism can be used, for example, to derive a typing for a process of the form $\overline{\mathsf{out}_\uparrow}\, n.\mathbf{0}$. Note also that the peripheral condition $a \leq s$ in rule T-AMB expresses conformity with the capacity of the ambient.

When opening an ambient, we release the resources it had acquired (e), but at the same time we have to provide at least as many resources as its original capacity (s). The $\overline{\mathsf{open}}$ capability plays no role from the point of view of resource control, as illustrated by rule T-COOPEN (note, still, that message types in the opening ambient and in the type of R are unified using this rule). We shall present in Sect. 4 a richer system where a more precise typing of opening (and co-opening) permits a better control.

We now explain the typing rules for communication. Since reception of a message can trigger a process that will necessitate a certain amount of resources, we attach to the type of an ambient the maximum amount of resources needed by a receiving process running within it: this is information t in an ambient's topic of conversation. Put differently, messages are decorated with an integer representing at least as many resources as needed by the processes they are liable to trigger: we are thus somehow measuring an effect in this case. Note that our approach is based on the idea that one emission typically corresponds to several receptions. The dual point of view could have been adopted by putting in correspondence one reception and several concurrent emissions. Our experience in writing examples suggests that the first choice is more useful.

Finally, rule T-REC expresses the fact that a recursively defined process should run "in constant space": as required by the premise, each time a recursive call is triggered (by X), the number t of allocated resources is the same as the number of resources allocated to the whole recursive process P.

3.2 Static resource control

3.2.1 Main properties

We start with some technical properties of typing derivations.

$$\text{T-NAME}\ \frac{\Gamma(n) = A}{\Gamma \vdash n : A} \qquad \text{T-VAR}\ \frac{\Gamma(X) = \text{CApr}(t)[T]}{\Gamma \vdash X : \text{CApr}(t')[T]}\ t' \leq t$$

$$\text{T-REC}\ \frac{\Gamma, X : \text{CApr}(t)[T] \vdash P : \text{CApr}(t)[T]}{\Gamma \vdash \mathit{rec}\ X.P : \text{CApr}(t')[T]}\ t' \geq t$$

$$\text{T-IN}\ \frac{\Gamma \vdash P : \text{CApr}(t)[T]}{\Gamma \vdash \mathit{in}\ m.P : \text{CApr}(t)[T]} \qquad \text{T-OUT}\ \frac{\Gamma \vdash P : \text{CApr}(t)[T]}{\Gamma \vdash \mathit{out}\ m.P : \text{CApr}(t)[T]}$$

$$\text{T-COIN}\ \frac{\Gamma \vdash P : \text{CApr}(t)[T] \qquad \Gamma \vdash m : \text{CAam}(s,e)[T']}{\Gamma \vdash \overline{\mathit{in}_\delta}\ m.P : \text{CApr}(t+e)[T]}$$

$$\text{T-COOUT}\ \frac{\Gamma \vdash P : \text{CApr}(t)[T] \qquad \Gamma \vdash m : \text{CAam}(s,e)[T']}{\Gamma \vdash \overline{\mathit{out}_\delta}\ m.P : \text{CApr}(t-e)[T]}\ t \geq e$$

$$\text{T-OPEN}\ \frac{\Gamma \vdash m : \text{CAam}(s,e)[T] \qquad \Gamma \vdash P : \text{CApr}(t)[T]}{\Gamma \vdash \mathit{open}\ m.P : \text{CApr}(t-e+s)[T]}\ t-e+s \geq 0$$

$$\text{T-COOPEN}\ \frac{\Gamma \vdash m : \text{CAam}(s,e)[T] \qquad \Gamma \vdash R : \text{CApr}(t)[T]}{\Gamma \vdash \overline{\mathit{open}}\ \{m,h\}.R : \text{CApr}(t)[T]}$$

$$\text{T-NIL}\ \Gamma \vdash \mathbf{0} : \text{U} \qquad \text{T-AMB}\ \frac{\Gamma \vdash m : \text{CAam}(s,e)[T] \qquad \Gamma \vdash P : \text{CApr}(a)[T]}{\Gamma \vdash m[P] : \text{CApr}(t)[T']}\ \begin{cases} a \leq s \\ e \leq t \end{cases}$$

$$\text{T-RES}\ \frac{\Gamma, n : A \vdash P : U}{\Gamma \vdash (\nu n : A)P : U} \qquad \text{T-PAR}\ \frac{\Gamma \vdash P : \text{CApr}(t)[T] \qquad \Gamma \vdash Q : \text{CApr}(t')[T]}{\Gamma \vdash P \mid Q : \text{CApr}(t+t')[T]}$$

$$\text{T-SND}\ \frac{\Gamma \vdash m : A}{\Gamma \vdash \langle m \rangle : \text{CApr}(t')[t, A]}\ t' \geq t \qquad \text{T-RCV}\ \frac{\Gamma, x : A \vdash P : \text{CApr}(t)[t, A]}{\Gamma \vdash (x : A)P : \text{CApr}(t')[t, A]}$$

Fig. 7. Typing rules

Lemma 1 (Subtyping). *Let P be a process and Γ an environment such that $\Gamma \vdash P : \text{CApr}(t)[T]$ for some t. Then for any $t' \geq t$, $\Gamma \vdash P : \text{CApr}(t')[T]$.*

Proof. By induction on the derivation of $\Gamma \vdash P : \text{CApr}(t)[T]$.

- *Cases* CA-NIL, CA-AMB, CA-REC, CA-VAR-PROC, CA-SEND, *and* CA-RECEIVE*:*
  In all these cases, the parameter t is free with a lower bound. As a consequence, we can increase it as much as we want.
- *Cases* CA-RES, CA-PAR, CA-IN, CA-OUT, CA-COIN, CA-COOUT, CA-OPEN, *and* CA-COOPEN*:*
  All these cases only require a simple induction step.

□

Corollary 1 (Minimal typing). *If a process P is typeable in Γ with a conversation topic type T, then there is a minimal $t \in \overline{\mathbb{N}}$ such that $\Gamma \vdash P : \text{CApr}(t)[T]$.*

Note that the minimal parameter t can be different for each possible value T (see, for example, rule T-SND).

Additionally, we have the following properties. The proofs of these lemmas are standard, and not given.

Lemma 2 (Strengthening, names). *If $\Gamma, n : A \vdash P : U$ and $n \notin fn(P)$, then $\Gamma \vdash P : U$.*

Lemma 3 (Strengthening, processes). *If $\Gamma, X : U' \vdash P : U$ and $X \notin fv(P)$, then $\Gamma \vdash P : U$.*

Lemma 4 (Weakening, names). *If $\Gamma \vdash P : U$ and $n \notin fn(P)$, then $\Gamma, n : A \vdash P : U$.*

3.2.2 Resource usage

Let us now examine resource control. In order to be able to state the properties we are interested in, we extend the notion of weight, which has been used for ambients, to processes, by introducing the notion of resource usage, together with a natural terminology:

Definition 1 (Resource policy and resource usage). *We call* resource policy *a typing context. Given a resource policy Γ, we define the* resource usage *of a process P according to Γ, written $Res_\Gamma(P)$, as follows:*

- *If $\Gamma(a) = \text{CAam}(s,e)[T]$, then $Res_\Gamma(a[\,]P) = e$;*
- *$Res_\Gamma(P_1 \mid P_2) = Res_\Gamma(P_1) + Res_\Gamma(P_2)$;*
- *$Res_\Gamma((\nu n : A)\, P) = Res_{\Gamma, n:A}(P)$;*
- *In all other cases, $Res_\Gamma(P) = 0$.*

Note in particular that, according to this definition, prefixed terms (capabilities, reception, recursion) do not contribute to a process's *current* resource usage (accordingly, their resource usage is equal to 0).

We now define formally what it means for a process to respect a given resource policy.

Definition 2 (Resource policy compliance). *Given a resource policy Γ, we define the judgment $\Gamma \models P$ (pronounced "P complies with Γ"), as follows:*

- *$\Gamma \models n[P]$ iff $\Gamma \models P$ and $Res_\Gamma(P) \leq s$, where capacity s is given by $\Gamma(n) = \text{CAam}(s,e)[T]$;*

– $\Gamma \models P_1|P_2$ *iff* $\Gamma \models P_1$ *and* $\Gamma \models P_2$;
– $\Gamma \models (\nu n : A)\, P$ *iff* $\Gamma, n : A \models P$;
– *In all other cases,* $\Gamma \models P$.

Intuitively, the judgment $\Gamma \models P$ means that any ambient occurring in P contains no more subambients (in relation to the corresponding weights) than what its capacity allows. The typing rules we have introduced ensure that a typeable term complies with a resource policy:

Proposition 1 (Typeable terms comply with resource policies). For any process P, resource policy Γ, and process type U, if $\Gamma \vdash P : U$, then $\Gamma \models P$.

In order to prove this property, we need the following lemma:

Lemma 5. *If* $\Gamma \vdash P : \text{CApr}(t)[T]$, *then* $Res_\Gamma(P) \leq t$.

Proof. By induction on the derivation of $\Gamma \vdash P : \text{CApr}(t)[T]$.

- *Case* CA-AMB We have $\Gamma \vdash n[P] : \text{CApr}(t)[T]$ with $e \leq t$, where e is the weight of n in Γ. Then, $Res_\Gamma(n[P]) = e \leq t$.
- *Case* CA-PAR $\Gamma \vdash P|Q : \text{CApr}(t_P + t_Q)[T]$ must have been derived from $\Gamma \vdash P : \text{CApr}(t_P)[T]$ and $\Gamma \vdash Q : \text{CApr}(t_Q)[T]$. By the induction hypothesis, $Res_\Gamma(P) \leq t_P$ and $Res_\Gamma(Q) \leq t_Q$. Then, $Res_\Gamma(P|Q) = Res_\Gamma(P) + Res_\Gamma(Q) \leq t_P + t_Q$.
- *Case* CA-RES $\Gamma \vdash (\nu n : A)P : \text{CApr}(t)[T]$ must have been derived from $\Gamma, n : A \vdash P : \text{CApr}(t)[T]$. Then, by the induction hypothesis, we have: $Res_\Gamma((\nu n : A)P) = Res_{\Gamma, n:A}(P) \leq t$.
- *Other cases* For all other cases, $Res_\Gamma(P) = 0 \leq t$.

□

Proof (of Proposition 1). By induction on the structure of P.

