In TACS 2001, Lecture Notes in Computer Science,© Springer, 2001. To appear.

Boxed Ambients ⋆

Michele Bugliesi[1], Giuseppe Castagna[2], and Silvia Crafa[1,2]

[1] Dipartimento di Informatica
Univ. "Ca' Foscari", Venezia, Italy

[2] Département d'Informatique
École Normale Supérieure, Paris, France

Abstract. *Boxed Ambients* are a variant of Mobile Ambients, that result from (i) dropping the open capability and (ii) providing new primitives for ambient communication while retaining the constructs in and out for mobility. The new model of communication is faithful to the principles of distribution and location-awareness of Mobile Ambients, and complements the constructs for Mobile Ambient mobility with finer-grained, and more effective, mechanisms for ambient interaction.

1 Introduction

There is a general agreement that calculi and programming languages for wide-area computing and mobile-code environments should be designed according to appropriate principles, among which distribution and location awareness are the most fundamental.

Cardelli and Gordon's Mobile Ambients [CG98] are one of the first, and currently one of the most successful implementations of these principles into a formal calculus. Their design is centered around four basic notions: location, mobility, authorization to move (based on acquisition of names and capabilities), and communication by shared location.

Boxed Ambients[1] are a variant of Mobile Ambients, from which they inherit the primitives in and out for mobility, with the exact same semantics. Though, Boxed Ambients rely on a completely different model of communication, which results from dropping the open capability. The new communication primitives fit nicely the design principles of Mobile Ambients, and complement the existing constructs for ambient mobility with finer-grained, and more effective, mechanisms for ambient interaction. As a result Boxed Ambients retain the expressive power and the computational flavor of Ambient Calculus, as well as the elegance of its formal presentation. In addition, they enhance the flexibility of typed communications over Mobile Ambients, and provide new insight into the relationship between synchronous and asynchronous input-output.

1.1 Mobile Ambients

Ambients are named process of the form $a[\![\, P \,]\!]$ where a is a name and P a process. Processes can be composed in parallel, as in $P \mid Q$, exercise a capability, as in $M.P$, declare local names as in $(\boldsymbol{\nu} x)P$, or simply do nothing as in $\mathbf{0}$. Ambients may be nested to form a tree structure that can be dynamically reconfigured by exercising the capabilities in, out and open. In addition, ambients and processes may communicate. Communication is anonymous, and happens inside ambients. The configuration $(\boldsymbol{x})P \mid \langle M \rangle$ represents the parallel composition of two processes, the output process $\langle M \rangle$ "dropping" the message M, and the input process $(\boldsymbol{x})P$ reading the message M and continuing as $P\{\boldsymbol{x} := M\}$. The open capability has a fundamental interplay with communication: in fact, communication results from a combination of mobility and opening control. To exemplify, consider two ambients running in parallel as in the following configuration $a[\![\, (\boldsymbol{x})P \mid Q \,]\!] \mid b[\![\, \langle M \rangle \mid R \,]\!]$. The exchange of the value M from b to the process P enclosed in a

⋆ Work partially supported by MURST Project 9901403824_003, by CNRS Program *Telecommunications*: "Collaborative, distributed, and secure programming for Internet", and by Galileo Action n. 02841UD

[1] The name is inspired by Sewell and Vitek's *Boxed π-Calculus* [SV00].

happens as a result of, say, first b moving inside a, and then a opening b. Thus, if Q is the process $\mathsf{open}\, b$, and R is $\mathsf{in}\, a$, communication is the result of the following sequence of reductions:

$$
\begin{array}{rll}
a[\![\, (\boldsymbol{x})P \mid \mathsf{open}\, b \,]\!] \mid b[\![\, \langle M\rangle \mid \mathsf{in}\, a \,]\!] & \rightarrow a[\![\, (\boldsymbol{x})P \mid \mathsf{open}\, b \mid b[\![\, \langle M\rangle \,]\!] \,]\!] & \text{by exercising } \mathsf{in}\, a \\
& \rightarrow a[\![\, (\boldsymbol{x})P \mid \langle M\rangle \,]\!] & \text{by } \mathsf{open}\, b\text{, unleashing } M \\
& \rightarrow a[\![\, P\{\boldsymbol{x} := M\} \,]\!] & \text{by local communication}
\end{array}
$$

A case against ambient opening. While fundamental to the Ambient Calculus for the reasons just illustrated, an unrestricted use of the open capability appears to bring about serious security concerns in wide-area distributed applications.

