Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON
n° 5668

Using Ambients to Control Resources

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

[1] LIP - ENS Lyon, France
[2] INRIA Sophia Antipolis, France

April 2002

Research Report N° 2002-16

École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : `lip@ens-lyon.fr`

Using Ambients to Control Resources

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

[1] LIP - ENS Lyon, France
[2] INRIA Sophia Antipolis, France

April 2002

Abstract

Current software and hardware systems, being parallel and reconfigurable, raise new safety and fiability problems, and the resolution of these problems requires new methods. Numerous proposals attempt at reducing the threat of bugs and preventing Denial of Service attacks. In this paper, we develop an extension of the calculus of Mobile Ambients, named *Controlled Ambients*, that is suited for expressing such issues. We present a type system for Controlled Ambients, which makes resource control possible in our setting.

Keywords: Distributed and mobile systems, resource control, process algebra, type system, Mobile Ambients

Résumé

Les systèmes actuels, reconfigurables et parallèles, soulèvent des questions nouvelles en termes de sûreté et de fiabilité. De nombreuses approches sont proposées afin de réduire le risque de bugs et se prévenir contre les attaques de type déni de service. Nous étendons le calcul des Mobile Ambients en un formalisme, les *Ambients Contrôlés*, dans lequel exprimer de manière formelle ce genre de problématique. Nous définissons également un système de types pour notre langage, permettant de garantir de manière statique une propriété de contrôle de ressources.

Mots-clés: Systèmes distribués et mobiles, contrôle de ressources, algèbre de processus, système de types, Ambients mobiles

Introduction

The latest generation of computer software and hardware makes use of numerous new technologies in order to enhance flexibility or performances. 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.

Active networks, for instance, attempt to introduce some form of sophistication within routing, by making networks programmable. However, until a control on used resources is developped, possibly using PCC frameworks, such systems will remain very sensitive to Denial of Service-like attacks or bugs.

In this paper, we present a formalism for *resource control* in parallel, distributed, mobile systems. By resource, we mean an entity, such as RAM, which may at will be acquired, used, then released. We present a method based on Ambient Calculi [2], and extending Safe Ambients [14]; we introduce these *Controlled Ambients*, and equip this language with a type system for resource control.

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, and show that a specific calculus should be designed for the purpose of guaranteeing some control on the use of resources. In Section 2, we introduce our calculus of Controlled Ambients and explain why it fits to our purposes. We then develop in Section 3 a type system which uses the specifics of this language to make resource control possible; we prove its correctness (i.e. that it does control the acquisition and release of resources), and use it to treat several examples. After this, we discuss some refinements of our type system, and, in the last section, we present possible extensions of this study as well as related works.

1 Resource control

To be general, let us define a resource as an entity which may *at will* be acquired, used, then released. Thus, this notion 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 requirement, 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. Ambient calculi can be used for these purposes.

Ambient Calculi. Ambient Calculi are based on locality: each *ambient* is a site. In turn, any ambient may contain subambients, as well as *process*es, controlling its behaviour through the use of *capabilities*. Capabilities make the

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

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

The city

$$city \triangleq city[cab \mid cab \mid cab \mid \cdots \mid site_1[client\ site_1 site_i \mid client\ site_1\ site_2 \mid \cdots] \mid \cdots \mid site_i[\cdots]]$$

Figure 1: Cab protocol - first attempt

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 sets its contents free. To draw some analogies with real systems, in and out 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 ...

This set of correspondences opens a way for a natural model of resource control: each site may have a finite (or infinite) quantity of resources of a given category. These resources will be used for data, programs, ... 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) which may be present in one ambient at any time.*

Do note that we could have chosen dual points of view and decided to count all subambients at all depths, or possibly only "leaf" ambients. Although these approaches seems equally valid, we have decided not to undertake them, since they did not seem more powerful, only slightly more complicated.

An example. Let us consider a cab protocol, as shown in Figure 1, which we will use as our main running example. The system consists of one city, n sites, and many 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. If a cab is available, one (and only one) cab must come fetch the client and bring her to her destination. 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 original 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. 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 which hosts the *client* and *to* must be the name of some site.

In order 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. 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 about 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 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 which 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.

Towards a better control. Difficulties with security and control are due, for the greatest part, to the nature of capabilities in, out and open. Actually, the way these capabilities are used seems too simple: 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 the opening of a program. In practice, if a program wishes to receive network information, it must first "listen" on some

[1] As a matter of fact, we are not exactly using the original MA calculus, since we have introduced the rec operator, for more readability.

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 [14]; in this calculus of *Safe Ambients*, three *cocapabilities* are introduced, which we will note $\overline{\mathsf{SAin}}$, $\overline{\mathsf{SAout}}$ and $\overline{\mathsf{SAopen}}$. When executed in m, capability $\overline{\mathsf{SAin}}\ m$ allows an ambient to enter m (when executing 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. Thinking of the example above, a rewritten cab may thus easily refuse entry right 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 a full ambient 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 $h[m[n[\mathsf{out}\ m] \mid \overline{\mathsf{SAout}}\ m]]$, 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.

The system presented in [10] offers 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. The formalism we present in Section 2 may be considered as a development of [10] towards even more robustness as well as resource control. Let us also mention [15], where a different mechanism for the $\overline{\mathsf{SAout}}$ cocapability w.r.t. [14] is introduced. Our proposal actually subsumes the solutions of [15] and [14].

Types. Several studies of ambients, such as [3], introduce type systems, possibly linked to modifications in Mobile Ambients, such as [14], in order to control interactions. For example, this may be useful in the cab protocol to prevent a malicious *exited* from modifying the structure of the system. In Section 3, we propose a type system associated to our formalism, to perform resource control. Basically, the type of an ambient carries two informations:

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

The goal of the type system is to statically divide the available resources between parallel processes, and check that they will not be affected by movements and openings of ambients.

2 The Language of Controlled Ambients

In order to be able to control resources, we must ensure a form of movement control, to be able to find out which entities are using which resources at a given site. In this section, we present the language of *Controlled Ambients* (CA), which has been designed for these purposes.

2.1 Syntax and Semantics

In CA, each movement is subject to a 3-way synchronization between the moving ambient, the ambient welcoming a new subambient and the ambient letting a subambient go. As for the opening of an ambient, it is subject to some synchronization between the opener and the ambient being opened. These synchronizations are handled using *cocapabilities*: $\overline{\mathsf{in}_\uparrow}$, $\overline{\mathsf{out}_\uparrow}$, $\overline{\mathsf{in}_\downarrow}$, $\overline{\mathsf{out}_\downarrow}$ and $\overline{\mathsf{open}}$.

$\overline{\mathbf{in}_\uparrow}\ m$ the *up coentry*, allows m to enter the current ambient by exiting some subambient;

$\overline{\mathbf{in}_\downarrow}\ m$ the *down coentry*, allows m to enter the current ambient from its parent ambient;

$\overline{\mathbf{out}_\uparrow}\ m$ the *up coexit*, allows m to leave the current ambient by exiting it;

$\overline{\mathbf{out}_\downarrow}\ m$ the *down coexit*, allows m to leave the current ambient by entering one of its subambients;

$\overline{\mathbf{open}}\ \{m, h\}$ the *coopening*, allows the parent ambient h to open the current ambient m.

Just like the corresponding capabilities, cocapabilities are consumed when executed.

Do note that $\uparrow$ and $\downarrow$ are not necessary for resource control. We added them since we found they ease the task of specification in mobile ambients. We will return on the use of these annotations in Section 2.3.

The syntax of Controlled Ambients is presented in Figure 2. We suppose we have two infinite sets of term variables, ranged over with capital letters (X, Y), and of names, ranged over with small letters $(m, n, h, x, \dots)$. Name binders (input and restriction) are decorated with some type information, that shall be made explicit in the next section. While several proposals for Mobile Ambient calculi use replication, infinite behaviour 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 we 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. $P|Q$ 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 occurences 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 is defined in two steps: structural congruence, written $\equiv$, is the least congruence relation statisfying the laws of Figure 3; reduction, written $\longrightarrow$, is then introduced in Figure 4. We let $\longrightarrow^*$ stand for the reflexive transitive closure of $\longrightarrow$.

$$
\begin{array}{lcll}
P, Q, R & ::= & \mathbf{0} & \text{null process} \\
 & \mid & M.Q & \text{adding a capability to a process} \\
 & \mid & m[Q] & \text{formation of ambient } m \\
 & \mid & Q \mid R & \text{parallel composition of processes} \\
 & \mid & (\nu n : A)Q & \text{generation of a fresh name } n \text{ with type annotation } A \\
 & \mid & \mathsf{rec}\ X.Q & \text{recursive construction} \\
 & \mid & X & \text{process variable} \\
 & \mid & (n : A)Q & \text{abstraction (message reception) with type annotation} \\
 & \mid & \langle m \rangle & \text{message emission} \\
 & & & \\
M & ::= & \mathsf{in}\ m & \text{enter } m \\
 & \mid & \mathsf{out}\ m & \text{leave } m \\
 & \mid & \mathsf{open}\ m, h & \text{open } m \\
 & \mid & \overline{\mathsf{in}_\uparrow}\ m & m \text{ may climb in upwards} \\
 & \mid & \overline{\mathsf{in}_\downarrow}\ m & m \text{ may climb in downwards} \\
 & \mid & \overline{\mathsf{out}_\uparrow}\ m & m \text{ may climb out upwards} \\
 & \mid & \overline{\mathsf{out}_\downarrow}\ m & m \text{ may climb out downwards} \\
 & \mid & \overline{\mathsf{open}}\ \{m, h\} & h \text{ may open } m
\end{array}
$$

Figure 2: Controlled Ambients - syntax

$$
\begin{array}{llll}
P \equiv P|\mathbf{0} & S-parnil & (\nu n : A)\mathbf{0} \equiv \mathbf{0} & S-resnil \\
P|Q \equiv Q|P & S-parcomm & P|(Q|R) \equiv (P|Q)|R & S-parass
\end{array}
$$

$$
\begin{array}{ll}
(\nu n : A)(P|Q) \equiv ((\nu n : A)P)|Q \text{ if } n \notin fn(Q) & S-respar \\
(\nu n : A)(\nu m : B)P \equiv (\nu m : B)(\nu n : A)P & S-resc \\
(\nu n : A)m[P] \equiv m[(\nu n : A)P] \text{ if } n \neq m & S-resamb
\end{array}
$$

Figure 3: Controlled Ambients - Structural Congruence

2.2 Examples

We now provide a few examples to illustrate the use of Controlled Ambients. We omit for the moment type annotations in restrictions; these will be made explicit in the next section.