- *Case* $P|Q$ Since $P|Q$ is typeable in Γ, so are P and Q. By the induction hypothesis, $\Gamma \models P$ and $\Gamma \models Q$. Then, $\Gamma \models P|Q$.
- *Case* $(\nu n : A)P$ Since $(\nu n : A)P$ is typeable in Γ, P must be tyepable in $\Gamma, n : A$. By the induction hypothesis, $\Gamma, n : A \models P$. Then, $\Gamma \models (\nu n : A)P$.
- *Case* $n[P]$ This is the main case; we must check two properties.
  - First, P should respect Γ. Since $n[P]$ is typeable in Γ, P is also typeable. We conclude $\Gamma \models P$ by the induction hypothesis.
  - Second, the resources of n should be locally controlled according to Γ, that is, $Res_\Gamma(P) \leq s$, where s is the capacity of n in Γ. Since $n[P]$ is typeable in Γ, we have from rule CA-AMB: $\Gamma(n) = \text{CAam}(s, e)[T]$ and $\Gamma \vdash P : \text{CApr}(t)[T]$, with the condition $t \leq s$. By Lemma 5, we can conclude: $Res_\Gamma(P) \leq t \leq s$.
- *Other cases* In all other cases, we have nothing to check.

□

The following theorem states that typability is preserved by the operational semantics of controlled ambients:

Theorem 1 (Subject reduction). *For any processes* P, Q, *resource policy* Γ, *and type* U, *if* $\Gamma \vdash P : U$ *and* $P \longrightarrow Q$, *then* $\Gamma \vdash Q : U$.

Proof. The proof of this result is given in the appendix.

As a direct consequence of Proposition 1 and Theorem 1, we obtain our main result:

Theorem 2 (Resource control). *Consider a resource policy* Γ *and a process* P *such that* $\Gamma \vdash P : U$ *for some* U. *Then for any* Q *such that* $P \longrightarrow^* Q$, *it holds that* $\Gamma \models Q$.

3.3 Examples

We now revisit some examples from Sect. 2.2 and explain how they can be typed. In each case, we exhibit a resource policy (i.e., a typing context Γ) that captures a property we wish to guarantee and describe the weight and capacity associated with every ambient in order to do so.

Renaming. As already established, one possible expression of renaming is

$$a \textit{ be } b.P \triangleq b[\mathsf{out}\ a.\overline{\mathsf{in}_\downarrow}\ a.\mathsf{open}\ a] \mid \overline{\mathsf{out}_\uparrow}\ b.\mathsf{in}\ b.\overline{\mathsf{open}}\ \{a, b\}.P.$$

Let us assume that there exists a typing environment Γ and a conversation type T such that

$$\begin{cases} \Gamma(a) &= \text{CAam}(s, e)[T] \\ \Gamma(b) &= \text{CAam}(s, e)[T] \\ \Gamma &\vdash P : \text{CApr}(s)[T] \\ s &\geq e, \end{cases}$$

where T is the conversation type of the ambient that gets renamed.

We may then build the derivation seen in Fig. 8. It proves that, according to the typing environment Γ, a *be* $b.P$ may be typed. As a trivial corollary, $a[a \textit{ be } b.P]$ may also be typed.

We can actually slightly relax the conditions on types. One can show that the least set of conditions to type $a[a \textit{ be } b.P]$ is

$$\begin{cases} \Gamma(a) &= \text{CAam}(s_a, e_a)[T] & t_p \leq s_a \quad e_b \leq s_a \\ \Gamma(b) &= \text{CAam}(s_b, e_b)[T] & s_a \leq s_b \quad e_a \leq s_b\,. \\ \Gamma &\vdash P : \text{CApr}(t_p)[T] \end{cases}$$

Firewall. Similarly, the firewall in controlled ambients, as defined in Sect. 2.2, can be typed in a context Γ such that:

			Typing $b[\mathsf{out}\ a.\overline{\mathsf{in}_\downarrow}\ a.\mathsf{open}\ a.\mathbf{0}]$
Γ	$\vdash \mathbf{0}:$	$\mathrm{CApr}(0)[T]$	by T-NIL
	we have	$e \leq s$	by hypothesis
$\Rightarrow \Gamma$	$\vdash \mathsf{open}\ a.\mathbf{0}:$	$\mathrm{CApr}(s-e)[T]$	by T-OPEN
$\Rightarrow \Gamma$	$\vdash \overline{\mathsf{in}_\downarrow}\ a.\mathsf{open}\ a.\mathbf{0}:$	$\mathrm{CApr}(s)[T]$	by T-COIN
$\Rightarrow \Gamma$	$\vdash \mathsf{out}\ a.\overline{\mathsf{in}_\downarrow}\ a.\mathsf{open}\ a.\mathbf{0}:$	$\mathrm{CApr}(s)[T]$	by T-OUT
$\Rightarrow \Gamma$	$\vdash b[\mathsf{out}\ a.\overline{\mathsf{in}_\downarrow}\ a.\mathsf{open}\ a.\mathbf{0}]:$	$\mathrm{CApr}(e)[T]$	by T-AMB
			Typing $\overline{\mathsf{out}_\uparrow}\ b.\mathsf{in}\ b.\overline{\mathsf{open}}\ \{a,b\}.P$
Γ	$\vdash P:$	$\mathrm{CApr}(s)[T]$	by hypothesis
$\Rightarrow \Gamma$	$\vdash \overline{\mathsf{open}}\ \{a,b\}.P:$	$\mathrm{CApr}(s)[T]$	by T-COOPEN
$\Rightarrow \Gamma$	$\vdash \mathsf{in}\ b.\overline{\mathsf{open}}\ \{a,b\}.P:$	$\mathrm{CApr}(s)[T]$	by T-IN
	we have	$e \leq s$	by hypothesis
$\Rightarrow \Gamma$	$\vdash \overline{\mathsf{out}_\uparrow}\ b.\mathsf{in}\ b.\overline{\mathsf{open}}\ \{a,b\}.P:$	$\mathrm{CApr}(s-e)[T]$	by T-COOUT
			Typing $a\ be\ b.P$
Γ	$\vdash b[\mathsf{out}\ a.\overline{\mathsf{in}_\downarrow}\ a.\mathsf{open}\ a.\mathbf{0}]:$	$\mathrm{CApr}(e)[T]$	see above
Γ	$\vdash \overline{\mathsf{out}_\uparrow}\ b.\mathsf{in}\ b.\overline{\mathsf{open}}\ \{a,b\}.P:$	$\mathrm{CApr}(s-e)[T]$	see above
$\Rightarrow \Gamma$	$\vdash a\ be\ b.P:$	$\mathrm{CApr}(s)[T]$	by T-PAR

Fig. 8. Typing $a\ be\ b.P$ with resource-policy Γ

$$\Gamma(agent) = \mathrm{CAam}(a_P + a_Q, 1)[T],$$
$$\Gamma(entered) = \mathrm{CAam}(0,0)[T],$$
$$\Gamma(f) = \mathrm{CAam}(\infty,0)[T],$$
$$\text{and } \Gamma(g) = \mathrm{CAam}(1,0)[T]\,.$$

In particular, the typing of the recursive process $\mathsf{rec}\ X.\ldots$ in *system* entails a constraint of the form $\mathrm{CApr}(t)[T] = \mathrm{CApr}(t+1)[T]$. This is possible if and only if $t = \infty$, and as a consequence the capacity of f should also be ∞, so that the firewall is supposed to have infinite size. This is no surprise since it may actually receive any number of external ambients. However, these ambients are contained in the firewall. Hence, one may still integrate this firewall as a component in a system with limited resources.

Cab. Let us consider an environment Γ such that

$$\begin{cases} \Gamma(client) &= \mathrm{CAam}(0,1)[T] \\ \Gamma(call) &= \mathrm{CAam}(1,0)[T] \\ \Gamma(trip) &= \mathrm{CAam}(0,0)[T] \\ \Gamma(arrived) &= \mathrm{CAam}(0,0)[T] \end{cases}$$

$$\begin{cases} \Gamma(end) &= \mathrm{CAam}(0,0)[T] \\ \Gamma(cab) &= \mathrm{CAam}(1,0)[T] \\ \Gamma(site_i) &= \mathrm{CAam}(\infty,0)[T] \\ \Gamma(city) &= \mathrm{CAam}(0,0)[T]. \end{cases}$$

Note in particular that this resource policy specifies that among the ambients that may enter the cab, only those named *client* are actually "controlled"; this corresponds to the property we focus on when analyzing the cab. With these assumptions, the complete cab system is typeable. This means that resources are statically controlled in cabs: *at any step of its execution, the cab may contain at most one client.*

Moreover, we may adopt a different resource policy, defined as follows:

$$\begin{cases} \Gamma(client) &= \mathrm{CAam}(0,0)[T] \\ \Gamma(call) &= \mathrm{CAam}(0,1)[T] \\ \Gamma(trip) &= \mathrm{CAam}(1,1)[T] \\ \Gamma(arrived) &= \mathrm{CAam}(1,1)[T] \end{cases}$$

$$\begin{cases} \Gamma(end) &= \mathrm{CAam}(0,1)[T] \\ \Gamma(cab) &= \mathrm{CAam}(1,0)[T] \\ \Gamma(site_i) &= \mathrm{CAam}(\infty,0)[T] \\ \Gamma(city) &= \mathrm{CAam}(0,0)[T]. \end{cases}$$

The system is also typeable with this choice for Γ, which allows us to control the number of "auxiliary" ambients: at any time, at most one of those may be present in *cab*.

4 More accurate analyses of opening

In this section, we present several refinements of the type system of Sect. 3 that we call systems R, Z, and RZ. While the basic system we have presented so far allows one to type many interesting processes, some relatively simple examples show its limitations. For instance, let us define

$$P_1 \triangleq a[\overline{\mathsf{open}}\ \{a,b\}.\mathsf{rec}\ X.(X \mid b[\mathbf{0}])] \mid \mathsf{open}\ a\,,$$

and suppose that the weight of b is not 0. The construction $\mathsf{rec}\ X.(X \mid b[\mathbf{0}])$ then requires infinite resources. Although the execution would not use any resource inside a, our type system cannot capture this property: the typing will require a to have an infinite capacity.

Similarly, let us define

$$P_2 \triangleq h[\mathsf{rec}\ X.(m[\overline{\mathsf{in}_\downarrow} n.\overline{\mathsf{out}_\uparrow} n.\overline{\mathsf{open}}\ \{m,h\}] \mid \overline{\mathsf{out}_\downarrow} n.\overline{\mathsf{in}_\uparrow} n.\mathsf{open}\ m.X) \mid n[\mathsf{rec}\ Y.\mathsf{in}\ m.\mathsf{out}\ m.Y]]\,,$$

and suppose that the weight of n is not 0. By following the evolution of this term, one may easily notice that a finite capacity for h should be sufficient. However, when deriving a typing for P_2, we conclude that the capacity of h must be infinite.