Consider a scenario where a process P running on host h downloads an application program Q from some other host over the network. This situation can be modeled by the configuration $a[\![\, \mathsf{in}\, h.Q \,]\!] \mid h[\![\, P \,]\!]$, where Q is included in the transport ambient a which is routed to h in response to the download request from P. As a result of a exercising the capability $\mathsf{in}\, h$, the system evolves into the new configuration $h[\![\, a[\![\, Q \,]\!] \mid P \,]\!]$, where the download is completed. The application program Q may be running and computing within a, but as long as it is encapsulated into a, there is no way that P and Q can effectively interact. To allow for interactions, P will need to dissolve the transport ambient a. Now, dissolving a produces the new configuration $h[\![\, P \mid Q \,]\!]$ where now P and Q are granted free access to each other's resources: the problem, of course, is that there is no way to tell what Q may do with them. An alternative, but essentially equivalent, solution to the above scenario, is to treat a as a *sandbox* and take the Java approach to security: P clones itself and enters the sandbox to interact with Q. Again, however, the kind of interaction between P and Q is not fully satisfactory: either they interact freely within a, or are isolated from each other.

Static or dynamic analysis of incoming code are often advocated as solutions to the above problem: incoming code must be statically checked and certified prior to being granted access to resources and sensitive data. Various papers explored this possibility, proposing control-flow analyses [NNHJ99,NN00,DLB00] and type systems [CGG99,DCS00,BC01] for Mobile Ambients. The problem with these solutions is that they may not be always possible, or feasible, in practice. The source code of incoming software may be not available for analysis, or be otherwise inaccessible or too complex to ensure rigorous assessment of its behavior. This is not meant to dismiss the role of static analysis. To the contrary, it should be taken as a motivation to seek new design principles enabling a more effective use of static analysis. One such principle for Mobile Ambients, which we advocate and investigate in this paper, is that ambient interaction should be controlled by finer-grained policies to prevent from unrestricted resource access while still providing effective communication primitives.

1.2 Boxed Ambients: overview and main results

Boxed Ambients result from Cardelli and Gordon's Mobile Ambients essentially by dropping the open capability while retaining the in and out capabilities for mobility. Disallowing open represents a rather drastic departure from the Ambient Calculus, and requires new primitives for process communication.

As in the Ambient Calculus, processes in the new calculus communicate via anonymous channels, inside ambients. This is a formal parsimony that simplifies the definition of the new calculus while not involving any loss of expressive power: in fact, named communication channels can be coded in faithful ways using the existing primitives. In addition, to compensate for the absence of open, Boxed Ambients are equipped with primitives for communication across ambient boundaries, between parent and children. Syntactically, this is obtained by means of tags specifying the *location* where the communication has to take place: for instance, $(x)^n P$ indicates an input from child ambient n, while $\langle M\rangle^\uparrow$ is an output to the parent ambient.

The choice of these primitives, and the resulting model of communication is inspired to Castagna and Vitek's *Seal Calculus* [VC99], from which Boxed Ambients also inherit the two principles of *mediation* and *locality*. Mediation implies that remote communication, i.e. between sibling ambients, is not possible: it either requires mobility, or intervention by the ambients' parent. Locality means that communication resources are *local* to ambients, and message exchanges result from explicit read and write requests on those resources. To exemplify, consider the following nested configuration:

$$n[\![\, (x)^p P \mid p[\![\, \langle M \rangle \mid (x)Q \mid q[\![\, \langle N \rangle^{\uparrow} \,]\!] \,]\!] \,]\!]$$

Ambient n makes a downward request to read p's local value M, while ambient q makes an upward write request to communicate its value N to its parent. The downward input request $(x)^p P$ may only synchronize with the output $\langle M \rangle$ local to p. Instead, $(x)Q$ may non-deterministically synchronize with either output: of course, type safety requires that M and N be of the same type. Interestingly, however, exchanges of different types may take place within the same ambient without type confusion:

$$n[\![\, (x)^p P \mid (x)^q Q \mid p[\![\, \langle M \rangle \,]\!] \mid q[\![\, \langle N \rangle \,]\!] \,]\!]$$

The two values M and N are local to p and q, respectively, and may very well have different types: there is no risk of type confusion, as $(x)^p P$ requests a read from p, while $(x)^q Q$ requests a read from q.