Renaming. Since movements in Controlled Ambients require some knowledge about the name of moving ambients (also in cocapabilites, which is not the case in Safe Ambients), renaming may 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.

$$
\begin{array}{rcll}
m[\mathsf{in}\ n.P \mid Q] \mid n[\overline{\mathsf{in}_\downarrow}\ m.R \mid S] \mid \overline{\mathsf{out}_\downarrow}\ m.T & \longrightarrow & n[m[P \mid Q] \mid R \mid S] \mid T & (R-in)\\
n[m[\mathsf{out}\ n.P \mid Q] \mid \overline{\mathsf{out}_\uparrow}\ m.R \mid S] \mid \overline{\mathsf{in}_\uparrow}\ m.T & \longrightarrow & m[P \mid Q] \mid n[R \mid S] \mid T & (R-out)\\
h[\mathsf{open}\ m.P \mid Q \mid m[\overline{\mathsf{open}}\ \{m,h\}.R \mid S]] & \longrightarrow & h[P \mid Q \mid R \mid S] & (R-open)\\
\langle n\rangle \mid (x:A)P & \longrightarrow & P\{x \leftarrow n\} & (R-msg)\\
\mathsf{rec}\ X.P & \longrightarrow & P\{X \leftarrow \mathsf{rec}\ X.P\} & (R-rec)
\end{array}
$$

$$
\frac{P \longrightarrow Q}{(\nu n:A)P \longrightarrow (\nu n:A)Q}\ (R-res) \qquad \frac{Q \longrightarrow R}{P|Q \longrightarrow P|R}\ (R-par)
$$

$$
\frac{P \longrightarrow Q}{n[P] \longrightarrow n[Q]}\ (R-amb) \qquad \frac{P \equiv Q \quad Q \longrightarrow R \quad R \equiv S}{P \longrightarrow S}\ (R-\equiv)
$$

Figure 4: Controlled Ambients - Reduction

Safe Ambients Cocapabilities. As mentioned earlier, Safe Ambients [14] 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

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

By carrying on the idea behind renaming, we can approximate the working 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

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

As specified, this expression reduces to $b[R \mid S \mid a[P \mid Q]]$. Again, just as was the case for renaming, the father must accept the transaction with $\overline{\mathsf{in}_\uparrow}\ m.\overline{\mathsf{out}_\downarrow}\ m.\overline{\mathsf{out}_\downarrow}\ a$. Hence, in order to accept this transaction, the father ambient must know the existence of a. This is coherent with the spirit of CA: cocapabilities allow each ambient to know its children at any time.

Firewall. We revisit the firewall example of [2], and consider a system f, protected by a firewall. Only agents aware of the password g are allowed in f. This may be modeled as:

$$
\begin{array}{lcl}
\mathit{Agent}\ P\ Q & \triangleq & \mathit{agent}[\mathsf{in}\ g.\overline{\mathsf{in}_\downarrow}\ \mathit{entered}.\mathsf{open}\ \mathit{entered}.P \mid Q]\\
\mathit{System} & \triangleq & (\nu f)f[\mathsf{rec}\ X.(g[\mathsf{out}\ f.\overline{\mathsf{in}_\downarrow}\ \mathit{agent}.\mathsf{in}\ f.\overline{\mathsf{open}}\ \{g,f\}]\\
& & \quad \mid \overline{\mathsf{out}_\uparrow}\, g.\overline{\mathsf{in}_\downarrow}\, g.\mathsf{open}\ g.(\mathit{entered}[\mathsf{in}\ \mathit{agent}.\overline{\mathsf{open}}\ \{\mathit{entered},\mathit{agent}\}]\\
& & \quad \mid \overline{\mathsf{out}_\uparrow}\ \mathit{entered}.X))]\\
& & \quad \mid \mathsf{rec}\ Y.\overline{\mathsf{in}_\uparrow}\ g.\overline{\mathsf{out}_\downarrow}\ \mathit{agent}.\overline{\mathsf{out}_\downarrow}\ g.Y
\end{array}
$$

This system implements two authentifications: in the first place, the Agent must be named *agent* - it will not enter f by accident. In the second place, it must know the password. Note that this is not the Firewall described in the original paper on Mobile Ambients [2], which relied on the secrecy of three keys.

This version uses only one key and takes advantage of the synchronization mechanism to execute correctly. The basic ideas are the following: *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 *agents*.

Method calls. The semantics of cocapabilities $\overline{\mathsf{in}}$ and $\overline{\mathsf{out}}$ allows us to draw some analogies: a process of the form $m[\overline{\mathsf{in}_\downarrow}\ a.P \mid \overline{\mathsf{in}_\downarrow}\ b.Q]$ accepts only incoming ambients named a or b, and may react immediately - and differently - upon arrival of these ambients. This may be used to represent a system with two ports or a program with two entry points a and b.

This mechanism is also somehow reminiscent of method calls in (concurrent) object-oriented languages: an ambient may be seen as an object offering several methods, say $m_1, \ldots, m_k$, which can be triggered upon entrance of an ambient having name m_i for $i \in [1 \ldots k]$ (corresopnding to the execution of cocapability $\overline{\mathsf{in}}\ m_i$). A logo-like turtle object would be implemented as follows:

$$turtle[\overline{\mathsf{in}}\ \mathtt{turn\_left}.P_1 \mid \overline{\mathsf{in}}\ \mathtt{turn\_right}.P_2 \mid \overline{\mathsf{in}}\ \mathtt{step\_forward}.P_3 \mid \ldots]$$

Of course, this observation just suggests an analogy; a real study of the encoding of object-orientation in Controlled Ambients goes beyond the scope of this paper.

Cab. As shown in Figure 5, the cab protocol may be rewritten so as to take advantage of Controlled Ambients. We do not present the new version of the city itself or of the sites, which only need to contain all authorizations to move in or out, in addition to *clients* and *cabs*.

Thanks to the cocapabilities, synchronizations in CA are both easier than those of Mobile Ambients and atomic. Additionally, the system is not subject to the interferences we have presented: only *clients* may enter the cab, not just any "parasite" ambient which happens to contain capability $\mathsf{in}\ cab$. Similarly, sites only welcome *clients*, *cabs* and *calls*.

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

Additionally, as was planned, Controlled Ambients permit the control of resources such as available space in cabs. As opposed to the Mobile Ambients version, we may easily check that the cab may contain at most *only one passenger* and possibly an auxiliary ambient *call*, *trip*, *arrived* or *end*. This will be expressed formally using our type system in Section 3.

2.3 Benefits

We now make a few more comments on the definition of Controlled Ambients, stressing some aspects we have seen on the previous examples.

The formalism of Controlled Ambients is more reasonable than Mobile Ambients or Safe Ambients. More reasonable insofar as the implementation of movements in ambient calculi suggests this kind of three-way synchronization. Let us consider the following transition in Mobile Ambients:

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

Message emitted by client

$$\begin{array}{lcl}
\mathit{call}\ \mathit{from} & \triangleq & \mathit{call}[\mathsf{out}\ \mathit{client}.\mathsf{out}\ \mathit{from}. \\
& & \qquad \mathsf{in}\ \mathit{cab}.\overline{\mathsf{open}}\ \{\mathit{call},\mathit{cab}\}.\mathsf{in}\ \mathit{from}.\overline{\mathsf{in}_{\downarrow}}\ \mathit{client}]
\end{array}$$

Instructions given by client

$$\begin{array}{lcl}
\mathit{trip}\ \mathit{from}\ \mathit{to} & \triangleq & \mathit{trip}[\mathsf{out}\ \mathit{client}.\overline{\mathsf{open}}\ \{\mathit{trip},\mathit{cab}\}.\mathsf{out}\ \mathit{from}. \\
& & \qquad \mathsf{in}\ \mathit{to}.\mathit{arrived}[\overline{\mathsf{open}}\ \{\mathit{arrived},\mathit{cab}\}. \\
& & \qquad\qquad \mathit{end}[\overline{\mathsf{open}}\ \{\mathit{end},\mathit{cab}\}.\mathsf{out}\ \mathit{to}]]]
\end{array}$$

The client

$$\begin{array}{lcl}
\mathit{client}\ \mathit{from}\ \mathit{to} & \triangleq & \mathit{client}[\mathit{call}\ \mathit{from} \mid \overline{\mathsf{out}_{\uparrow}}\ \mathit{call}.\mathsf{in}\ \mathit{cab}.\mathit{trip}\ \mathit{from}\ \mathit{to} \\
& & \quad \mid \overline{\mathsf{out}_{\uparrow}}\ \mathit{trip}.\mathsf{out}\ \mathit{cab}]
\end{array}$$

The cab

$$\begin{array}{lcl}
\mathit{cab} & \triangleq & \mathit{cab}[\mathsf{rec}\ X.\overline{\mathsf{in}_{\downarrow}}\ \mathit{call}.\mathsf{open}\ \mathit{call}.\overline{\mathsf{in}_{\uparrow}}\ \mathit{trip}.\mathsf{open}\ \mathit{trip}. \\
& & \qquad \mathsf{open}\ \mathit{arrived}.\overline{\mathsf{out}_{\uparrow}}\ \mathit{client}.\mathsf{open}\ \mathit{end}.X]
\end{array}$$

Figure 5: Cab protocol – CA-style

As shown in [8, 18], 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 rule will need a synchronization between n (who is present), m (who looks for n) and h (who knows about 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 in h and update presence registers of h.