In both cases, the typing system is not refined enough to express a resource-control property. More specifically, the opening primitive is associated with a resource control that is too strict. For the discussion that follows, we shall use the following notations for the rule R-OPEN:

$$h[\mathsf{open}\ m.P \mid Q \mid m[\overline{\mathsf{open}}\ \{m,h\}.R \mid S]] \longrightarrow h[P \mid Q \mid R \mid S].$$

In order to try and refine the typing of the opening, one may want to make the control on P, Q, R, or S more precise. For technical reasons, we have chosen to concentrate on R and S.

System R. In system R, we introduce a third parameter in ambient types, called r. In $\mathrm{CA}_{\mathrm{AM}}(s,e,r)[T]$, $r \in \overline{\mathbb{N}}$ is an upper bound for the number of resources allocated to R in the opening ambient. Typing rules for open and $\overline{\mathsf{open}}$ become:

$$\frac{\Gamma \vdash m : \mathrm{CA}_{\mathrm{AM}}(s,e,r)[T] \quad \Gamma \vdash P : \mathrm{CA}_{\mathrm{PR}}(t)[T]}{\Gamma \vdash \mathsf{open}\ m.P : \mathrm{CA}_{\mathrm{PR}}(t-e+s+r)[T]}\ CA-open,\ t-e+s+r \geq 0$$

$$\frac{\Gamma \vdash m : \mathrm{CA}_{\mathrm{AM}}(s,e,r)[T] \quad \Gamma \vdash R : \mathrm{CA}_{\mathrm{PR}}(t)[T]}{\Gamma \vdash \overline{\mathsf{open}}\ \{m,h\}.R : \mathrm{CA}_{\mathrm{PR}}(t')[T]}\ CA-coopen,\ t \leq r\ .$$

Using these alternative rules, term P_1 may be satisfactorily typed (i.e., with a finite capacity for a), taking $r = \infty$. Additionally, all results of Sect. 3.2 still remain valid. However, system R does not help with term P_2.

System Z. System Z, on the other hand, improves the control on S. This is particularly important for processes such as

$$M_1.\cdots.M_n.\overline{\mathsf{open}}\ \{m,h\}.R\,,$$

although $M_1.\cdots.M_n$ might acquire as many as, say, s resources, it might also release some or all of them before the actual opening. By taking these releases into account, we may get a better approximation of resource consumption. To do so, we can introduce a parameter z that is compelled to satisfy $z \leq s$. In system Z, ambient types become $\mathrm{CA}_{\mathrm{AM}}(s,e,z)[T]$ with $z \in \mathbb{N}$ and $z \leq s$, and the typing rules are:

$$\frac{\Gamma \vdash m : \mathrm{CA}_{\mathrm{AM}}(s,e,z)[T] \quad \Gamma \vdash P : \mathrm{CA}_{\mathrm{PR}}(t)[T]}{\Gamma \vdash \mathsf{open}\ m.P : \mathrm{CA}_{\mathrm{PR}}(t-e+z)[T]}\ CA-open,\ t-e+z \geq 0$$

$$\frac{\Gamma \vdash m : \mathrm{CA}_{\mathrm{AM}}(s,e,z)[T] \quad \Gamma \vdash R : \mathrm{CA}_{\mathrm{PR}}(t)[T]}{\Gamma \vdash \overline{\mathsf{open}}\ \{m,h\}.R : \mathrm{CA}_{\mathrm{PR}}(t+s-z)[T]}\ CA-coopen\,.$$

Results from Sect. 3.2 also remain valid for system Z. System Z permits a good analysis of term P_2 but cannot handle term P_1 any better than the basic system.

System RZ. Systems R and Z may be naturally merged into system RZ, which yields a more accurate analysis of resources, with ambient types of the form $\mathrm{CA}_{\mathrm{AM}}(s,e,r,z)[T]$, $r \in \overline{\mathbb{N}}$, $z \in \mathbb{N}$ and $z \leq s$ and the following rules:

$$\frac{\Gamma \vdash m : \mathrm{CA}_{\mathrm{AM}}(s,e,r,z)[T] \quad \Gamma \vdash P : \mathrm{CA}_{\mathrm{PR}}(t)[T]}{\Gamma \vdash \mathsf{open}\ m.P : \mathrm{CA}_{\mathrm{PR}}(t-e+z+r)[T]}\ CA-open,\ t-e+z+r \geq 0$$

$$\frac{\Gamma \vdash m : \mathrm{CA}_{\mathrm{AM}}(s,e,r,z)[T] \quad \Gamma \vdash R : \mathrm{CA}_{\mathrm{PR}}(t)[T]}{\Gamma \vdash \overline{\mathsf{open}}\ \{m,h\}.R : \mathrm{CA}_{\mathrm{PR}}(t')[T]}\ \begin{bmatrix} CA-coopen \\ t \leq r, t' \geq s-z \end{bmatrix}.$$

As expected, system RZ correctly handles both terms P_1 and P_2, and results from Sect. 3.2 also remain valid. Hence, system RZ is a more refined, although more complicated, type system.

5 A generalized type system

The type system introduced in Sect. 3 uses the synchronization mechanism of controlled ambients to guarantee, through some simple arithmetic manipulations, the control of resources. By abstracting away from the integers used to represent resource occupation/awareness, and keeping the same general mechanism for type checking, we obtain a general and versatile type system that can be used to express and control several kinds of properties.

This is the subject of this section, where we introduce a *generalized control for ambients* (GCA). GCA, which features almost identical typing rules as the system of Sect. 3, permits the control of more complex properties. After defining this system, we illustrate some of its possible uses.

5.1 An abstract notion of controlled entity

The GCA type system is defined by isolating the essential properties we need when manipulating ambient capacities and weights in the system of Sect. 3. Instead of

counting integers, we parametrize the type system over a set of *resource-usage levels*, corresponding to the following definition:

Definition 3 (Resource-usage levels). *A set of resource-usage levels is given by a tuple* $(\mathcal{R}, \bot, \oplus, \ominus, \preceq)$*, where* $\bot \in \mathcal{R}$*,* $\oplus$*, and* $\ominus$ *are binary operations on* $\mathcal{R}$ *and* $\preceq$ *is a relation on* $\mathcal{R}$*, satisfying the following properties:*

- $(\mathcal{R}, \oplus)$ *is a commutative monoid admitting* $\bot$ *as a neutral element;*
- $\preceq$ *is a partial order on* $\mathcal{R}$*, and for any* $a \in \mathcal{R}$*,* $\bot \preceq a$*;*
- *For any* $a, b, c \in \mathcal{R}$*, if* $a \preceq b$*, then* $a \oplus c \preceq b \oplus c$ *and* $a \ominus c \preceq b \ominus c$*;*
- *For any* $a, b \in \mathcal{R}$*,* $a \preceq (a \ominus b) \oplus b$ *and* $a \preceq (a \oplus b) \ominus b$*.*

Given a set of resource-usage levels, we define the grammar of types according to the rules of Fig. 9. As will be exemplified below, as soon as we have a set of accountable entities, we can think of deriving an instantiation of GCA. Indeed, by instantiating GCA with $(\overline{\mathbb{N}}, 0, +, -, \leq)$, with the appropriate definition of $-$, we obtain a type system very close to that of Sect. 3.

The typing rules for the generalized type system are given in Fig. 10 (we give only the most relevant rules). They are very similar to the rules of Fig. 7, the calculations being performed using operators $\oplus$ and $\ominus$ (see in particular rules T-COOUT, T-OPEN, and T-COOPEN).

As may be seen in rules T-COOPEN or T-PAR, the handling of inequality is slightly different. This is due to the fact that the simple induction step used to prove Lemma 1 in Sect. 3 has to be adapted when the set of resource-usage levels is more complicated than $\mathbb{N}$. For example, in rule T-PAR, increasing t and/or t' is not always sufficient to reach all possible values greater than $t \oplus t'$.

$$
\begin{array}{rcll}
A & ::= & \mathrm{GCA}_{\mathrm{AM}}(s,e)[T] & s,e \in \mathcal{R} \quad \textit{ambients} \\
U & ::= & \mathrm{GCA}_{\mathrm{PR}}(t)[T] & t \in \mathcal{R} \quad \textit{processes} \\
T & ::= & Ssh & \quad\quad\ \textit{messages} \\
 & | & t, A & t \in \mathcal{R}
\end{array}
$$

Fig. 9. Extended types

The extended type system enjoys basically the same properties as the type system of Sect. 3:

Definition 4 (Resource policy and resource usage). *Given a set of resource-usage levels, we call* resource policy *a typing context in the corresponding instantiation of GCA. Given a resource policy* Γ*, we define the* resource usage of a process P according to Γ*, written* $Res_\Gamma(P)$*, as follows:*

- *If* $\Gamma(a) = \mathrm{GCA}_{\mathrm{AM}}(\_, e)[T]$*, then* $Res_\Gamma(a[\,]P) = e$*;*
- $Res_\Gamma(P_1 \mid P_2) = Res_\Gamma(P_1) \oplus Res_\Gamma(P_2)$*;*
- $Res_\Gamma((\nu n : A)\, P) = Res_{\Gamma, n:A}(P)$*.*
- *In all other cases,* $Res_\Gamma(P) = \bot$*.*

Definition 5 (Resource policy compliance). *Given a resource policy* Γ*, we define the judgment* $\Gamma \models P$ *(pronounced "P complies with* Γ*"), as follows:*

- $\Gamma \models n[P]$ *iff* $\Gamma \models P$ *and* $Res_\Gamma(P) \preceq s$*, where capacity* s *is given by* $\Gamma(n) = \mathrm{GCA}_{\mathrm{AM}}(s, \_)[T]$*;*
- $\Gamma \models P_1 | P_2$ *iff* $\Gamma \models P_1$ *and* $\Gamma \models P_2$*;*
- $\Gamma \models (\nu n : A)\, P$ *iff* $\Gamma, n : A \models P$*;*
- *in all other cases,* $\Gamma \models P$*.*

Proposition 2 (Typeable terms comply with resource policies). For any process P, resource policy Γ and process type U, if $\Gamma \vdash P : U$, then $\Gamma \models P$.