This flexibility of communication results from the combination of two design choices: directed input/output operations, and resource locality. In fact, these choices have other interesting consequences.

- They provide the calculus with fine-grained primitives for ambient interaction, and with clear notions of resource ownership and access request. Based on that, Boxed Ambients enable a rather direct formalization of classical security policies for resource protection and access control: this is not easy (if at all possible) with Mobile Ambients (see [BCC01]).
- They ease the design of type systems providing precise accounts of ambient behavior. As we show in § 4, a rather simple structure of types suffices for that purpose. Ambient and process types are defined simply as two-place constructors describing the types of exchanges that may take place locally, and with the enclosing context. Interestingly, this simple type structure is all that is needed to give a full account of ambient interaction. This is a consequence of (i) there being no way for ambients to communicate directly across more than one boundary, and (ii) communication being the only means for ambient to interact. Based on that, the typing of Boxed Ambients provides for more flexibility of communication and mobility than existing type systems for Mobile Ambients (see § 5).
- Finally, resource locality and directed input/output provide new insight into the relation between the synchronous and asynchronous models of communication. Specifically, the classic π-calculus relations between synchronous and asynchronous output, as stated by Boudol in [Bou92], no longer hold as a result of combining remote communications, resource locality and mobility. More precisely asynchronous output may *no longer* be viewed as a special case of synchronous output with null continuation, neither can it be encoded by structural equivalence, by stipulating that $\langle M \rangle P \equiv \langle M \rangle \mid P$. As we show (see § 7) these two solutions, which are equivalent in the π-calculus, have rather different consequences in Boxed Ambients.

1.3 Plan of the paper

§ 2 introduces Boxed Ambients. We first define the calculus in terms of synchronous output, and the subsequent four sections discuss various aspects of this formulation. In § 3 we give the encoding of different forms of channeled communications, based either on π-calculus' channels or on the Seal Calculus' *located* channels. In § 4, we introduce a type system, and in § 5 we compare the expressive power of typed Boxed Ambients and Mobile Ambients. In § 6 we give a different and more refined type system that further enhances typed communication and mobility. In § 7 we study an asynchronous variant of the calculus, and discuss the relationship between the two forms of communication and their impact on mobility. We conclude in § 8, with final remarks and comparisons with related work.

2 Boxed Ambients

Syntax. The syntax of the untyped calculus is defined by the following productions:

Expressions

M ::=	$a, b, \ldots$	names
|	$x, y, \ldots$	variables
|	$\mathsf{in}\ M$	enter M
|	$\mathsf{out}\ M$	exit M
|	$(M_1, \ldots, M_k)$	tuple, $k \geqslant 0$
|	$M.M$	path

Locations

η ::=	M	names and variables
|	$\uparrow$	enclosing ambient
|	$\star$	local

Processes

P ::=	$\mathbf{0}$	stop
|	$M.P$	action
|	$(\boldsymbol{\nu} x)P$	restriction
|	$P \mid P$	composition
|	$M[\![P]\!]$	ambient
|	$!P$	replication
|	$(\boldsymbol{x})^{\eta}P$	patterned input
|	$\langle M \rangle^{\eta}P$	synchronous output

Patterns

$\boldsymbol{x}$ ::=	x	variable
|	$\boldsymbol{x}_1, \ldots, \boldsymbol{x}_k$	tuple, $k \geqslant 0$

Patterns and expressions are as in the polyadic Ambient Calculus, save that the capability open and the "empty path" ε are left out. As in the Ambient Calculus, the syntax allows the formation of meaningless process forms such as $\mathsf{inout}m$ or $(\mathsf{open}\ a)[\![P]\!]$: these terms may arise as a result of reduction, in the untyped calculus, but are ruled out by the type system. We use a number of notation conventions. We reserve $a, b, c, \ldots, n, m, p, q$ for ambient names, and x, y, z for variables. As usual we omit trailing dead processes, writing M for $M.\mathbf{0}$, and $\langle M \rangle$ for $\langle M \rangle\mathbf{0}$. The empty tuple plays the role of synchronization messages. Finally, the superscript $\star$ denoting local communication, is omitted.