Controlled Ambients are also more realistic as modeling tools. When a system receives informations, 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 ... Unfortunately, this listening aspect is not rendered at all by Mobile Ambients and only in half of the cases by Safe Ambients. Similarly, a system must be able to wait for several kinds of informations 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. As opposed to these two formalisms, and as shown above, Controlled Ambients may naturally render this *port listening* aspect, as well some form of method invocation, or of pattern matching. Of course, through renaming and infinite loops of cocapacities, we may also model situations where some part of the system (i.e. the network connexion itself) accepts data without listening for it.

3 Typing Controlled Ambients

This section is devoted to the presentation of a basic 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 capture on some examples.

3.1 The Type System

Types and Type Judgments. The grammar for types is given in Figure 6, and includes entries for the types of ambients, processes and messages (we let $\overline{\mathbb{N}}$ stand for $\mathbb{N} \cup \{\infty\}$).

$$
\begin{array}{rcllr}
A & ::= & \mathrm{CAA{\small MB}}(s,e)[T] & s \in \overline{\mathbb{N}}, e \in \mathbb{N} & \text{ambients} \\
U & ::= & \mathrm{CAP{\small ROC}}(t)[T] & t \in \overline{\mathbb{N}} & \text{processes} \\
T & ::= & Ssh & & \text{messages} \\
 & | & t, A & t \in \overline{\mathbb{N}} &
\end{array}
$$

Figure 6: Types

The typing judgments are defined using *environments,* ranged over with Γ, which 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) to 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

$$\Gamma \vdash n : \mathrm{CAA{\small MB}}(s,e)[T]\,,$$

and expresses the fact that under assumptions Γ, n is the name of an ambient of *capacity* s, *weight* e, and within which messages carrying information of type T may be exchanged. The capacity s represents the number of resource units (*resources* for short) that are available within n, while e is the number of resources this ambient is using in its surrounding ambient. Note that while an ambient may have an infinite capacity ($s = \infty$), it cannot manipulate infinitely many resources ($e < \infty$). The type T for messages captures the kind of names being exchanged within n, similarly to Cardelli and Gordon's *topics of conversation* [3], augmented with an information t which represents a higher bound on the effect of exchanging messages within n (we shall come back to this below).

The typing judgment for processes is written

$$\Gamma \vdash P : \mathrm{CAP{\small ROC}}(t)[T]\,,$$

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.

Typing Rules. The only typing rule for ambient names is straightforward:

$$\frac{\Gamma(n) = \mathrm{CAA{\small MB}}(s,e)[T]}{\Gamma \vdash n : \mathrm{CAA{\small MB}}(s,e)[T]}\ CA - name$$

Let us now examine the typing rules for terms. Typing cocapabilities just amounts to express the meaning of types, as introduced above:

$$\frac{\Gamma \vdash P : \mathrm{CAP{\small ROC}}(t)[T] \quad \Gamma \vdash m : \mathrm{CAA{\small MB}}(s,e)[T']}{\Gamma \vdash \overline{\mathsf{in}_\delta}\ m.P : \mathrm{CAP{\small ROC}}(t+e)[T]}\ CA - coin$$

$$\frac{\Gamma \vdash P : \mathrm{CAP{\small ROC}}(t)[T] \quad \Gamma \vdash m : \mathrm{CAA{\small MB}}(s,e)[T']}{\Gamma \vdash \overline{\mathsf{out}_\delta}\ m.P : \mathrm{CAP{\small ROC}}(t-e)[T]}\ \begin{array}{c} CA - coout \\ t \geq e \end{array}$$

Here δ ranges over a direction tag, which can be $\uparrow$ or $\downarrow$. Exercing a $\overline{\mathsf{in}}\ m$ capability has the effect of acquiring an amount of resources equal to m's weight in the current ambient. Similarly, letting ambient m leave releases some resources. Note however that the number t of resources allocated to the process must remain positive after decreasing. This is made possible by the subtyping property of the system (Lemma 1). In particular, the typing rule for $\mathbf{0}$ allows one to allocate any number of resources for the inactive process:

$$\frac{}{\Gamma \vdash \mathbf{0} : \mathrm{CAP}\textsc{roc}(t)[T]}\ CA - nil$$

Thus, to type a term whose action is just to let ambients exit the current ambient (using $\overline{\mathsf{out}}$ capabilities), we have to allocate enough resources for the occurrences of the terminal $\mathbf{0}$ process. In this sense, information t in the type of a process P is not really in this case a measure of the *effect* of P from the point of view of resource usage, but rather represents some kind of *resource allocation* for P.

Exercing movement capabilities does not change anything in the current ambient from the point of view of resource usage:

$$\frac{\Gamma \vdash P : \mathrm{CAP}\textsc{roc}(t)[T]}{\Gamma \vdash \mathsf{in}\ m.P : \mathrm{CAP}\textsc{roc}(t)[T]}\ CA - in$$

$$\frac{\Gamma \vdash P : \mathrm{CAP}\textsc{roc}(t)[T]}{\Gamma \vdash \mathsf{out}\ m.P : \mathrm{CAP}\textsc{roc}(t)[T]}\ CA - out$$

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):

$$\frac{\Gamma \vdash m : \mathrm{CAA}\textsc{mb}(s,e)[T] \quad \Gamma \vdash P : \mathrm{CAP}\textsc{roc}(t)[T]}{\Gamma \vdash \mathsf{open}\ m.P : \mathrm{CAP}\textsc{roc}(t-e+s)[T]}\ \begin{array}{c} CA - open \\ t-e+s \geq 0 \end{array}$$

The $\overline{\mathsf{open}}$ capability plays no role from the point of view of resource control, as illustrated by the rule below (note, still, that message types in the opening ambient and in the type of R are unified using this rule). We shall present in Section 4 more refined systems where a more precise typing of opening permits a better control.

$$\frac{\Gamma \vdash m : \mathrm{CAA}\textsc{mb}(s,e)[T] \quad \Gamma \vdash R : \mathrm{CAP}\textsc{roc}(t)[T]}{\Gamma \vdash \overline{\mathsf{open}}\ \{m,h\}.R : \mathrm{CAP}\textsc{roc}(t)[T]}\ CA - coopen$$

When forming an ambient, we must check that the resource allocation for the contents is compatible with the ambient's capacity (condition $a \leq s$):

$$\frac{\Gamma \vdash m : \mathrm{CAA}\textsc{mb}(s,e)[T] \quad \Gamma \vdash P : \mathrm{CAP}\textsc{roc}(a)[T]}{\Gamma \vdash m[P] : \mathrm{CAP}\textsc{roc}(t)[T']}\ \begin{array}{c} CA - amb \\ a \leq s, e \leq t \end{array}$$

Here we allocate *at least* e resources for $m[P]$ (condition $e \leq t$): the ability to "waste resources" this way is of no harm, and is needed in order to guarantee a subtyping property (Lemma 1). A similar mechanism will be used in the typing rules for recursion and communication (see below).

The typing rule for parallel composition says that we have to split resources between the components.

$$\frac{\Gamma \vdash P : \text{CAP}\textsc{roc}(t)[T] \quad \Gamma \vdash Q : \text{CAP}\textsc{roc}(t')[T]}{\Gamma \vdash P|Q : \text{CAP}\textsc{roc}(t+t')[T]}\ CA-par$$

Creating a new name has no influence on the management of resources, as shown by the rule for restriction.

$$\frac{\Gamma, n : A \vdash P : U}{\Gamma \vdash (\nu n : A)P : U}\ CA-res$$

To type recursion, we must make sure that the process runs "in constant space", as expressed by rule $CA-rec$: any call to the loop will require the same amount of resources (t) as the whole term; this is expressed by adding hypothesis $X : \text{CAP}\textsc{roc}(t)[T]$ to Γ.

$$\frac{\Gamma(X) = \text{CAP}\textsc{roc}(t)[T]}{\Gamma \vdash X : \text{CAP}\textsc{roc}(t')[T]}\ \begin{matrix} CA-var-proc \\ t' \geq t \end{matrix}$$

$$\frac{\Gamma, X : \text{CAP}\textsc{roc}(t)[T] \vdash P : \text{CAP}\textsc{roc}(t)[T]}{\Gamma \vdash \mathsf{rec}\ X.P : \text{CAP}\textsc{roc}(t')[T]}\ \begin{matrix} CA-rec \\ t' \geq t \end{matrix}$$

We now explain the typing rules for communication. Since reception of a message can trigger a process which 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.

$$\frac{\Gamma \vdash m : A}{\Gamma \vdash \langle m \rangle : \text{CAP}\textsc{roc}(t')[t, A]}\ \begin{matrix} CA-send \\ t' \geq t \end{matrix}$$

$$\frac{\Gamma, x : A \vdash P : \text{CAP}\textsc{roc}(t)[t, A]}{\Gamma \vdash (x : A)P : \text{CAP}\textsc{roc}(t')[t, A]}\ CA-receive$$

In this approach, we assume that one emission typically corresponds to several receptions. The dual approach could have been used, by putting in correspondence one reception and several concurrent emissions. Our experience in writing examples suggests that our hypothesis is the most frequent one, not to mention that the dual option seems less friendly to other type systems such as Single-Threadedness Types [14].