$$
\text{T-VAR}\ \frac{\Gamma(X) = \mathrm{GCA}_{\mathrm{PR}}(t)[T]}{\Gamma \vdash X : \mathrm{GCA}_{\mathrm{PR}}(t')[T]}\ t' \succeq t
$$

$$
\text{T-REC}\ \frac{\Gamma, X : \mathrm{GCA}_{\mathrm{PR}}(t)[T] \vdash P : \mathrm{GCA}_{\mathrm{PR}}(t)[T]}{\Gamma \vdash \mathit{rec}\ X.P : \mathrm{GCA}_{\mathrm{PR}}(t')[T]}\ t' \succeq t
$$

$$
\text{T-COIN}\ \frac{\Gamma \vdash P : \mathrm{GCA}_{\mathrm{PR}}(t)[T] \quad \Gamma \vdash m : \mathrm{GCA}_{\mathrm{AM}}(s,e)[T']}{\Gamma \vdash \overline{\mathit{in}_\delta}\ m.P : \mathrm{GCA}_{\mathrm{PR}}(f)[T]}\ f \succeq t \oplus e
$$

$$
\text{T-COOUT}\ \frac{\Gamma \vdash P : \mathrm{GCA}_{\mathrm{PR}}(t)[T] \quad \Gamma \vdash m : \mathrm{GCA}_{\mathrm{AM}}(s,e)[T']}{\Gamma \vdash \overline{\mathit{out}_\delta}\ m.P : \mathrm{GCA}_{\mathrm{PR}}(f)[T]}\ f \succeq t \ominus e
$$

$$
\text{T-OPEN}\ \frac{\Gamma \vdash m : \mathrm{GCA}_{\mathrm{AM}}(s,e)[T] \quad \Gamma \vdash P : \mathrm{GCA}_{\mathrm{PR}}(t)[T]}{\Gamma \vdash \mathit{open}\ m.P : \mathrm{GCA}_{\mathrm{PR}}(f)[T]}\ f \succeq (t \oplus s) \ominus e
$$

$$
\text{T-COOPEN}\ \frac{\Gamma \vdash m : \mathrm{GCA}_{\mathrm{AM}}(s,e)[T] \quad \Gamma \vdash R : \mathrm{GCA}_{\mathrm{PR}}(t)[T]}{\Gamma \vdash \overline{\mathit{open}}\ \{m,h\}.R : \mathrm{GCA}_{\mathrm{PR}}(f)[T]}\ f \succeq t
$$

$$
\text{T-AMB}\ \frac{\Gamma \vdash m : \mathrm{GCA}_{\mathrm{AM}}(s,e)[T] \quad \Gamma \vdash P : \mathrm{GCA}_{\mathrm{PR}}(t)[T]}{\Gamma \vdash m[P] : \mathrm{GCA}_{\mathrm{PR}}(f)[T']}\ f \succeq e, t \preceq s
$$

$$
\text{T-PAR}\ \frac{\Gamma \vdash P : \mathrm{GCA}_{\mathrm{PR}}(t)[T] \quad \Gamma \vdash Q : \mathrm{GCA}_{\mathrm{PR}}(t')[T]}{\Gamma \vdash P \mid Q : \mathrm{GCA}_{\mathrm{PR}}(f)[T]}\ f \succeq t \oplus t'
$$

$$
\text{T-SND}\ \frac{\Gamma \vdash m : A}{\Gamma \vdash \langle m \rangle : \mathrm{GCA}_{\mathrm{PR}}(t')[t, A]}\ t' \succeq t \qquad \text{T-RCV}\ \frac{\Gamma, x : A \vdash P : \mathrm{GCA}_{\mathrm{PR}}(t)[t, A]}{\Gamma \vdash (x : A)P : \mathrm{GCA}_{\mathrm{PR}}(t')[t, A]}
$$

Fig. 10. Typing rules for generalized control

Theorem 3 (Subject reduction). *For any processes P, Q, resource policy Γ, and type U, if $\Gamma \vdash P : U$ and $P \longrightarrow Q$, then $\Gamma \vdash Q : U$.*

We do not present the proofs as they follow faithfully the proofs for the system of Sect. 3.

5.2 Examples

We now show some possible uses of GCA, corresponding to different choices for the set of resource-usage levels.

5.2.1 Cost control

We return to our cab example and show how we can use system GCA to limit the number of trips a cab can make. For this, we introduce the following set of resource-usage levels:

- $\mathcal{R} = \overline{\mathbb{N}}$
- $\forall x, y\ x \oplus y = x + y$
- $\forall x\ x \ominus y = x$
- $\forall x, y\ x \preceq y \iff x \leq y$.

Now consider the following definitions:

$$cab \triangleq cab[\underbrace{ONE\_TRIP.ONE\_TRIP.\cdots.ONE\_TRIP}_{k \text{ times}}.\mathbf{0}],$$

where *ONE_TRIP* is the sequence of (co)capabilities corresponding to one trip of the cab (see above).

Using $\mathcal{R}_2$ and the new definition of *cab*, we may specify the following resource-control policy (the most important points are typeset in bold):

$$\begin{cases} \Gamma(client) &= \text{CA}_{\text{AM}}(0, \mathbf{1})[T] \\ \Gamma(call) &= \text{CA}_{\text{AM}}(1, 0)[T] \\ \Gamma(trip) &= \text{CA}_{\text{AM}}(0, 0)[T] \\ \Gamma(arrived) &= \text{CA}_{\text{AM}}(0, 0)[T] \end{cases}$$

$$\begin{cases} \Gamma(end) &= \text{CA}_{\text{AM}}(0, 0)[T] \\ \Gamma(cab) &= \text{CA}_{\text{AM}}(\mathbf{k}, 0)[T] \\ \Gamma(site_i) &= \text{CA}_{\text{AM}}(\infty, 0)[T] \\ \Gamma(city) &= \text{CA}_{\text{AM}}(0, 0)[T]. \end{cases}$$

Among other things, this policy specifies that a cab will not answer more than k requests. And since it can be proved that the system is resource controlled according to Γ, we have successfully used $\mathcal{R}_2$ to control a new form of resource – actually, a form of *nonreleasable resource*.

Using this instantiation of GCA, this very simple example could be developed along the lines of schedule policy checking. In such a scenario, a limited amount of tasks (the client ambients of the cab protocol) can run at a given site, the type system being used to guarantee this bound. Moreover, all tasks need not be present from the very beginning as is the case above, but we could think of having the tasks entering the host site and being scheduled until the limit number of runnable tasks is reached.

It should be remarked that GCA is used here to assess a property that is somehow more "dynamic" than the resource usage policies discussed above.

5.2.2 Combining policies.

One could think of several other instantiations of GCA, using, e.g., booleans for $\mathcal{R}$ to check binary properties of ambient-based descriptions of systems. We shall not present here these other possible extensions, but instead stress another important feature of this generalized type system, namely, that different type systems can be associated componentwise with combine different kinds of analyses. This is expressed using the following property:

Proposition 3 (Combining resource-usage levels). Let us consider two sets of resource-usage levels: $(\mathcal{R}_1, \bot_1, \oplus_1, \ominus_1, \preceq_1)$ and $(\mathcal{R}_2, \bot_2, \oplus_2, \ominus_2, \preceq_2)$. Then the set defined by

- $\mathcal{R} = \mathcal{R}_1 \times \mathcal{R}_2$;
- $\bot = (\bot_1, \bot_2)$;
- $(a_1, a_2) \oplus (b_1, b_2) = (a_1 \oplus_1 b_1, a_2 \oplus_2 b_2)$,
  $(a_1, a_2) \ominus (b_1, b_2) = (a_1 \ominus_1 b_1, a_2 \ominus_2 b_2)$;
- $(a_1, a_2) \preceq (b_1, b_2)$ iff $a_1 \preceq_1 b_1$ and $a_2 \preceq b_2$

is a set of resource-usage levels.

Using this property, it is easy to combine any number of policies and check them simultaneously. For instance, we could check that the cab never carries more than one passenger (Sect. 3.3), never contains more than one auxiliary ambient (Sect. 3.3), and cannot move more than ten times (Sect. 5.2.1).

6 Conclusion

The language of controlled ambients has been introduced to analyze resource control in a distributed and mobile setting through an accurate programming of movements and synchronizations. We have enhanced our formalism with a type system for the static control of resources, and extensions of the basic type system have also been presented under the form of GCA. Further, examples show that indications on the maximal amount of resources needed by a process match rather closely the actual amount of resources that may be reached in the worst case, which suggests that the solution we propose could serve as the basis for a study of resource-control properties on a larger scale.

Among extensions of the present work, we are currently enriching the language and type system to include communication of capabilities, as in the original mobile ambients calculus [5]. We are also studying type inference for our system, which would enhance (untyped) controlled ambients with a procedure for the automatic guessing of resource needs. It seems that by requiring the recursion variables to be explicitly typed, type inference

is decidable, and a rather natural algorithm can compute a minimal type for a given process, if it exists. In particular, the "message" component of terms leads to a classical unification problem. The question becomes more problematic if no information is given for recursion variables: one can compute a set of inequalities (like those given for the example of renaming in Sect. 3), but solving it in the general case would require more work, as would also the porting of such an inference procedure to GCA.

As reported in [25], our approach can be adapted to other formalisms for mobile and distributed computation that provide a primitive notion of location, such as seals [28], boxed ambients [2, 3], nomadic π [27], and kells [24]. In π-calculus-like languages, a natural notion of resource is given by *channels,* which represents a slightly different point of view w.r.t. the present work. Introducing resource control in calculi like the π-calculus or the distributed π-calculus [21] represents a challenging direction for future work.

We could also consider combining our type system for resource control with other typing disciplines, adapted from the single threadness types of [18], or the mandatory access control of [2]. It seems that, using the generalized type system, controlled ambients could also be used to approximate some of the analyses done in [9, 14], where, in a context in which *security levels* are associated with processes, types are used to check that no agent can access an information having a security level higher than its own. We are also trying to enhance GCA and to instantiate it to a form of movement typing, approximating the analysis of [4].

We have not addressed the issue of behavioral equivalences for CA. A possible outcome of such a study could be to validate a more elaborate treatment of resources involving operations like garbage collection, which would allow one to make available uselessly occupied resources. An example is the perfect firewall equation of [12]: when $c \notin \mathrm{fn}(P)$, process $(\nu c)\, c[P]$ may manipulate some resources while being actually equivalent to $\mathbf{0}$.

Other related works. Another attempt at resource control in ambients is developed in [23]. As opposed to CA, the *capacity-bounded computational ambients* use a dynamic type system and slots, somewhat reminiscent of some of our cocapabilities, to stand for resources. The type system then counts the number of available slots (at any depth) in a process.

Yet another form of accounting on mobile ambients is introduced in [7]. In a calculus with a slightly different form of recursion than in CA (and without cocapabilities), the authors introduce a type system to count the number of active outputs and ambients (at any depth) in a process. This analysis, however, is not aimed at modeling resources: it tries to isolate a finite-control fragment of mobile ambients on which model checking w.r.t. the ambient logic is decidable through state-space exploration.