The operational semantics is defined in terms of reduction and structural congruence.

Structural Congruence. As in the Ambient Calculus, structural congruence is defined as the least congruence relation that is a commutative monoid for $\mathbf{0}$ and $\mid$ and closed under the following rules (*fn*(P), the set of free names of P, has standard definition)

(Res Dead)	$(\boldsymbol{\nu} x)\mathbf{0} \equiv \mathbf{0}$	(Res Res)	$(\boldsymbol{\nu} x)(\boldsymbol{\nu} y)P \equiv (\boldsymbol{\nu} y)(\boldsymbol{\nu} x)P$	$x \neq y$
(Path Assoc)	$(M.M').P \equiv M.(M'.P)$	(Res Par)	$(\boldsymbol{\nu} x)(P \mid Q) \equiv P \mid (\boldsymbol{\nu} x)Q$	$x \notin fn(P)$
(Repl)	$!P \equiv !P \mid P$	(Res Amb)	$(\boldsymbol{\nu} x)y[\![P]\!] \equiv y[\![(\boldsymbol{\nu} x)P]\!]$	$x \neq y$

As usual, structural congruence is used to rearrange the process so that they can reduce: $P' \equiv P,\ P \rightarrow Q,\ Q \equiv Q' \Rightarrow P' \rightarrow Q'$. Reduction is defined by the rules for mobility and communication given below plus the standard context reduction rules [CG98].

Mobility. Ambient mobility is governed by the following rules, inherited from Mobile Ambients:

$$\begin{array}{ll} (\textit{enter}) & a[\![\,\mathsf{in}\ b.P \mid Q\,]\!] \mid b[\![\,R\,]\!] \rightarrow b[\![\,a[\![\,P \mid Q\,]\!] \mid R\,]\!] \\ (\textit{exit}) & a[\![\,b[\![\,\mathsf{out}\ a.P \mid Q\,]\!] \mid R\,]\!] \rightarrow b[\![\,P \mid Q\,]\!] \mid a[\![\,R\,]\!] \end{array}$$

Communication. Communication can be local, as in Mobile Ambients, or across ambient boundaries, between parent and child.

$$\begin{array}{ll} (\textit{local}) & (x)P \mid \langle M \rangle Q \rightarrow P\{x := M\} \mid Q \\ (\textit{input } n) & (x)^n P \mid n[\![\,\langle M \rangle Q \mid R\,]\!] \rightarrow P\{x := M\} \mid n[\![\,Q \mid R\,]\!] \\ (\textit{input} \uparrow) & \langle M \rangle P \mid n[\![\,(x)^{\uparrow} Q \mid R\,]\!] \rightarrow P \mid n[\![\,Q\{x := M\} \mid R\,]\!] \\ (\textit{output } n) & \langle M \rangle^n P \mid n[\![\,(x)Q \mid R\,]\!] \rightarrow P \mid n[\![\,Q\{x := M\} \mid R\,]\!] \\ (\textit{output} \uparrow) & (x)P \mid n[\![\,\langle M \rangle^{\uparrow} Q \mid R\,]\!] \rightarrow P\{x := M\} \mid n[\![\,Q \mid R\,]\!] \end{array}$$

Four different reduction rules for parent-child communication may be thought of as redundant, especially because there are only two reducts. Instead, introducing different *directions* for input/output is a key design choice that provides the calculus with precise notions of resource locality and resource access request. Directed (towards parent or child) output captures the notion of *write access* to a resource, by identifying the ambient towards which the request is directed. Dually, directed input captures the notion of *read access*[2].

As such, the formulation given above enables the study of properties related to standard resource access control policies, even in the absence of channels (see § 5; for more details refer to [BCC01]). Channels, on the other hand, can be encoded elegantly, as we show in the next section.

As we noted, in the present formulation output is synchronous. For the time being, asynchronous output can either be considered as the special case of synchronous communication with null continuation, or else be accounted for by introducing the following rule of structural equivalence inspired by [Bou92]: $\langle M \rangle^{\eta} P \equiv \langle M \rangle^{\eta} \mid P$. As we shall discuss later on, these interpretations are equivalent, and both type sound, with "simple" types, that is, up to § 5. Instead, with "moded types" we introduce in § 6 they are different, and neither one satisfactory, as the former is too restrictive while the latter is unsound.