3.2 Static Resource Control

We now present the main properties of the type system. We start by some technical properties about typing derivations.

Lemma 1 (Subtyping) *Let P be a process and Γ an environment. Then:*
if $\Gamma \vdash P : \text{CAP}\textsc{roc}(t)[T]$ *then* $\forall t' \geq t, \Gamma \vdash P : \text{CAP}\textsc{roc}(t')[T]$.

Proof: See Annex A.1. □

Corollary 2 (Minimal typing) *If a process P is typeable in Γ, there is a minimal $t \in \overline{\mathbb{N}}$ such that $\Gamma \vdash P : \mathrm{CAP}\textsc{roc}(t)[T]$.*

Note that the minimal parameter t can be different for each possible value T (see for example rule $CA - send$). Also note that not every term may be typed (taking $s = \infty$ for all ambients), because of the type constraints of the form $(\nu n : A)$.

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 3 (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 $\Gamma(a) = \mathrm{CAA}\textsc{mb}(s, e)[T]$, then $Res_\Gamma(a[P]) = e$;*
- *$Res_\Gamma(P_1|P_2) = Res_\Gamma(P_1) + Res_\Gamma(P_2)$;*
- *$Res_\Gamma((\nu n : A)\, P) = Res_{\Gamma,n:A}(P)$.*
- *in all other cases, $Res_\Gamma(P) = 0$.*

Note in particular that according to this definition, prefixed terms (capabilities, reception, recursion) do not take part in a process' *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 4 (Resource policy compliance) *Given a resource policy Γ, we define the judgment $\Gamma \models P$ (pronounced "P complies with Γ"), as follows:*

- *if $\Gamma \models n[P]$ iff $\Gamma \models P$ and there exists s, e, T such that $\Gamma(n) = \mathrm{CAA}\textsc{mb}(s, e)[T]$, and $Res_\Gamma(P) \leq s$;*
- *$\Gamma \models P_1|P_2$ 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, $\Gamma \models P$.*

The typing rules ensure that a typeable term complies with a resource policy:

Lemma 5 (Typeable terms comply with resource policies) *For any process P, resource policy Γ and process type U, if $\Gamma \vdash P : U$, then $\Gamma \models P$.*

Proof: See Annex A.3. □

The following theorem states that typability is preserved by the operational semantics of Controlled Ambients:

Theorem 6 (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: See Section A.7. $\square$

We finally come to our main result, which is a direct consequence of Lemma 5 and Theorem 6:

Theorem 7 (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

Renaming. As already mentioned, one possible expression of renaming is:

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

Let us try and type $a[a\ be\ b.P]$ in environment Γ such that

$$\begin{cases} \Gamma(a) & = & \mathrm{CAA{\small MB}}(s,e)[T] \\ \Gamma(b) & = & \mathrm{CAA{\small MB}}(s,e)[T] \\ \Gamma & \vdash & P : \mathrm{CAP{\small ROC}}(s)[T] \end{cases}$$

and $s \geq e$. The derivation appears on Figure 7. It shows that this renaming mechanism is compatible with the typing discipline we propose.

	Typing $b[\cdots]$	
$\Gamma \vdash \mathbf{0}$:	$\mathrm{CAP{\small ROC}}(0)[T]$	$CA - nil$
$\Gamma \vdash \mathsf{open}\ a$:	$\mathrm{CAP{\small ROC}}(s-e))[T]$	$CA - open$
$\Gamma \vdash \overline{\mathsf{in}_\downarrow}\ a.\mathsf{open}\ a$:	$\mathrm{CAP{\small ROC}}(s)[T]$	$CA - coin$
$\Gamma \vdash \mathsf{out}\ a.\overline{\mathsf{in}_\downarrow}\ a.\mathsf{open}\ a$:	$\mathrm{CAP{\small ROC}}(s)[T]$	$CA - out$
$\Gamma \vdash b[\mathsf{out}\ a.\overline{\mathsf{in}_\downarrow}\ a.\mathsf{open}\ a]$:	$\mathrm{CAP{\small ROC}}(e)[T]$	$CA - amb$
	Typing $a[\cdots]$	
$\Gamma \vdash P$:	$\mathrm{CAP{\small ROC}}(s)[T]$	by hypothesis
$\Gamma \vdash \overline{\mathsf{open}}\ \{a,b\}.P$:	$\mathrm{CAP{\small ROC}}(s)[T]$	$CA - coopen$
$\Gamma \vdash \mathsf{in}\ b.\overline{\mathsf{open}}\ \{a,b\}.P$:	$\mathrm{CAP{\small ROC}}(s)[T]$	$CA - in$
$\Gamma \vdash \overline{\mathsf{out}_\uparrow}\ b.\mathsf{in}\ b.\overline{\mathsf{open}}\ \{a,b\}.P$:	$\mathrm{CAP{\small ROC}}(s-e)[T]$	$CA - coout$
$\Gamma \vdash b[\cdots]$:	$\mathrm{CAP{\small ROC}}(e)[T]$	proved above
$\Gamma \vdash b[\cdots] \mid \overline{\mathsf{out}_\uparrow}\ b.\cdots$:	$\mathrm{CAP{\small ROC}}(s)[T]$	$CA - par$
$\Gamma \vdash a[\cdots]$:	$\mathrm{CAP{\small ROC}}(e)[U]$	$CA - amb$

Figure 7: Typing of renaming

We can actually slightly relax the conditions on types. One can show that the least set of conditions to type the renaming is:

$$\begin{cases} \Gamma(a) & = & \mathrm{CAA{\small MB}}(s_a,e_a)[T] \\ \Gamma(b) & = & \mathrm{CAA{\small MB}}(s_b,e_b)[T] \end{cases} \qquad \begin{cases} \Gamma \ \vdash \ P : \mathrm{CAP{\small ROC}}(t_P)[T] \\ t_P \leq s_a, e_b \leq s_a, s_a \leq s_b, e_a \leq s_b \end{cases}$$

Firewall. Let us recall one possible expression of a firewall:

Typing $g[\mathsf{out}\ f.\overline{\mathsf{in}_\downarrow}\ agent.\mathsf{in}\ f.\overline{\mathsf{open}}\ \{g,f\}]$		
$\Delta \vdash \mathsf{out}\ f.\overline{\mathsf{in}_\downarrow}\ agent.\ldots :$	$\mathrm{CAP}\textsc{roc}(1)[T]$	$CA - coin$
$\Delta \vdash g[\cdots] :$	$\mathrm{CAP}\textsc{roc}(0)[T]$	$CA - amb$
Typing $\overline{\mathsf{out}_\uparrow}\ g.\overline{\mathsf{in}_\downarrow}\ g.\mathsf{open}\ g.(entered[\cdots] \vert \overline{\mathsf{out}_\uparrow}\ entered.X)$		
$\Delta \vdash X :$	$\mathrm{CAP}\textsc{roc}(t)[T]$	$CA - var - proc$
$\Delta \vdash \overline{\mathsf{out}_\uparrow}\ entered.X :$	$\mathrm{CAP}\textsc{roc}(t)[T]$	$CA - coout$
$\Delta \vdash entered[\cdots] :$	$\mathrm{CAP}\textsc{roc}(0)[T]$	$CA - amb$
$\Delta \vdash entered[\cdots] \mid \overline{\mathsf{out}_\uparrow}\ entered.X :$	$\mathrm{CAP}\textsc{roc}(t)[T]$	$CA - par$
$\Delta \vdash \mathsf{open}\ g.(\cdots) :$	$\mathrm{CAP}\textsc{roc}(t+1)[T]$	$CA - open$
$\Delta \vdash \overline{\mathsf{in}_\downarrow}\ g.\mathsf{open}\ g, f.\cdots :$	$\mathrm{CAP}\textsc{roc}(t+2)[T]$	$CA - coin$
$\Delta \vdash \overline{\mathsf{out}_\uparrow}\ g.\overline{\mathsf{in}_\downarrow}\ g.\mathsf{open}\ g, f.\cdots :$	$\mathrm{CAP}\textsc{roc}(t+1)[T]$	$CA - coout$
$\Delta \vdash g[\cdots] \mid \overline{\mathsf{out}_\uparrow}\ g.\cdots :$	$\mathrm{CAP}\textsc{roc}(t+1)[T]$	$CA - par$
$\Gamma \vdash \mathsf{rec}\ X.\cdots :$	$\mathrm{CAP}\textsc{roc}(t)[T]\quad A = B$	$CA - rec$
Where $A = \mathrm{CAP}\textsc{roc}(t)[T]$ and $B = \mathrm{CAP}\textsc{roc}(t+1)[T]$		