Other projects aim at controlling resources in possibly mobile systems without resorting to mobile process algebras. Hughes and Pareto [17] present a modified ML language with sized types in which bounds may be given to stack consumption. As in our framework, resources are releasable entities; however, this approach seems more specialized than ours and moreover concentrates on a sequential model. Similarly, [8] introduces a variant of the Typed Assembly Language *"augmenting TAL's very low-level safety certification with running-time guarantees"*, while Quantum [20] may be used to describe distributed systems from the point of view of their resource consumption. In contrast to our work, both these approaches consider nonreleasable resources. Another programming language, PLAN [15], has been designed specifically for active networks and also handles some form of resource bounds. Although PLAN accounts for both releasable (space, bandwidth) and nonreleasable (time) resources, it handles neither recursion nor concurrency on one node. A related line of research is followed in [1, 16], where means to guarantee bounds on the time or space consumption required for the execution of (sequential) functions are proposed.

These works all focus on resource control; however, none of these approaches can be directly compared to ours. It might be interesting to study if and how our methods could be integrated with these works in order to combine several forms of resource control.

Acknowledgements. We would like to thank Davide Sangiorgi for suggesting the original idea behind CA and providing insightful suggestions along this work.

References

1. Amadio R (2003) Max-plus quasi-interpretations. In: Proceedings of TLCA'03. Lecture notes in computer science, vol 2701. Springer, Berlin Heidelberg New York, pp 31–45
2. Bugliesi M, Castagna G, Crafa S (2001) Boxed ambients. In: Proceedings of TACS 2001. Lecture notes in computer science, vol 2215. Springer, Berlin Heidelberg New York, pp 38–63
3. Bugliesi M, Crafa S, Merro M, Sassone V (2002) Communication interference in mobile boxed ambients. In: Proceedings of FST-TCS'02 Lecture notes in computer science. Springer, Berlin Heidelberg New York
4. Cardelli L, Ghelli G, Gordon A (2002) Types for the ambient calculus. Inf Comput 177(2):160–194
5. Cardelli L, Gordon AD (1998) Mobile ambients. In: Proceedings of FOSSACS'98. Lecture notes in computer science, vol 1378. Springer, Berlin Heidelberg New York, pp 140–155
6. Cardelli L, Gordon AD (1999) Types for mobile ambients. In: Proceedings of the symposium on principles of programming languages (POPL'99). ACM Press, New York, pp 79–92
7. Charatonik W, Gordon AD, Talbot J-M (2002) Finite-control mobile ambients. In: Proceedings of ESOP'02. Lecture notes in computer science, vol 2305. Springer, Berlin Heidelberg New York, pp 295–313
8. Crary K, Weirich S (2000) Resource bound certification. In: Proceedings of the symposium on principles of programming languages (POPL'00). ACM Press, New York, pp 184–198
9. Dezani-Ciancaglini M, Salvo I (2000) Security types for mobile safe ambients. In: Proceedings of ASIAN'00. Lecture notes in computer science, vol 1961. Springer, Berlin Heidelberg New York, pp 215–236

10. Fournet C, Lévy J-J, Schmitt A (1872) A distributed implementation of mobile ambients. In: Proceedings of IFIP TCS'00. Springer, Berlin Heidelberg New York, pp 348–364
11. Gazagnaire T, Pous D (2002) Implémentation des Controlled Ambients en JoCaml. Student project – Magistère d'Informatique ENS Lyon, France
12. Gordon AD, Cardelli L (1999) Equational properties of mobile ambients. In: Proceedings of FOSSACS'99. Lecture notes in computer science, vol 1578. Springer, Berlin Heidelberg New York, pp 212–226
13. Guan X, Yang Y, You J (2000) Making ambients more robust. In: Proceedings of the international conference on software: theory and practice, pp 377–384
14. Hennessy M, Riely J (1998) Resource access control in systems of mobile agents. In: Proceedings of HLCL '98. Electronic notes in theoretical computer science, vol 163. Elsevier, Amsterdam, pp 3–17
15. Hicks M, Kakkar P, Moore JT, Gunter CA, Nettles S (1999) PLAN: A Packet Language for Active Networks. In: Proceedings of ICFP'99. ACM Press, New York, pp 86–93
16. Hofmann M (2002) The strength of non-size increasing computation. In: Proceedings of the 29th ACM symposium on principles of programming languages (POPL'02). ACM Press, New York, pp 260–269
17. Hughes J, Pareto L (1999) Recursion and dynamic datastructures in bounded space: towards embedded ML programming. In: Proceedings of ICFP'99. ACM Press, New York, pp 70–81
18. Levi F, Sangiorgi D (2000) Controlling interference in ambients. In: Proceedings of the symposium on principles of programming languages. ACM Press, New York, pp 352–364
19. Merro M, Hennessy M (2002) Bisimulation congruences in safe ambients. In: Proceedings of POPL'02. ACM Press, New York, pp 71–80
20. Moreau L (1998) A distributed garbage collector with diffusion tree reorganisation and mobile objects. In: Proceedings of ICFP'98. ACM Press, New York, pp 204–215
21. Riely J, Hennessy M (1998) A typed language for distributed mobile processes. In: Proceedings of POPL'98. ACM Press, New York, pp 378–390
22. Sangiorgi D, Valente A (2001) A distributed abstract machine for safe ambients. In: Proceedings of ICALP'01
23. Sassone V, Barbanera F, Bugliesi M, Dezani M (2003) A calculus of bounded capacities. In: Proceedings of ASIAN'03. Lecture notes in computer science. Springer, pp 205–223
24. Stefani J-B (2003) A calculus of higher-order distributed components. Technical Report 4692, INRIA
25. Teller D (2003) Formalisms for mobile resource control. In: Proceedings of FGC'03. Electronic notes in theoretical computer science, vol 85. Elsevier, Amsterdam
26. Teller D, Zimmer P, Hirschkoff D (2002) Using ambients to control resources. In: Proceedings of CONCUR'02. Lecture notes in computer science. Springer, Berlin Heidelberg New York
27. Unyapoth A (2001) Nomadic pi calculi: expressing and verifying infrastructure for mobile computation. PhD thesis, Computer Laboratory, University of Cambridge, June 2001
28. Vitek J, Castagna G (1999) Seal: a framework for secure mobile computations. In: Internet programming languages. Lecture notes in computer science, vol 1686. Springer, Berlin Heidelberg New York

Appendix – Proof of subject reduction (Theorem 1)

Strengthening and weakening. The proofs of the following lemmas are easy and are not given.

Lemma 3. *If* $\Gamma, n : A \vdash P : U$ *and* $n \notin fn(P)$, *then* $\Gamma \vdash P : U$.

Lemma 4. *If* $\Gamma, X : U' \vdash P : U$ *and* $X \notin fv(P)$, *then* $\Gamma \vdash P : U$.

Lemma 5. *If* $\Gamma \vdash P : U$ *and* $n \notin fn(P)$, *then* $\Gamma, n : A \vdash P : U$.

Structural congruence

Lemma 6. *If* $P \equiv Q$ *and* $\Gamma \vdash P : U$, *then* $\Gamma \vdash Q : U$.
If $Q \equiv P$ *and* $\Gamma \vdash P : U$, *then* $\Gamma \vdash Q : U$.

Proof: By mutual induction, on the derivation of $P \equiv Q$ and $Q \equiv P$.

Case S-parnil.

- If $\Gamma \vdash P : \text{CAPR}(t)[T]$, we can type $\Gamma \vdash \mathbf{0} : \text{CAPR}(0)[T]$ and then $\Gamma \vdash P \mid \mathbf{0} : \text{CAPR}(t)[T]$ by CA-PAR.
- If $\Gamma \vdash P \mid \mathbf{0} : \text{CAPR}(t)[T]$, this must have been derived by CA-PAR from $\Gamma \vdash P : \text{CAPR}(t_1)[T]$ and $\Gamma \vdash \mathbf{0} : \text{CAPR}(t_2)[T]$ with $t_1 + t_2 = t$. Since $t_1 \leq t$, we have $\Gamma \vdash P : \text{CAPR}(t)[T]$ using Lemma 1.

Case S-respar.

- Suppose that $\Gamma \vdash (\nu n : A)(P|Q) : \text{CAPR}(t)[T]$. This must have been derived by CA-RES from $\Gamma, n : A \vdash P|Q : \text{CAPR}(t)[T]$, which in turn must have been derived by CA-PAR from $\Gamma, n : A \vdash P : \text{CAPR}(t_1)[T]$ and $\Gamma, n : A \vdash Q : \text{CAPR}(t_2)[T]$ with $t_1 + t_2 = t$. From the first affirmation we get $\Gamma \vdash (\nu n : A)P : \text{CAPR}(t_1)[T]$ by CA-RES. From the second one, and since $n \notin fn(Q)$, we can use Lemma 2 and obtain $\Gamma \vdash Q : \text{CAPR}(t_2)[T]$. Finally, using CA-PAR, we get $\Gamma \vdash (\nu : A)P \mid Q : \text{CAPR}(t)[T]$.
- Starting from $\Gamma \vdash (\nu : A)P \mid Q : \text{CAPR}(t)[T]$, the reasoning is similar, except that we use Lemma 4 instead of Lemma 2.

Case S-amb (for example). $\Gamma \vdash m[P] : \text{CAPR}(t)[T']$ must have been derived by CA-AMB from $\Gamma \vdash m : \text{CAAM}(s, e)[T]$ and $\Gamma \vdash P : \text{CAPR}(a)[T]$ with $a \leq s$ and $e \leq t$. By induction hypothesis, since $P \equiv Q$, we have $\Gamma \vdash Q : \text{CAPR}(a)[T]$. Then, using CA-AMB, we can derive $\Gamma \vdash m[Q] : \text{CAPR}(t')[T]$.

The other cases are similar or trivial. □

Substitution

Lemma 7. *If* $\Gamma, X : U \vdash P : U'$ *and* $\Gamma \vdash Q : U$, *then* $\Gamma \vdash P\{X \leftarrow Q\} : U'$.

Proof: Let $U = \text{CAPR}(t)[T]$. In the derivation tree of $\Gamma, X : U \vdash P : U'$, all occurrences of X must have been typed using CA-VAR-PROC. These occurrences have the form $\Gamma, X : U \vdash X : \text{CAPR}(t')[T]$ with $t' \geq t$. By Lemma 1, we also have $\Gamma \vdash Q : \text{CAPR}(t')[T]$. We can then replace the node X by a derivation subtree for Q. Thus, we get a derivation tree for $\Gamma, X : U \vdash P\{X \leftarrow Q\} : U'$. Finally, using Lemma 3, we can remove X from the environment, since it is no longer used, and obtain: $\Gamma \vdash P\{X \leftarrow Q\} : U'$. □

Lemma 8. *If* $\Gamma, y : A \vdash P : U$ *and* $\Gamma \vdash x : A$, *then* $\Gamma \vdash P\{y \leftarrow x\} : U$.