3 Communication Channels

We first show how to define π-calculus channels. Then, we refine the encoding to account for parent-child channeled communications. Throughout this section, we use two extra congruence laws: $(\boldsymbol{\nu} x)x[\![\,\mathbf{0}\,]\!] \equiv \mathbf{0}$ and $!P \equiv !P \mid !P$ to garbage collect inert ambients and useless replications, respectively. Neither of the two relations is derivable as a congruence law: on the other hand, they both would be derivable as observational equivalences for any reasonable notion of observable for the calculus.

3.1 π-calculus channels

We start with asynchronous output, and then use it to encode synchronous output.

[2] Other alternatives would be possible: we discuss them in § 8.

Asynchronous output. Two processes communicating over channel c have the form $c\langle y\rangle P$ (output) and $c(x)Q$ (input). For uniformity, we consider the case where output processes have continuations, and one has $c\langle x\rangle P \equiv c\langle x\rangle \mid P$. With Boxed Ambients, a first way to implement the channel c is to use the ambient $c[\![\,!(x)\langle x\rangle\,]\!]$ representing a buffer with an unbounded number of positions: the buffer simply waits for local input and, once received, releases local output. The output and input prefixes may then be represented, respectively, as the write request $\langle y\rangle^c$ and the read request $(x)^c$ on the buffer. We use a slightly different encoding based on an unbounded number of these buffers.

The following sequence of reductions shows that the encoding captures the intended behavior:

$$\begin{aligned} !c[\![\,!(x)\langle x\rangle\,]\!] \mid \langle y\rangle^c \mid (x)^c P &\equiv\ !\,c[\![\,!(x)\langle x\rangle\,]\!] \mid c[\![\,!(x)\langle x\rangle \mid (x)\langle x\rangle\,]\!] \mid \langle y\rangle^c \mid (x)^c P \\ &\rightarrow\ !\,c[\![\,!(x)\langle x\rangle\,]\!] \mid c[\![\,!(x)\langle x\rangle \mid \langle y\rangle\,]\!] \mid (x)^c P \\ &\rightarrow\ !\,c[\![\,!(x)\langle x\rangle\,]\!] \mid P\{x := y\} \end{aligned}$$

Based on these intuitions, we may define the encoding compositionally, as follows:

$$\begin{array}{lll} \langle\!\langle\, (\boldsymbol{\nu} x)P \,\rangle\!\rangle = (\boldsymbol{\nu} x)\,\langle\!\langle\, P \,\rangle\!\rangle & \langle\!\langle\, !P \,\rangle\!\rangle = !\,\langle\!\langle\, P \,\rangle\!\rangle & \langle\!\langle\, c\langle x\rangle P \,\rangle\!\rangle = !c[\![\,!(x)\langle x\rangle\,]\!] \mid \langle x\rangle^c \mid \langle\!\langle\, P \,\rangle\!\rangle \\ \langle\!\langle\, P \mid Q \,\rangle\!\rangle = \langle\!\langle\, P \,\rangle\!\rangle \mid \langle\!\langle\, Q \,\rangle\!\rangle & \langle\!\langle\, \mathbf{0} \,\rangle\!\rangle = \mathbf{0} & \langle\!\langle\, c(x)P \,\rangle\!\rangle = !c[\![\,!(x)\langle x\rangle\,]\!] \mid (x)^c\,\langle\!\langle\, P \,\rangle\!\rangle \end{array}$$