Figure 8: Typing the firewall - simplified derivation

$$
\begin{array}{lcl}
Agent\ P\ Q & \triangleq & agent[\mathsf{in}\ g.\overline{\mathsf{in}_\downarrow}\ entered.\mathsf{open}\ entered.P \mid Q] \\
System & \triangleq & (\nu f : \mathrm{CAA}\textsc{mb}(\infty,0)[T]) \\
 & & f[\mathsf{rec}\ X.(g[\mathsf{out}\ f.\overline{\mathsf{in}_\downarrow}\ agent.\mathsf{in}\ f.\overline{\mathsf{open}}\ \{g,f\}] \\
 & & \quad \mid \overline{\mathsf{out}_\uparrow}\ g.\overline{\mathsf{in}_\downarrow}\ g.\mathsf{open}\ g.(entered[\mathsf{in}\ agent.\overline{\mathsf{open}}\ \{entered, agent\}] \\
 & & \quad \mid \overline{\mathsf{out}_\uparrow}\ entered.X))] \\
 & & \quad \mid \mathsf{rec}\ Y.\overline{\mathsf{in}_\uparrow}\ g.\overline{\mathsf{out}_\downarrow}\ agent.\overline{\mathsf{out}_\downarrow}\ g.Y
\end{array}
$$

Any fully formal typing of this system would take pages. For the sake of simplicity, let us make the following assumptions:

$$
\left\{ \begin{array}{lcl} \Gamma(agent) & = & \mathrm{CAA}\textsc{mb}(a_P + a_Q, 1)[T] \\ \Gamma(entered) & = & \mathrm{CAA}\textsc{mb}(0,0)[T] \end{array} \right.
\quad
\left\{ \begin{array}{lcl} \Gamma(f) & = & \mathrm{CAA}\textsc{mb}(\infty,0)[T] \\ \Gamma(g) & = & \mathrm{CAA}\textsc{mb}(1,0)[T] \end{array} \right.
$$

Figure 8 presents a simplified typing derivation for part of the system. In this figure, we write Δ for $\Gamma, X : \mathrm{CAP}\textsc{roc}(t)[T]$.

This typing requires $\mathrm{CAP}\textsc{roc}(t)[T] = \mathrm{CAP}\textsc{roc}(t+1)[T]$. This is possible if and only if $t = \infty$, and as a consequence the capacity of f should also be ∞. In other words, 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 and resource control.

Cab. Let us consider an environment Γ such that

$$\begin{cases} \Gamma(\mathit{client}) &= \textsc{CAAmb}(0,1)[T] \\ \Gamma(\mathit{call}) &= \textsc{CAAmb}(1,0)[T] \\ \Gamma(\mathit{trip}) &= \textsc{CAAmb}(0,0)[T] \\ \Gamma(\mathit{arrived}) &= \textsc{CAAmb}(0,0)[T] \end{cases} \qquad \begin{cases} \Gamma(\mathit{end}) &= \textsc{CAAmb}(0,0)[T] \\ \Gamma(\mathit{cab}) &= \textsc{CAAmb}(1,0)[T] \\ \Gamma(\mathit{site}_i) &= \textsc{CAAmb}(\infty,0)[T] \\ \Gamma(\mathit{city}) &= \textsc{CAAmb}(0,0)[T] \end{cases}$$

Note in particular that this resource policy specifies that among the ambients that may enter the cab, only those named *client* are taken into account for resource control: this corresponds to the property we focus on when analyzing the cab. With these assumptions, the complete cab system is typeable. From the derivation, we prove that resources are statically controlled in cabs: *at any step of its execution, the cab may contain at most one client.* Moreover, by changing our resource policy in such a way that ambients *call*, *trip*, *arrived* and *end* have weight 1 while *client* has weight 0, we can type the cab as having size 1. This second typing lets us control the number of "auxiliary" ambients: at any time, at most one of those may be present in the *cab*.

4 Other Systems

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

$$T_1 \triangleq a[\overline{\mathsf{open}}\ \{a,b\}.\mathsf{rec}\ X.(X \mid b[\mathbf{0}])] \mid \mathsf{open}\ a\,,$$

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

$$\begin{aligned} T_2 \triangleq\ & h[\mathsf{rec}\ X.(m[\overline{\mathsf{in}_\downarrow}\ n.\overline{\mathsf{out}_\uparrow}\ n.\overline{\mathsf{open}}\ \{m,h\}] \mid \overline{\mathsf{out}_\downarrow}\ n.\overline{\mathsf{in}_\uparrow}\ n.\mathsf{open}\ m.X) \\ & \qquad\qquad\qquad\qquad \mid n[\mathsf{rec}\ Y.\mathsf{in}\ m.\mathsf{out}\ m.Y]]\,, \end{aligned}$$

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, trying to apply our system to term T_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 controls resources too strictly. During the discussion, let us use the following notations for the rule $R-\mathit{open}$:

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

In order to try and refine the typing of 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, named r. In $\textsc{CAAmb}(s,e,r)[T]$, $r \in \overline{\mathbb{N}}$ 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:

$$\frac{\Gamma \vdash m : \mathrm{CAA{\scriptstyle MB}}(s,e,r)[T] \quad \Gamma \vdash P : \mathrm{CAP{\scriptstyle ROC}}(t)[T]}{\Gamma \vdash \mathsf{open}\ m.P : \mathrm{CAP{\scriptstyle ROC}}(t-e+s+r)[T]} \quad \begin{array}{c} CA-open \\ t-e+s+r \geq 0 \end{array}$$

$$\frac{\Gamma \vdash m : \mathrm{CAA{\scriptstyle MB}}(s,e,r)[T] \quad \Gamma \vdash R : \mathrm{CAP{\scriptstyle ROC}}(t)[T]}{\Gamma \vdash \overline{\mathsf{open}}\ \{m,h\}.R : \mathrm{CAP{\scriptstyle ROC}}(t')[T]} \quad \begin{array}{c} CA-coopen \\ t \leq r \end{array}$$

Using these alternative rules, term T_1 may be satisfactorily typed (i.e. with a finite capacity for a), taking $r = \infty$. Additionally, we checked that all results of Section 3.2 still remain valid. However, System R does not help with term T_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 $z \leq s$, representing this better approximation. More precisely, in System Z, ambient types become $\mathrm{CAA{\scriptstyle MB}}(s,e,z)[T]$ with $z \in \mathbb{N}$ and $z \leq s$, and the typing rules are:

$$\frac{\Gamma \vdash m : \mathrm{CAA{\scriptstyle MB}}(s,e,z)[T] \quad \Gamma \vdash P : \mathrm{CAP{\scriptstyle ROC}}(t)[T]}{\Gamma \vdash \mathsf{open}\ m.P : \mathrm{CAP{\scriptstyle ROC}}(t-e+z)[T]} \quad \begin{array}{c} CA-open \\ t-e+z \geq 0 \end{array}$$

$$\frac{\Gamma \vdash m : \mathrm{CAA{\scriptstyle MB}}(s,e,z)[T] \quad \Gamma \vdash R : \mathrm{CAP{\scriptstyle ROC}}(t)[T]}{\Gamma \vdash \overline{\mathsf{open}}\ \{m,h\}.R : \mathrm{CAP{\scriptstyle ROC}}(t+s-z)[T]} \quad CA-coopen$$

In turn, System Z permits a good analysis of term T_2, but cannot handle term T_1 any better than the basic system. Additionally, results from Section 3.2 also remain valid on System Z.

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

$$\frac{\Gamma \vdash m : \mathrm{CAA{\scriptstyle MB}}(s,e,r,z)[T] \quad \Gamma \vdash P : \mathrm{CAP{\scriptstyle ROC}}(t)[T]}{\Gamma \vdash \mathsf{open}\ m.P : \mathrm{CAP{\scriptstyle ROC}}(t-e+z+r)[T]} \quad \begin{array}{c} CA-open \\ t-e+z+r \geq 0 \end{array}$$

$$\frac{\Gamma \vdash m : \mathrm{CAA{\scriptstyle MB}}(s,e,r,z)[T] \quad \Gamma \vdash R : \mathrm{CAP{\scriptstyle ROC}}(t)[T]}{\Gamma \vdash \overline{\mathsf{open}}\ \{m,h\}.R : \mathrm{CAP{\scriptstyle ROC}}(t')[T]} \quad \begin{array}{c} CA-coopen \\ t \leq r, t' \geq s-z \end{array}$$

As expected, System RZ correctly handles both terms T_1 and T_2, and results from Section 3.2 also remain valid. Hence, System RZ is a more refined although more complicated type system.

5 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 synchronisations. 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. Further, examples show that indications on the maximal amount of resources needed by a process match rather closely the actual amount of resources which 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.

We are indeed working on applying this formalism (or an extension of it) to analyze some examples of concurrent computation involving a form of mobility.

Among extensions of the present work, we are currently working on type inference for our system. It seems that by requiring the recursion variables to be explicitely 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 (resembling those given for the example of renaming in Section 3), but solving it in the general case would require more work.

We are also working on an extension of the language and type system, to include communication of capabilities, as in the original Mobile Ambients calculus [2]. This should make the translation of properties and encodings from Mobile Ambients more easy.

Meanwhile, we plan to study whether our approach can be adapted to other formalisms for mobile and distributed computation such as the distributed π-calculus [17] or the distributed join-calculus [7]. Along the same ideas, in some process calculi without any primitive notion of site, we could choose to regard name creation as a form of resource allocation, and find out whether (some of) our ideas can be adapted to this setting.

We could also consider combining our type system for resource control with other typing disciplines, adapted from the Single Threadness types of [14], or the Mandatory Access Control of [1]. It seems that Controlled Ambients could also be used to approximate some of the analyses done in [6, 11], where, in a context where *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. In the simple case where we have two security levels, we could attach weight 0 to agents of high level, and 1 to low-level agents, and store high-level information in ambients of size 0: in such a framework, our type system can guarantee that only high-level processes can enter high-level data. Of course this is a very rough approximation, and a more refined account of access control in Controlled Ambients needs further investigation.