Proof: In the derivation tree of $\Gamma, y : A \vdash P : U$, all occurrences of y must have been typed using CA-NAME. If we replace them with $\Gamma \vdash x : A$, we get a derivation tree for $\Gamma, y : A \vdash P\{y \leftarrow x\} : U$. Finally, using Lemma 2, we can remove y from the environment since it is no longer used in $P\{y \leftarrow x\}$. □

Proof of Theorem 1 (Subject reduction). For any processes P, Q, resource policy Γ and type U, if $\Gamma \vdash P : U$ and $P \longrightarrow Q$, then $\Gamma \vdash Q : U$. *Proof:* By induction on the derivation of $P \longrightarrow Q$.

Case R-in. If $A \longrightarrow B$ by one step of R-IN, we have

$$\begin{cases} A &= m[\mathsf{in}\ n.P \mid Q] \mid n[\overline{\mathsf{in}_\downarrow}\ m.R \mid S] \mid \overline{\mathsf{out}_\downarrow}\ m.T \\ B &= n[R \mid S \mid m[P \mid Q]] \mid T\,. \end{cases}$$

Let Γ be an environment such that

$$\begin{cases} \Gamma(m) &= \text{CA}_{\text{AM}}(s_m, e_m)[T_m] \\ \Gamma(n) &= \text{CA}_{\text{AM}}(s_n, e_n)[T_n] \end{cases}$$

$$\begin{cases} \Gamma \vdash P : \text{CA}_{\text{PR}}(t_P)[T_P] \\ \Gamma \vdash Q : \text{CA}_{\text{PR}}(t_Q)[T_Q] \\ \Gamma \vdash R : \text{CA}_{\text{PR}}(t_R)[T_R] \\ \Gamma \vdash S : \text{CA}_{\text{PR}}(t_S)[T_S] \\ \Gamma \vdash T : \text{CA}_{\text{PR}}(t_T)[T_T]. \end{cases}$$

This environment is generic and represents the general case. Let us follow the only derivation that may type A in Γ:

			Typing $\mathsf{in}\ n.P$
Γ	$\vdash P :$	$\text{CA}_{\text{PR}}(t_P)[T_P]$	by hypothesis
Γ	$\vdash n :$	$\text{CA}_{\text{AM}}(s_n, e_n)[T_n]$	by hypothesis
$\Rightarrow \Gamma$	$\vdash \mathsf{in}\ n.P :$	$\text{CA}_{\text{PR}}(t_P)[T_P]$	CA-IN
			Typing $\mathsf{in}\ n.P \mid Q$
Γ	$\vdash \mathsf{in}\ n.P :$	$\text{CA}_{\text{PR}}(t_P)[T_P]$	see above
Γ	$\vdash Q :$	$\text{CA}_{\text{PR}}(t_Q)[T_Q]$	by hypothesis
$\Rightarrow \Gamma$	$\vdash \mathsf{in}\ n.P \mid Q :$	$\text{CA}_{\text{PR}}(t_P + t_Q)[T_P]$	CA-PAR
	where	$T_P = T_Q$	
			Typing $m[\mathsf{in}\ n.P \mid Q]$
Γ	$\vdash \mathsf{in}\ m.P \mid Q :$	$\text{CA}_{\text{PR}}(t_P + t_Q)[T_P]$	see above
Γ	$\vdash m :$	$\text{CA}_{\text{AM}}(s_m, e_m)[T_m]$	by hypothesis
$\Rightarrow \Gamma$	$\vdash m[\mathsf{in}\ n.P \mid Q] :$	$\text{CA}_{\text{PR}}(t_1)[T_1]$	CA-AMB
	where	$t_P + t_Q \le s_m$	
		$e_m \le t_1$	
		$T_P = T_m$	

			Typing $\overline{\mathsf{in}_\downarrow}\ m.R$
Γ	$\vdash m :$	$\text{CA}_{\text{AM}}(s_m, e_m)[T_m]$	by hypothesis
Γ	$\vdash R :$	$\text{CA}_{\text{PR}}(t_R)[T_R]$	by hypothesis
$\Rightarrow \Gamma$	$\vdash \overline{\mathsf{in}_\downarrow}\ m.R :$	$\text{CA}_{\text{PR}}(t_R + e_m)[T_R]$	CA-COIN
			Typing $\overline{\mathsf{in}_\downarrow}\ m.R \mid S$
Γ	$\vdash \overline{\mathsf{in}_\downarrow}\ m.R :$	$\text{CA}_{\text{PR}}(t_R + e_m)[T_R]$	see above
Γ	$\vdash S :$	$\text{CA}_{\text{PR}}(t_S)[T_S]$	by hypothesis
$\Rightarrow \Gamma$	$\vdash \overline{\mathsf{in}_\downarrow}\ m.R \mid S :$	$\text{CA}_{\text{PR}}(t_S + t_R + e_m)[T_R]$	CA-PAR
	where	$T_S = T_R$	
			Typing $n[\overline{\mathsf{in}_\downarrow}\ m.R \mid S]$
Γ	$\vdash \overline{\mathsf{in}_\downarrow}\ m.R \mid S :$	$\text{CA}_{\text{PR}}(t_S + t_R + e_m)[T_R]$	see above
Γ	$\vdash n :$	$\text{CA}_{\text{AM}}(s_n, e_n)[T_n]$	by hypothesis
$\Rightarrow \Gamma$	$\vdash n[\overline{\mathsf{in}_\downarrow}\ m.R \mid S] :$	$\text{CA}_{\text{PR}}(t_2)[T_2]$	CA-AMB
	where	$t_S + t_R + e_m \le s_n$	
		$e_n \le t_2$	
		$T_R = T_n$	

			Typing $\overline{\mathsf{out}_\downarrow}\ m.T$
Γ	$\vdash T :$	$\text{CA}_{\text{PR}}(t_T)[T_T]$	by hypothesis
Γ	$\vdash m :$	$\text{CA}_{\text{AM}}(s_m, e_m)[T_m]$	by hypothesis
$\Rightarrow \Gamma$	$\vdash \overline{\mathsf{out}_\downarrow}\ m.T :$	$\text{CA}_{\text{PR}}(t_T - e_m)[T_T]$	CA-COOUT
	where	$t_T \ge e_m$	
			Typing $n[\ldots] \mid \overline{\mathsf{out}_\downarrow}\ m.T$
Γ	$\vdash \overline{\mathsf{out}_\downarrow}\ m.T :$	$\text{CA}_{\text{PR}}(t_T - e_m)[T_T]$	see above
Γ	$\vdash n[\cdots] :$	$\text{CA}_{\text{PR}}(t_2)[T_2]$	see above
$\Rightarrow \Gamma$	$\vdash n[\cdots] \mid$		
	$\overline{\mathsf{out}_\downarrow}\ m.T :$	$\text{CA}_{\text{PR}}(t_T + t_2 - e_m)[T_T]$	CA-PAR
	where	$T_T = T_2$	
			Typing A
Γ	$\vdash n[\cdots] \mid$		
	$\overline{\mathsf{out}_\downarrow}\ m.T :$	$\text{CA}_{\text{PR}}(t_T + t_2 - e_m)[T_T]$	see above
Γ	$\vdash m[\cdots] :$	$\text{CA}_{\text{PR}}(t_1)[T_1]$	see above
$\Rightarrow \Gamma$	$\vdash A :$	$\text{CA}_{\text{PR}}(t_T + t_1 + t_2 - e_m)[T_T]$	CA-PAR
	where	$T_T = T_1$	

From this set of preconditions we deduce that the following derivation is also valid:

			Typing $P \mid Q$
Γ	$\vdash P :$	$\text{CA}_{\text{PR}}(t_P)[T_P]$	by hypothesis
Γ	$\vdash Q :$	$\text{CA}_{\text{PR}}(t_Q)[T_Q]$	by hypothesis
	since	$T_P = T_Q$	
$\Rightarrow \Gamma$	$\vdash P \mid Q :$	$\text{CA}_{\text{PR}}(t_P + t_Q)[T_P]$	CA-PAR
			Typing $m[P \mid Q]$
Γ	$\vdash P \mid Q :$	$\text{CA}_{\text{PR}}(t_P + t_Q)[T_P]$	see above
Γ	$\vdash m :$	$\text{CA}_{\text{AM}}(s_m, e_m)[T_m]$	by hypothesis
	since	$T_P = T_m$	
		$t_P + t_Q \le s_m$	
$\Rightarrow \Gamma$	$\vdash m[P \mid Q] :$	$\text{CA}_{\text{PR}}(e_m)[T_R]$	CA-AMB
			Typing $R \mid S$
Γ	$\vdash R :$	$\text{CA}_{\text{PR}}(t_R)[T_R]$	by hypothesis
Γ	$\vdash S :$	$\text{CA}_{\text{PR}}(t_S)[T_S]$	by hypothesis
	since	$T_R = T_S$	
$\Rightarrow \Gamma$	$\vdash R \mid S :$	$\text{CA}_{\text{PR}}(t_R + t_S)[T_R]$	CA-PAR
			Typing $R \mid S \mid m[\cdots]$
Γ	$\vdash m[P \mid Q] :$	$\text{CA}_{\text{PR}}(e_m)[T_R]$	see above
Γ	$\vdash R \mid S :$	$\text{CA}_{\text{PR}}(t_R + t_S)[T_R]$	see above
$\Rightarrow \Gamma$	$\vdash R \mid S \mid m[\cdots] :$	$\text{CA}_{\text{PR}}(t_R + t_S + e_m)[T_R]$	CA-PAR

			Typing $n[R \mid S \mid m[\cdots]]$
Γ	$\vdash R\|S\|m[\cdots]:$	$\text{CAPR}(t_R+t_S+e_m)[T_R]$	see above
Γ	$\vdash n:$	$\text{CAAM}(s_n,e_n)[T_n]$	by hypothesis
	since	$e_n \le t_2 \le t_1+t_2-e_m$	
		$T_R = T_n$	
		$t_R+t_S+e_m \le s_n$	
$\Rightarrow\Gamma$	$\vdash n[\cdots]:$	$\text{CAPR}(t_1+t_2-e_m)[T_T]$	CA-AMB