We have not addressed the issue of behavioural equivalences for Controlled Ambients. 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 [9]: 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. Other projects aim at controlling resources in possibly mobile systems without resorting to mobile process algebras. [13] presents a modified ML language with sized types in which bounds may be given to stack consumption. Like in our framework, resources are releasable entities; however, this approach seems more specialized than ours, and moreover concentrates on a sequential model. Similarly, [5] introduces a variant of the Typed Assembly Language *"augmenting TAL's very low-level safety certification with running-time guarantees"*, while Quantum, [16] may be used to describe distributed systems from the point of view of their resource consumption. In contrast to our approach, both systems consider non-releasable resources. Another programming language, PLAN [12], has been designed specifically for active networks, and also handles some form of resource bounds. Although PLAN accounts for both releasable (space, bandwidth) and non-releasable (time) resources, it handles neither recursion nor concurrency on one node.

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 approach can be integrated to these works, in order to combine several forms of resource control.

Another form of accounting on mobile ambients is introduced in [4]: Finite-Control Mobile Ambients. 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 resources: it tries and isolate a finite-control fragment of mobile ambients on which model checking w.r.t. the Ambient Logic is decidable through state-space exploration.

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

References

[1] M. Bugliesi, G. Castagna, and S. Crafa. Boxed ambients. In *Proc. TACS 2001*, LNCS 2215, pages 38–63. Springer Verlag, 2001.

[2] L. Cardelli and A. D. Gordon. Mobile ambients. In M. Nivat, editor, *Proceedings of Foundations of Software Science and Computation Structures (FoSSaCS'98)*, volume 1378, pages 140–155. Springer Verlag, 1998.

[3] L. Cardelli and A. D. Gordon. Types for mobile ambients. In *Symposium on Principles of Programming Languages (POPL'99)*, pages 79–92. ACM Press, 1999.

[4] W. Charatonik, A. D. Gordon, and J.-M. Talbot. Finite-control mobile ambients. In *European Symposium on Programming 2002 - Lecture Notes on Computer Science*. to appear.

[5] K. Crary and S. Weirich. Resource bound certification. In *Symposium on Principles of Programming Languages (POPL'00)*, pages 184–198. ACM Press, 2000.

[6] M. Dezani-Ciancaglini and I. Salvo. Security types for mobile safe ambients. In *Asian Computing Science Conference (ASIAN'00)*, LNCS 1961, pages 215–236. Springer Verlag, 2000.

[7] C. Fournet, G. Gonthier, J.-J. Lévy, L. Maranget, and D. Rémy. A calculus of mobile agents. In *Proceedings of the 7th International Conference on Concurrency Theory (CONCUR'96)*, pages 406–421. Springer Verlag, 1996.

[8] C. Fournet, J.-J. Lévy, and A. Schmitt. A distributed implementation of mobile ambients. In *Proceedings of IFIP International Conference on Theoretical Computer Science (IFIP TCS 2000)*, pages 348–364. Springer Verlag, 1872.

[9] A. D. Gordon and L. Cardelli. Equational properties of mobile ambients. In W. Thomas, editor, *Proceedings of the Second International Conference on Foundations of Software Science and Computation Structures (FoSSaCS '99)*, volume 1578 of *LNCS*, pages 212–226. Springer Verlag, 1999.

[10] X. Guan, Y. Yang, and J. You. Making ambients more robust. In *Proceedings of the International Conference on Software: Theory and Practice*, pages 377–384, Aug 2000.

[11] M. Hennessy and J. Riely. Resource access control in systems of mobile agents. In *Proceedings of HLCL '98: High-Level Concurrent Languages*, number 16.3 in Electronic Notes in Theoretical Computer Science, pages 3–17. Elsevier, 1998.

[12] M. Hicks, P. Kakkar, J. T. Moore, C. A. Gunter, and S. Nettles. PLAN: A Packet Language for Active Networks. In *Proceedings of the Third ACM SIGPLAN International Conference on Functional Programming*, pages 86–93. ACM Press, 1999.

[13] J. Hughes and L. Pareto. Recursion and dynamic data-structures in bounded space: Towards embedded ML programming. In *International Conference on Functional Programming*, pages 70–81. ACM Press, 1999.

[14] F. Levi and D. Sangiorgi. Controlling interference in ambients. In *Symposium on Principles of Programming Languages*, pages 352–364. ACM Press, 2000.

[15] M. Merro and M. Hennessy. Bisimulation congruences in safe ambients, 2002. to appear in Proc. of POPL'02.

[16] L. Moreau. A distributed garbage collector with diffusion tree reorganisation and mobile objects. In *International Conference on Functional Programming*, pages 204–215. ACM Press, 1998.

[17] J. Riely and M. Hennessy. A typed language for distributed mobile processes. In *Conference Record of POPL 98: The 25TH ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, San Diego, California*, pages 378–390. ACM Press, 1998.

[18] D. Sangiorgi and A. Valente. A distributed abstract machine for Safe Ambients. In *Proc. of ICALP'01*, 2001.

A Proofs

A.1 Subtyping

Lemma 1 (Subtyping) Let P be a process and Γ an environment. Then

$$\text{if} \quad \Gamma \vdash P : \mathrm{CAP}\textsc{roc}(t)[T] \quad \text{then} \quad \forall t' \geq t, \Gamma \vdash P : \mathrm{CAP}\textsc{roc}(t')[T]$$

Proof: By induction on the derivation of $\Gamma \vdash P : \mathrm{CAP}\textsc{roc}(t)[T]$.

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 upgrade 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 require only a simple induction step.

□

A.2 Strengthening and Weakening

Lemma 8 *If* $\Gamma, n : A \vdash P : U$ *and* $n \notin fn(P)$, *then* $\Gamma \vdash P : U$.

Lemma 9 *If* $\Gamma, X : U' \vdash P : U$ *and* $X \notin fv(P)$, *then* $\Gamma \vdash P : U$.

Lemma 10 *If* $\Gamma \vdash P : U$ *and* $n \notin fn(P)$, *then* $\Gamma, n : A \vdash P : U$.

A.3 Resource Control

Lemma 11 *If* $\Gamma \vdash P : \mathrm{CAP}\textsc{roc}(t)[T]$, *then* $Res_\Gamma(P) \leq t$.

Proof: By induction on the derivation of $\Gamma \vdash P : \mathrm{CAP}\textsc{roc}(t)[T]$.

Case CA-amb We have $\Gamma \vdash n[P] : \mathrm{CAP}\textsc{roc}(t)[T]$ with $e \leq t$, where e is the weight of n in Γ. Then, $Res_\Gamma(n[P]) = e \leq t$.

Case CA-par $\Gamma \vdash P|Q : \mathrm{CAP}\textsc{roc}(t_P + t_Q)[T]$ must have been derived from $\Gamma \vdash P : \mathrm{CAP}\textsc{roc}(t_P)[T]$ and $\Gamma \vdash Q : \mathrm{CAP}\textsc{roc}(t_Q)[T]$. By 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 $\Gamma \vdash (\nu n : A)P : \mathrm{CAP}\textsc{roc}(t)[T]$ must have been derived from $\Gamma, n : A \vdash P : \mathrm{CAP}\textsc{roc}(t)[T]$. Then, by induction hypothesis, we have: $Res_\Gamma((\nu n : A)P) = Res_{\Gamma,n:A}(P) \leq t$.

Other cases For all other cases, $Res_\Gamma(P) = 0 \leq t$.

□

Lemma 5 (Typeable terms comply with resource policies) For any process P, resource policy Γ and process type U, if $\Gamma \vdash P : U$, then $\Gamma \models P$.

Proof: By induction on the structure of P.

Case $P|Q$ Since $P|Q$ is typeable in Γ, so are P and Q. By 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 induction hypothesis, $\Gamma, n : A \models P$. Then, $\Gamma \models (\nu n : A)P$.

Case $n[P]$ This is the main case; we have to 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 induction hypothesis.
- Secondly, 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 $CA - amb$: $\Gamma(n) = \text{CA}\textsc{Amb}(s, e)[T]$ and $\Gamma \vdash P : \text{CAP}\textsc{roc}(t)[T]$, with the condition $t \leq s$. By Lemma 11, we can conclude: $Res_\Gamma(P) \leq t \leq s$.

Other cases In all other cases, we have nothing to check.

□

A.4 Equivalence Lemma

Lemma 12 *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 $\Gamma \vdash P : \text{CAP}\textsc{roc}(t)[T]$, we can type $\Gamma \vdash \mathbf{0} : \text{CAP}\textsc{roc}(0)[T]$ and then $\Gamma \vdash P \mid \mathbf{0} : \text{CAP}\textsc{roc}(t)[T]$ by $CA - par$.
- If $\Gamma \vdash P \mid \mathbf{0} : \text{CAP}\textsc{roc}(t)[T]$, this must have been derived by $CA - par$ from $\Gamma \vdash P : \text{CAP}\textsc{roc}(t_1)[T]$ and $\Gamma \vdash \mathbf{0} : \text{CAP}\textsc{roc}(t_2)[T]$ with $t_1 + t_2 = t$. Since $t_1 \leq t$, we have $\Gamma \vdash P : \text{CAP}\textsc{roc}(t)[T]$ using Lemma 1.

Case S-respar

- Suppose that $\Gamma \vdash (\nu n : A)(P|Q) : \text{CAP}\textsc{roc}(t)[T]$. This must have been derived by $CA - res$ from $\Gamma, n : A \vdash P|Q : \text{CAP}\textsc{roc}(t)[T]$, which in turn must have been derived by $CA - par$ from $\Gamma, n : A \vdash P : \text{CAP}\textsc{roc}(t_1)[T]$ and $\Gamma, n : A \vdash Q : \text{CAP}\textsc{roc}(t_2)[T]$ with $t_1 + t_2 = t$. From the first affirmation, we get $\Gamma \vdash (\nu n : A)P : \text{CAP}\textsc{roc}(t_1)[T]$ by $CA - res$. From the second one, and since $n \notin fn(Q)$, we can use Lemma 8 and obtain $\Gamma \vdash Q : \text{CAP}\textsc{roc}(t_2)[T]$. Finally, using $CA - par$, we get $\Gamma \vdash (\nu : A)P \mid Q : \text{CAP}\textsc{roc}(t)[T]$.
- Starting from $\Gamma \vdash (\nu : A)P \mid Q : \text{CAP}\textsc{roc}(t)[T]$, the reasoning is similar, except that we use Lemma 10 instead of Lemma 8.

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

The other cases are similar or trivial. □

A.5 Process Substitution Lemma

Lemma 13 *If* $\Gamma, X : U \vdash P : U'$ *and* $\Gamma \vdash Q : U$, *then* $\Gamma \vdash P\{X \leftarrow Q\} : U'$.

Proof: Let $U = \mathrm{CAP{\small ROC}}(t)[T]$. 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 $\Gamma, X : U \vdash X : \mathrm{CAP{\small ROC}}(t')[T]$ with $t' \geq t$. By Lemma 1, we also have $\Gamma \vdash Q : \mathrm{CAP{\small ROC}}(t')[T]$. 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 9, we can remove X from the environment since it is no more used, and obtain: $\Gamma \vdash P\{X \leftarrow Q\} : U'$. □

A.6 Name Substitution Lemma

Lemma 14 *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 8, we can remove y from the environment since it is no more used in $P\{y \leftarrow x\}$. □

A.7 Subject Reduction

Theorem 6 (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

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

Let Γ be an environment such that

$$\left\{ \begin{array}{lcl} \Gamma(m) & = & \mathrm{CAA{\small MB}}(s_m, e_m)[T_m] \\ \Gamma(n) & = & \mathrm{CAA{\small MB}}(s_n, e_n)[T_n] \\ \Gamma & \vdash & P : \mathrm{CAP{\small ROC}}(t_P)[T_P] \\ \Gamma & \vdash & Q : \mathrm{CAP{\small ROC}}(t_Q)[T_Q] \\ \Gamma & \vdash & R : \mathrm{CAP{\small ROC}}(t_R)[T_R] \\ \Gamma & \vdash & S : \mathrm{CAP{\small ROC}}(t_S)[T_S] \\ \Gamma & \vdash & T : \mathrm{CAP{\small ROC}}(t_T)[T_T] \end{array} \right.$$

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

$$\text{Typing } m[\mathsf{in}\ n.P \mid Q]$$

$$\begin{array}{llll}
\Gamma \vdash P : & \text{CAP}\textsc{roc}(t_P)[T_P] & & \text{by hypothesis} \\
\Gamma \vdash \mathsf{in}\ n.P : & \text{CAP}\textsc{roc}(t_P)[T_P] & & CA - in \\
\Gamma \vdash Q : & \text{CAP}\textsc{roc}(t_Q)[T_Q] & & \text{by hypothesis} \\
\Gamma \vdash \mathsf{in}\ m.P|Q : & \text{CAP}\textsc{roc}(t_P + t_Q)[T_P] & T_P = T_Q & CA - par \\
\Gamma \vdash m[\cdots] : & \text{CAP}\textsc{roc}(t_1)[T_1] & t_P + t_Q \leq s_m, & CA - amb \\
 & & e_m \leq t_1, T_P = T_m &
\end{array}$$

$$\text{Typing } n[\overline{\mathsf{in}_\downarrow}\ m.R \mid S]$$

$$\begin{array}{llll}
\Gamma \vdash R : & \text{CAP}\textsc{roc}(t_R)[T_R] & & \text{by hypothesis} \\
\Gamma \vdash \overline{\mathsf{in}_\downarrow}\ m.R : & \text{CAP}\textsc{roc}(t_R + e_m)[T_R] & & CA - coin \\
\Gamma \vdash S : & \text{CAP}\textsc{roc}(t_S)[T_S] & & \text{by hypothesis} \\
\Gamma \vdash \overline{\mathsf{in}_\downarrow}\ m.R|S : & \text{CAP}\textsc{roc}(t_S + t_R + e_m)[T_R] & T_S = T_R & CA - par \\
\Gamma \vdash n[\cdots] : & \text{CAP}\textsc{roc}(t_2)[T_2] & t_S + t_R + e_m \leq s_n, & CA - amb \\
 & & e_n \leq t_2, T_R = T_n &
\end{array}$$

$$\text{Typing toplevel}$$

$$\begin{array}{llll}
\Gamma \vdash T : & \text{CAP}\textsc{roc}(t_T)[T_T] & & \text{by hypothesis} \\
\Gamma \vdash \overline{\mathsf{out}_\downarrow}\ m.T : & \text{CAP}\textsc{roc}(t_T - e_m)[T_T] & t_T \geq e_m & CA - coout \\
\Gamma \vdash n[\ldots]|\overline{\mathsf{out}_\downarrow}\ m.T : & \text{CAP}\textsc{roc}(t_T + t_2 - e_m)[T_T] & T_T = T_2 & CA - par \\
\Gamma \vdash A : & \text{CAP}\textsc{roc}(t_T + t_1 + t_2 - e_m)[T_T] & T_T = T_1 & CA - par
\end{array}$$

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

$$\begin{array}{llll}
\Gamma \vdash P : & \text{CAP}\textsc{roc}(t_P)[T_P] & & \text{by hypothesis} \\
\Gamma \vdash Q : & \text{CAP}\textsc{roc}(t_Q)[T_Q] & & \text{by hypothesis} \\
\Gamma \vdash P|Q : & \text{CAP}\textsc{roc}(t_Q + t_Q)[T_P] & T_P = T_Q\surd & CA - par \\
\Gamma \vdash m[P|Q] & \text{CAP}\textsc{roc}(e_m)[T_R] & T_P = T_m, t_P + t_Q \leq s_m\surd & CA - amb \\
\Gamma \vdash R : & \text{CAP}\textsc{roc}(t_R)[T_R] & & \text{by hypothesis} \\
\Gamma \vdash S : & \text{CAP}\textsc{roc}(t_S)[T_S] & & \text{by hypothesis} \\
\Gamma \vdash R|S : & \text{CAP}\textsc{roc}(t_R + t_S)[T_R] & T_R = T_S\surd & CA - par \\
\Gamma \vdash R \mid S \mid m[\cdots] : & \text{CAP}\textsc{roc}(t_R + t_S + e_m)[T_R] & & CA - par \\
\Gamma \vdash n[\cdots] : & \text{CAP}\textsc{roc}(t_1 + t_2 - e_m)[T_T] & e_n \leq t_2 \leq t_1 + t_2 - e_m\surd & CA - amb \\
 & & T_R = T_n, t_R + t_S + e_m \leq s_n\surd & \\
\Gamma \vdash T : & \text{CAP}\textsc{roc}(t_T)[T_T] & & \text{by hypothesis} \\
\Gamma \vdash B : & \text{CAP}\textsc{roc}(t_T + t_1 + t_2 - e_m)[T_T] & & CA - par
\end{array}$$

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

$$\begin{cases} \Gamma(m) & = & \mathrm{CAA{\small MB}}(s_m, e_m)[T_m] \\ \Gamma(n) & = & \mathrm{CAA{\small MB}}(s_n, e_n)[T_n] \\ \Gamma & \vdash & P : \mathrm{CAP{\small ROC}}(t_P)[T_P] \\ \Gamma & \vdash & Q : \mathrm{CAP{\small ROC}}(t_Q)[T_Q] \\ \Gamma & \vdash & R : \mathrm{CAP{\small ROC}}(t_R)[T_R] \\ \Gamma & \vdash & S : \mathrm{CAP{\small ROC}}(t_S)[T_S] \\ \Gamma & \vdash & T : \mathrm{CAP{\small ROC}}(t_T)[T_T] \end{cases}$$