			Typing B
Γ	$\vdash n[\cdots]:$	$\text{CAPR}(t_1+t_2-e_m)[T_T]$	see above
Γ	$\vdash T:$	$\text{CAPR}(t_T)[T_T]$	by hypothesis
$\Rightarrow\Gamma$	$\vdash B:$	$\text{CAPR}(t_T+t_1+t_2-e_m)[T_T]$	CA-PAR

Case R-out. If $A \longrightarrow B$ by one step of R-OUT, we have

$$\begin{cases} A &= n[m[\mathsf{out}\ n.P \mid Q] \mid \overline{\mathsf{out}_\uparrow}\ m.R \mid S] \mid \overline{\mathsf{in}_\uparrow}\ m.T \\ B &= m[P \mid Q] \mid n[R \mid S] \mid T\,. \end{cases}$$

Let Γ be an environment such that

$$\begin{cases} \Gamma(m) &= \text{CAAM}(s_m,e_m)[T_m] \\ \Gamma(n) &= \text{CAAM}(s_n,e_n)[T_n] \end{cases}$$

$$\begin{cases} \Gamma \vdash P : \text{CAPR}(t_P)[T_P] \\ \Gamma \vdash Q : \text{CAPR}(t_Q)[T_Q] \\ \Gamma \vdash R : \text{CAPR}(t_R)[T_R] \\ \Gamma \vdash S : \text{CAPR}(t_S)[T_S] \\ \Gamma \vdash T : \text{CAPR}(t_T)[T_T]. \end{cases}$$

This environment is generic and represents the general case. Let us follow the only derivation that may type A in Γ:

			Typing $\mathsf{out}\ n.P$
Γ	$\vdash P:$	$\text{CAPR}(t_P)[T_P]$	by hypothesis
$\Rightarrow\Gamma$	$\vdash \mathsf{out}\ n.P:$	$\text{CAPR}(t_P)[T_P]$	CA-OUT

			Typing $\mathsf{out}\ n.P \mid Q$
Γ	$\vdash \mathsf{out}\ n.P:$	$\text{CAPR}(t_P)[T_P]$	see above
Γ	$\vdash Q:$	$\text{CAPR}(t_Q)[T_Q]$	by hypothesis
$\Rightarrow\Gamma$	$\vdash \mathsf{out}\ m.P\|Q:$	$\text{CAPR}(t_P+t_Q)[T_P]$	CA-PAR
	where	$T_P = T_Q$	

			Typing $m[\mathsf{out}\ n.P \mid Q]$
Γ	$\vdash \mathsf{out}\ m.P\|Q:$	$\text{CAPR}(t_P+t_Q)[T_P]$	see above
Γ	$\vdash m:$	$\text{CAAM}(s_m,e_m)[T_m]$	by hypothesis
$\Rightarrow\Gamma$	$\vdash m[\mathsf{out}\ n.P \mid Q]:$	$\text{CAPR}(t_1)[T_1]$	CA-AMB
	where	$t_P+t_Q \le s_m$	
		$e_m \le t_1, T_P = T_m$	

			Typing $\overline{\mathsf{out}_\uparrow}\ m.R$
Γ	$\vdash R:$	$\text{CAPR}(t_R)[T_R]$	by hypothesis
Γ	$\vdash m:$	$\text{CAAM}(s_m,e_m)[T_m]$	by hypothesis
$\Rightarrow\Gamma$	$\vdash \overline{\mathsf{out}_\uparrow}\ m.R:$	$\text{CAPR}(t_R-e_m)[T_R]$	CA-COOUT
	where	$e_m \le t_R$	

			Typing $\overline{\mathsf{out}_\uparrow}\ m.R\|S$
Γ	$\vdash \overline{\mathsf{out}_\uparrow}\ m.R:$	$\text{CAPR}(t_R-e_m)[T_R]$	see above
Γ	$\vdash S:$	$\text{CAPR}(t_S)[T_S]$	by hypothesis
$\Rightarrow\Gamma$	$\vdash \overline{\mathsf{out}_\uparrow}\ m.R\|S:$	$\text{CAPR}(t_S+t_R-e_m)[T_R]$	CA-PAR
	where	$T_S = T_R$	

			Typing $m[\cdots] \mid \overline{\mathsf{out}_\uparrow}\ m.R \mid S$
Γ	$\vdash \overline{\mathsf{out}_\uparrow}\ m.R \mid S:$	$\text{CAPR}(t_S+t_R-e_m)[T_R]$	see above
Γ	$\vdash m[\mathsf{out}\ n.P \mid Q]:$	$\text{CAPR}(t_1)[T_1]$	see above
$\Rightarrow\Gamma$	$\vdash m[\cdots]\|$	CAPR	
	$\overline{\mathsf{out}_\uparrow}\ m.R\|S:$	$(t_1+t_S+t_R-e_m)[T_R]$	CA-PAR
	where	$T_1 = T_R$	

			Typing $n[m[\cdots] \mid \overline{\mathsf{out}_\uparrow}\ m.R \mid S]$
Γ	$\vdash m[\cdots]\|$	CAPR	
	$\overline{\mathsf{out}_\uparrow}\ m.R\|S:$	$(t_1+t_S+t_R-e_m)[T_R]$	see above
Γ	$\vdash n:$	$\text{CAAM}(s_n,e_n)[T_n]$	by hypothesis
$\Rightarrow\Gamma$	$\vdash n[\cdots]:$	$\text{CAPR}(t_2)[T_2]$	CA-AMB
	where	$e_n \le t_2$	
		$T_R = T_n$	
		$t_1+t_S+t_R-e_m \le s_n$	

			Typing $\overline{\mathsf{in}_\uparrow}\ m.T$
Γ	$\vdash T:$	$\text{CAPR}(t_T)[T_T]$	by hypothesis
Γ	$\vdash m:$	$\text{CAAM}(s_m,e_m)[T_m]$	by hypothesis
$\Rightarrow\Gamma$	$\vdash \overline{\mathsf{in}_\uparrow}\ m.T:$	$\text{CAPR}(t_T+e_m)[T_T]$	CA-COIN

			Typing A
Γ	$\vdash n[\cdots]:$	$\text{CAPR}(t_2)[T_2]$	see above
Γ	$\vdash \overline{\mathsf{in}_\uparrow}\ m.T:$	$\text{CAPR}(t_T+e_m)[T_T]$	see above
$\Rightarrow\Gamma$	$\vdash A:$	$\text{CAPR}(t_2+t_T+e_m)[T_T]$	CA-PAR
	where	$T_2 = T_T$	

From this set of preconditions we deduce that the following derivation is also valid:

			Typing $P\|Q$
Γ	$\vdash P:$	$\text{CAPR}(t_P)[T_P]$	by hypothesis
Γ	$\vdash Q:$	$\text{CAPR}(t_Q)[T_Q]$	by hypothesis
	since	$T_P = T_Q$	
$\Rightarrow\Gamma$	$\vdash P\|Q:$	$\text{CAPR}(t_P+t_Q)[T_P]$	CA-PAR

			Typing $m[P\|Q]$
Γ	$\vdash P\|Q:$	$\text{CAPR}(t_P+t_Q)[T_P]$	see above
Γ	$\vdash m:$	$\text{CAAM}(s_m,e_m)[T_m]$	by hypothesis
	since	$T_P = T_m$	
		$t_P+t_Q \le s_m$	
$\Rightarrow\Gamma$	$\vdash m[P\|Q]$	$\text{CAPR}(e_m)[T_T]$	CA-AMB

			Typing $R\|S$
Γ	$\vdash R:$	$\text{CAPR}(t_R)[T_R]$	by hypothesis
Γ	$\vdash S:$	$\text{CAPR}(t_S)[T_S]$	by hypothesis
	since	$T_R = T_S$	
$\Rightarrow\Gamma$	$\vdash R\|S:$	$\text{CAPR}(t_R+t_S)[T_R]$	CA-PAR

			Typing $n[R\|S]$
Γ	$\vdash R\|S:$	$\text{CAPR}(t_R+t_S)[T_R]$	see above
Γ	$\vdash n:$	$\text{CAAM}(s_n,e_n)[T_n]$	by hypothesis
	since	$e_n \le t_2$	
		$T_R = T_n$	
		$t_R+t_S \le t_1-e_m+t_R+t_S \le s_n$	
$\Rightarrow\Gamma$	$\vdash n[R\|S]$	$\text{CAPR}(t_2)[T_T]$	CA-AMB

			Typing $n[R \mid S] \mid T$
Γ	$\vdash n[R\|S]$	$\text{CAPR}(t_2)[T_T]$	see above
Γ	$\vdash T:$	$\text{CAPR}(t_T)[T_T]$	by hypothesis
$\Rightarrow\Gamma$	$\vdash n[R\|S]\|T:$	$\text{CAPR}(t_2+t_T)[T_T]$	CA-PAR

			Typing B
Γ	$\vdash n[R\|S]\|T:$	$\text{CAPR}(t_2+t_T)[T_T]$	see above
Γ	$\vdash m[P\|Q]$	$\text{CAPR}(e_m)[T_T]$	see above
$\Rightarrow\Gamma$	$\vdash B:$	$\text{CAPR}(t_2+t_T+e_m)[T_T]$	CA-PAR

Case R-open. If $A \longrightarrow B$ by one step of R-OPEN, we have

$$\begin{cases} A &= h[\mathsf{open}\ m.P \mid Q \mid m[\overline{\mathsf{open}}\ \{m,h\}.R \mid S]] \\ B &= h[P \mid Q \mid R \mid S]\,. \end{cases}$$

Let Γ be an environment such that

$$\begin{cases} \Gamma(m) &= \mathrm{CA}_{\mathrm{AM}}(s_m, e_m)[T_m] \\ \Gamma(h) &= \mathrm{CA}_{\mathrm{AM}}(s_h, e_h)[T_h] \end{cases}$$

$$\begin{cases} \Gamma \vdash P : \mathrm{CA}_{\mathrm{PR}}(t_P)[T_P] \\ \Gamma \vdash Q : \mathrm{CA}_{\mathrm{PR}}(t_Q)[T_Q] \\ \Gamma \vdash R : \mathrm{CA}_{\mathrm{PR}}(t_R)[T_R] \\ \Gamma \vdash S : \mathrm{CA}_{\mathrm{PR}}(t_S)[T_S]. \end{cases}$$

This environment is generic and represents the general case. Let us follow the only derivation that may type A in Γ:

			Typing $\overline{\mathsf{open}}\ \{m,h\}.R$
Γ	$\vdash R:$	$\mathrm{CA}_{\mathrm{PR}}(t_R)[T_R]$	by hypothesis
Γ	$\vdash m:$	$\mathrm{CA}_{\mathrm{AM}}(s_m, e_m)[T_m]$	by hypothesis
$\Rightarrow \Gamma$	$\vdash \overline{\mathsf{open}}\ \{m,h\}.R:$	$\mathrm{CA}_{\mathrm{PR}}(t_R)[T_R]$	CA-COOPEN
	where	$T_m = T_R$	

			Typing $\overline{\mathsf{open}}\ \{m,h\}.R \mid S$
Γ	$\vdash \overline{\mathsf{open}}\ \{m,h\}.R:$	$\mathrm{CA}_{\mathrm{PR}}(t_R)[T_R]$	see above
Γ	$\vdash S:$	$\mathrm{CA}_{\mathrm{PR}}(t_S)[T_S]$	by hypothesis
$\Rightarrow \Gamma$	$\vdash \overline{\mathsf{open}}\ \{m,h\}.R \mid S:$	$\mathrm{CA}_{\mathrm{PR}}(t_R + t_S)[T_R]$	CA-PAR
	where	$T_R = T_S$	

			Typing $m[\overline{\mathsf{open}}\ \{m,h\}.R \mid S]$
Γ	$\vdash \overline{\mathsf{open}}\ \{m,h\}.R \mid S:$	$\mathrm{CA}_{\mathrm{PR}}(t_R + t_S)[T_R]$	see above
Γ	$\vdash m:$	$\mathrm{CA}_{\mathrm{AM}}(s_m, e_m)[T_m]$	by hypothesis
$\Rightarrow \Gamma$	$\vdash m[\cdots]:$	$\mathrm{CA}_{\mathrm{PR}}(t_1)[T_1]$	CA-AMB
	where	$t_R + t_S \leq s_m$	
		$e_m \leq t_1$	

			Typing $\mathsf{open}\ m.P$
Γ	$\vdash P:$	$\mathrm{CA}_{\mathrm{PR}}(t_P)[T_P]$	by hypothesis
Γ	$\vdash m:$	$\mathrm{CA}_{\mathrm{AM}}(s_m, e_m)[T_m]$	by hypothesis
$\Rightarrow \Gamma$	$\vdash \mathsf{open}\ m.P:$	$\mathrm{CA}_{\mathrm{PR}}(t_P - e_m + s_m)[T_P]$	CA-OPEN
	where	$T_m = T_P$	
		$t_P - e_m + s_m \geq 0$	

			Typing $\mathsf{open}\ m.P \mid Q$
Γ	$\vdash \mathsf{open}\ m.P:$	$\mathrm{CA}_{\mathrm{PR}}(t_P - e_m + s_m)[T_P]$	see above
Γ	$\vdash Q:$	$\mathrm{CA}_{\mathrm{PR}}(t_Q)[T_Q]$	by hypothesis
$\Rightarrow \Gamma$	$\vdash \mathsf{open}\ m.P \mid Q:$	$\mathrm{CA}_{\mathrm{PR}}(t_P + t_Q - e_m + s_m)[T_P]$	CA-PAR
	where	$T_P = T_Q$	

			Typing $\mathsf{open}\ m.P \mid Q \mid m[\cdots]$
Γ	$\vdash \mathsf{open}\ m.P \mid Q:$	$\mathrm{CA}_{\mathrm{PR}}(t_P + t_Q - e_m + s_m)[T_P]$	see above
Γ	$\vdash m[\cdots]:$	$\mathrm{CA}_{\mathrm{PR}}(t_1)[T_1]$	see above
$\Rightarrow \Gamma$	$\vdash \mathsf{open}\ m.P \mid Q \mid m[\cdots]:$	$\mathrm{CA}_{\mathrm{PR}}(t_0)[T_P]$	CA-PAR
	where	$T_P = T_1$	
		$t_0 = t_P + t_Q - e_m + s_m + t_1$	

			Typing A
Γ	$\vdash \mathsf{open}\ m.P$		
	$\mid Q \mid m[\cdots]:$	$\mathrm{CA}_{\mathrm{PR}}(t_0)[T_P]$	see above
	where	$t_0 = t_P + t_Q - e_m + s_m + t_1$	
Γ	$\vdash h:$	$\mathrm{CA}_{\mathrm{AM}}(s_h, e_h)[T_h]$	by hypothesis
$\Rightarrow \Gamma$	$\vdash A:$	$\mathrm{CA}_{\mathrm{PR}}(t_2)[T_2]$	CA-AMB
	where	$e_h \leq t_2$	
		$T_P = T_h$	
		$t_P + t_Q - e_m + s_m + t_1 \leq s_h$	

From these conditions, we deduce: $T_P = T_Q = T_h = T_R = T_S$. Then, we can easily type the following process with multiple applications of CA-PAR:

$$\Gamma \vdash P \mid Q \mid R \mid S : \mathrm{CA}_{\mathrm{PR}}(t_P + t_Q + t_R + t_S)[T_h]\,.$$

Using the previous conditions, we find:

$$\begin{aligned} t_P + t_Q + t_R + t_S &\leq t_P + t_Q + s_m \leq t_P + t_Q + s_m + t_1 - e_m \\ &\leq s_h\,. \end{aligned}$$

Finally, we can apply CA-AMB and obtain the typing:

$$\Gamma \vdash B : \mathrm{CA}_{\mathrm{PR}}(t_2)[T_2]\,.$$

Case R-msg. If $A \longrightarrow B$ by one step of R-MSG, we have

$$\begin{cases} A &= \langle n \rangle \mid (x:N)P \\ B &= P\{x \leftarrow n\}\,. \end{cases}$$

Let Γ be an environment such that

$$\begin{cases} \Gamma(n) &= A_n \\ \Gamma, x:N \ \vdash & P : \mathrm{CA}_{\mathrm{PR}}(t_P)[T_P]. \end{cases}$$

This environment is generic and represents the general case. Let us follow the only derivation that may type A in Γ:

			Typing $(x:N)P$
$\Gamma, x:N$	$\vdash P:$	$\mathrm{CA}_{\mathrm{PR}}(t_P)[T_P]$	by hypothesis
$\Rightarrow \Gamma$	$\vdash (x:N)P:$	$\mathrm{CA}_{\mathrm{PR}}(t_2)[t_P, N]$	CA-RECEIVE
	where	$T_P = t_P$	

			Typing $\langle n \rangle$
$\Rightarrow \Gamma$	$\vdash \langle n \rangle:$	$\mathrm{CA}_{\mathrm{PR}}(t_1)[t, A_n]$	CA-SEND
	where	$t_1 \geq t$	

			Typing A
Γ	$\vdash (x:N)P:$	$\mathrm{CA}_{\mathrm{PR}}(t_2)[t_P, N]$	see above
Γ	$\vdash \langle n \rangle:$	$\mathrm{CA}_{\mathrm{PR}}(t_1)[t, A_n]$	see above
$\Rightarrow \Gamma$	$\vdash A:$	$\mathrm{CA}_{\mathrm{PR}}(t_1 + t_2)[t_P, A_n]$	CA-PAR
	where	$N = A_n$	
		$t = t_P$	

With the above conditions and by hypothesis, we have

$$\Gamma, x:A_n \vdash P : \mathrm{CA}_{\mathrm{PR}}(t_P)[t_P, A_n]$$

and $\Gamma \vdash n : A_n$. Using Lemma 8, we get:

$$\Gamma \vdash P\{x \leftarrow n\} : \mathrm{CA}_{\mathrm{PR}}(t_P)[t_P, A_n]\,.$$

Then, since $t_P = t \leq t_1 \leq t_1 + t_2$, we can apply Lemma 1 and obtain

$$\Gamma \vdash B : \text{CAPR}(t_1 + t_2)[t_P, A_n] .$$

Case R-rec. If $A \longrightarrow B$ by one step of R-REC, we have

$$\begin{cases} A &= \mathsf{rec}\ X.P \\ B &= P\{X \leftarrow \mathsf{rec}\ X.P\} . \end{cases}$$

The typing $\Gamma \vdash A : \text{CAPR}(t')[T]$ must have been derived from

$$\Gamma, X : \text{CAPR}(t)[T] \vdash P : \text{CAPR}(t)[T]$$

with $t' \geq t$. Using the same rule, we can also conclude that $\Gamma \vdash \mathsf{rec}\ X.P : \text{CAPR}(t)[T]$. By Lemma 7, we have

$$\Gamma \vdash P\{X \leftarrow \mathsf{rec}\ X.P\} : \text{CAPR}(t)[T] .$$

And finally, we get $\Gamma \vdash B : \text{CAPR}(t')[T]$ by Lemma 1.

Case R-res. If $A \longrightarrow B$ by one step of R-RES, we have $A = (\nu n : N)P$ and $B = (\nu n : N)Q$, where $P \longrightarrow Q$. Let us note $\Delta = \Gamma, n : N$. Since A may be typed in Γ, we easily find out that P may be typed in Δ with type U.

By the induction hypothesis, we have $\Delta \vdash Q : U$. Hence, by CA-RES, we conclude $\Gamma \vdash B : U$.

Case R-par. If $A \longrightarrow B$ by one step of R-PAR, we have $A = P|Q$ and $B = P|R$, where $Q \longrightarrow R$. Since A may be typed in Γ, so do P and Q. Necessarily, we have the following typings:

$$\Gamma \vdash A : \text{CAPR}(t_P + t_Q)[T]$$
$$\Gamma \vdash P : \text{CAPR}(t_P)[T]$$
$$\Gamma \vdash Q : \text{CAPR}(t_Q)[T].$$

By the induction hypothesis, we have $\Gamma \vdash R : \text{CAPR}(t_Q)[T]$. Finally, by CA-PAR, we can conclude $\Gamma \vdash B : \text{CAPR}(t_P + t_Q)[T]$.

Case R-amb. If $A \longrightarrow B$ by one step of R-AMB, we have $A = m[P]$ and $B = m[Q]$ where $P \longrightarrow Q$. Since A may be typed in Γ with type $\text{CAPR}(t)[T]$, so does P with type $\text{CAPR}(t_P)[T_m]$, and m using the type $\text{CAAM}(s_m, e_m)[T_m]$. Additionally, we have $t_P \leq s_m$ and $e_m \leq t$.

By the induction hypothesis, since $P \longrightarrow Q$, we also have that $\Gamma \vdash Q : \text{CAPR}(t_P)[T_m]$. Since $t_P \leq s_m$ and $e_m \leq t$, we may once again use CA-AMB. We then conclude that $\Gamma \vdash B : \text{CAPR}(t)[T]$.

Case R-$\equiv$. This case is trivial, by the induction hypothesis and using Lemma 6 twice.

□