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

$$\begin{array}{llll}
\multicolumn{4}{c}{\text{Typing } m[\mathsf{out}\ n.P \mid Q]} \\
\Gamma \vdash P : & \mathrm{CAP{\small ROC}}(t_P)[T_P] & & \text{by hypothesis} \\
\Gamma \vdash \mathsf{out}\ n.P : & \mathrm{CAP{\small ROC}}(t_P)[T_P] & & CA-out \\
\Gamma \vdash Q : & \mathrm{CAP{\small ROC}}(t_Q)[T_Q] & & \text{by hypothesis} \\
\Gamma \vdash \mathsf{out}\ m.P|Q : & \mathrm{CAP{\small ROC}}(t_P + t_Q)[T_P] & T_P = T_Q & CA-par \\
\Gamma \vdash m[\cdots] : & \mathrm{CAP{\small ROC}}(t_1)[T_1] & t_P + t_Q \leq s_m, & CA-amb \\
 & & e_m \leq t_1, T_P = T_m & \\
\multicolumn{4}{c}{\text{Typing } n[m[\cdots] \mid \overline{\mathsf{out}_\uparrow}\ m.R \mid S]} \\
\Gamma \vdash R : & \mathrm{CAP{\small ROC}}(t_R)[T_R] & & \text{by hypothesis} \\
\Gamma \vdash \overline{\mathsf{out}_\uparrow}\ m.R : & \mathrm{CAP{\small ROC}}(t_R - e_m)[T_R] & e_m \leq t_R & CA-coout \\
\Gamma \vdash S : & \mathrm{CAP{\small ROC}}(t_S)[T_S] & & \text{by hypothesis} \\
\Gamma \vdash \overline{\mathsf{out}_\uparrow}\ m.R|S : & \mathrm{CAP{\small ROC}}(t_S + t_R - e_m)[T_R] & T_S = T_R & CA-par \\
\Gamma \vdash m[\cdots]|\overline{\mathsf{out}_\uparrow}\ m.R|S : & \mathrm{CAP{\small ROC}}(t_1 + t_S + t_R - e_m)[T_R] & T_1 = T_R & CA-par \\
\Gamma \vdash n[\cdots] : & \mathrm{CAP{\small ROC}}(t_2)[T_2] & e_n \leq t_2, T_R = T_n & CA-amb \\
 & & t_1 + t_S + t_R - e_m \leq s_n, & \\
\multicolumn{4}{c}{\text{Typing toplevel}} \\
\Gamma \vdash T : & \mathrm{CAP{\small ROC}}(t_T)[T_T] & & \text{by hypothesis} \\
\Gamma \vdash \overline{\mathsf{out}_\downarrow}\ m.T : & \mathrm{CAP{\small ROC}}(t_T + e_m)[T_T] & & CA-coin \\
\Gamma \vdash A : & \mathrm{CAP{\small ROC}}(t_2 + t_T + e_m)[T_T] & T_2 = T_T & CA-par
\end{array}$$

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

$$\begin{array}{llll}
\Gamma \vdash P : & \mathrm{CAP{\small ROC}}(t_P)[T_P] & & \text{by hypothesis} \\
\Gamma \vdash Q : & \mathrm{CAP{\small ROC}}(t_Q)[T_Q] & & \text{by hypothesis} \\
\Gamma \vdash P|Q : & \mathrm{CAP{\small ROC}}(t_Q + t_Q)[T_P] & T_P = T_Q \surd & CA-par \\
\Gamma \vdash m[P|Q] & \mathrm{CAP{\small ROC}}(e_m)[T_T] & T_P = T_m, t_P + t_Q \leq s_m \surd & CA-amb \\
\Gamma \vdash R : & \mathrm{CAP{\small ROC}}(t_R)[T_R] & & \text{by hypothesis} \\
\Gamma \vdash S : & \mathrm{CAP{\small ROC}}(t_S)[T_S] & & \text{by hypothesis} \\
\Gamma \vdash R|S : & \mathrm{CAP{\small ROC}}(t_R + t_S)[T_R] & T_R = T_S \surd & CA-par \\
\Gamma \vdash n[R|S] & \mathrm{CAP{\small ROC}}(t_2)[T_T] & e_n \leq t_2, T_R = T_n \surd & CA-amb \\
 & & t_R + t_S \leq t_1 - e_m + t_R + t_S \leq s_n \surd & \\
\Gamma \vdash T : & \mathrm{CAP{\small ROC}}(t_T)[T_T] & & \text{by hypothesis} \\
\Gamma \vdash B : & \mathrm{CAP{\small ROC}}(t_2 + t_T + e_m)[T_T] & & CA-par(twice)
\end{array}$$

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

$$\begin{cases} \Gamma(m) &=& \text{CAA}\textsc{mb}(s_m, e_m)[T_m] \\ \Gamma(h) &=& \text{CAA}\textsc{mb}(s_h, e_h)[T_h] \\ \Gamma &\vdash& P : \text{CAP}\textsc{roc}(t_P)[T_P] \\ \Gamma &\vdash& Q : \text{CAP}\textsc{roc}(t_Q)[T_Q] \\ \Gamma &\vdash& R : \text{CAP}\textsc{roc}(t_R)[T_R] \\ \Gamma &\vdash& S : \text{CAP}\textsc{roc}(t_S)[T_S] \end{cases}$$

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

$$\begin{array}{llll}
 & \text{Typing } m[\cdots] & & \\
\Gamma \vdash S : & \text{CAP}\textsc{roc}(t_S)[T_S] & & \text{by hypothesis} \\
\Gamma \vdash R : & \text{CAP}\textsc{roc}(t_R)[T_R] & & \text{by hypothesis} \\
\Gamma \vdash \overline{\mathsf{open}}\ \{m,h\}.R : & \text{CAP}\textsc{roc}(t_R)[T_R] & T_m = T_R & CA - coopen \\
\Gamma \vdash \overline{\mathsf{open}}\ \{m,h\}.R \mid S : & \text{CAP}\textsc{roc}(t_R + t_S)[T_R] & T_R = T_S & CA - par \\
\Gamma \vdash m[\cdots] : & \text{CAP}\textsc{roc}(t_1)[T_1] & t_R + t_S \leq s_m, & CA - amb \\
 & & e_m \leq t_1 & \\
 & \text{Typing } h[\mathsf{open}\ m.P \mid Q \mid m[\cdots]] & & \\
\Gamma \vdash Q : & \text{CAP}\textsc{roc}(t_Q)[T_Q] & & \text{by hypothesis} \\
\Gamma \vdash P : & \text{CAP}\textsc{roc}(t_P)[T_P] & & \text{by hypothesis} \\
\Gamma \vdash \mathsf{open}\ m.P : & \text{CAP}\textsc{roc}(t_P - e_m + s_m)[T_P] & T_m = T_P & CA - open \\
 & & t_P - e_m + s_m \geq 0 & \\
\Gamma \vdash \mathsf{open}\ m.P \mid Q : & \text{CAP}\textsc{roc}(t_P + t_Q - e_m + s_m)[T_P] & T_P = T_Q & CA - par \\
\Gamma \vdash \mathsf{open}\ m.P|Q|m[\cdots] : & \text{CAP}\textsc{roc}(t_P + t_Q - e_m + s_m + t_1)[T_P] & T_P = T_1 & CA - par \\
\Gamma \vdash A : & \text{CAP}\textsc{roc}(t_2)[T_2] & T_P = T_h, & CA - amb \\
 & & e_h \leq t_2, & \\
 & & t_P + t_Q - e_m + s_m + t_1 \leq s_h &
\end{array}$$

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 $CA - par$:

$$\Gamma \vdash P \mid Q \mid R \mid S : \text{CAP}\textsc{roc}(t_P + t_Q + t_R + t_S)[T_h]$$

. Using the previous conditions, we find:

$$t_P + t_Q + t_R + t_S \leq t_P + t_Q + s_m \leq t_P + t_Q + s_m + t_1 - e_m \leq s_h$$

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

$$\Gamma \vdash B : \text{CAP}\textsc{roc}(t_2)[T_2]$$

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

$$\begin{cases} \Gamma(n) & = & A_n \\ \Gamma, x : N & \vdash & P : \text{CAP}\textsc{roc}(t_P)[T_P] \end{cases}$$

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

$$\begin{array}{llll}
\Gamma \vdash \langle n \rangle : & \text{CAP}\textsc{roc}(t_1)[t, A_n] & t_1 \geq t & CA - send \\
\Gamma, x : N \vdash P : & \text{CAP}\textsc{roc}(t_P)[T_P] & & \text{by hypothesis} \\
\Gamma \vdash (x : N)P : & \text{CAP}\textsc{roc}(t_2)[t_P, N] & T_P = t_P & CA - receive \\
\Gamma \vdash A : & \text{CAP}\textsc{roc}(t_1 + t_2)[t_P, A_n] & N = A_n, t = t_P & CA - par
\end{array}$$

With the above conditions and by hypothesis, we have:

$$\Gamma, x : A_n \vdash P : \text{CAP}\textsc{roc}(t_P)[t_P, A_n]$$

and $\Gamma \vdash n : A_n$. Using Lemma 14, we get:

$$\Gamma \vdash P\{x \leftarrow n\} : \text{CAP}\textsc{roc}(t_P)[t_P, A_n]$$

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

$$\Gamma \vdash B : \text{CAP}\textsc{roc}(t_1 + t_2)[t_P, A_n]$$

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 $\Gamma \vdash A : \text{CAP}\textsc{roc}(t')[T]$ must have been derived from

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

with $t' \geq t$. Using the same rule, we can also conclude that $\Gamma \vdash \mathsf{rec}\ X.P$: $\text{CAP}\textsc{roc}(t)[T]$. By Lemma 13, we have:

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

And finally, we get $\Gamma \vdash B : \text{CAP}\textsc{roc}(t')[T]$ 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 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. Necessiraly, we have the following typings:

$$\begin{array}{l}
\Gamma \vdash A : \text{CAP}\textsc{roc}(t_P + t_Q)[T] \\
\Gamma \vdash P : \text{CAP}\textsc{roc}(t_P)[T] \\
\Gamma \vdash Q : \text{CAP}\textsc{roc}(t_Q)[T]
\end{array}$$

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

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 $\text{CAP}\textsc{roc}(t)[T]$, so does P with type $\text{CAP}\textsc{roc}(t_P)[T_m]$, and m with type $\text{CAA}\textsc{mb}(s_m, e_m)[T_m]$. Additionnally, we have $t_P \leq s_m$ and $e_m \leq t$.

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

Case R-$\equiv$ This case is trivial, by induction hypothesis and using Lemma 12 twice.

□