An input and an output on the same channel, say c, generate multiple copies of the buffer $!c[\![\,!(x)\langle x\rangle\,]\!]$: this is not a problem, however, since the multiple copies can be garbage collected by the congruence law $!P \mid !P \equiv !P$. Of course, there are other ways for structuring the encoding. For instance, one could create the buffer when the channel name is introduced by a restriction: this would avoid the proliferation of channels. A further alternative would be to collect free and bound names in the inductive cases of the translation, as in $\langle\!\langle\, (\boldsymbol{\nu} x)P \,\rangle\!\rangle_{\Delta} = (\boldsymbol{\nu} x)\,\langle\!\langle\, P \,\rangle\!\rangle_{\Delta\cup\{x\}}$, $\langle\!\langle\, c(x)P \,\rangle\!\rangle_{\Delta} = (x)^c\,\langle\!\langle\, P \,\rangle\!\rangle_{\Delta\cup\{x\}}$, and introduce them in the base cases[3]: $\langle\!\langle\, \mathbf{0} \,\rangle\!\rangle_{\{c_1,\ldots,c_n\}} = !c_1[\![\,!(x)\langle x\rangle\,]\!] \mid \ldots \mid !c_n[\![\,!(x)\langle x\rangle\,]\!]$. With this encoding, we could prove that if P reduces to Q in the π-calculus, then the same is true of the encoding of P and Q, and conversely, if an encoding of P reduces to a process then this is equivalent to the encoding of a reductum of P.

However, none of the alternatives would scale to the case of communication with local channels of § 3.2: this justifies our choice of the encoding given above.

Synchronous output. Let now $cx.P$ and $\bar{c}x.Q$ denote π-calculus *synchronous* input and output on channel c. Synchronous input-output can be encoded in the asynchronous polyadic π-calculus as follows: $\bar{c}x.P = (\boldsymbol{\nu} r)c\langle x, r\rangle r()P$, $cy.Q = c(y,r)r\langle\rangle P$ where $r \notin \mathit{fn}(P)$. That is, the output process sends the intended message as well as a private channel r for the reader to acknowledge receipt of the message.

The encoding of the synchronous π-calculus with Boxed Ambients is obtained by simply composing the two encodings and simplifying the result[4]. One only needs a different definition for the prefix forms:

$$\begin{array}{ll} \langle\!\langle\, \bar{c}x.P \,\rangle\!\rangle = (\boldsymbol{\nu} r)(r[\![\,()\langle\rangle\,]\!] \mid \langle x, r\rangle^c ()^r\,\langle\!\langle\, P \,\rangle\!\rangle) & r \notin \mathit{fn}(P) \\ \langle\!\langle\, cy.Q \,\rangle\!\rangle = !c[\![\,!(u,v)\langle u,v\rangle\,]\!] \mid (y,s)^c\langle\rangle^s\,\langle\!\langle\, Q \,\rangle\!\rangle & s \notin \mathit{fn}(Q) \end{array}$$

The rest is unchanged.

[3] Of course, for parallel composition, one has to arbitrarily choose one subprocess; for example the left one: $\langle\!\langle\, P \mid Q \,\rangle\!\rangle_{\Delta} = \langle\!\langle\, P \,\rangle\!\rangle_{\Delta} \mid \langle\!\langle\, Q \,\rangle\!\rangle_{\emptyset}$

[4] To make it more readable we erase duplicates and replace sequential composition for some parallel ones.

3.2 Parent-child channeled communication à la Seal Calculus

Having looked at π-calculus channels, we now discuss an extension of the encoding that conforms with the notion of locality and directed input-output of the core calculus. The extension yields a notion of located channels and a set of communication protocols that are similar, in spirit, to those given as primitive in the Seal Calculus [VC99]. In the Seal Calculus, one can express output prefixes of the form $c^n\langle M\rangle$ requesting a write access on the channel c residing in ambient (or seal) n. Dually, the input prefix $c^{\uparrow}(x)$ denotes a read request on the channel c residing in the parent ambient. Upward output and downward input on local channels may be expressed in similar ways. All these communication protocols can be expressed in the core calculus: below, we only consider only the asynchronous case (i.e. continuationless outputs) and we give detailed descriptions of downward output and upward input.

The intended reduction of a downward output is:

$$c^n\langle M\rangle \mid n[\![\, c(x)P \mid Q \,]\!] \rightarrow n[\![\, P\{x := M\} \mid Q \,]\!] .$$

The channel c is local to n, and the outer process requests a write access on c. There are several ways that the reduction can be captured with the existing constructs. Here, we describe an encoding that renders the locality of c. The channel c is represented as before as a buffer, and the input prefix $c(x)$ as read access request to c:

$$c(x)P \triangleq !c[\![\, !(x)\langle x\rangle \,]\!] \mid (x)^c P$$

Now, however, the write access to c cannot be represented directly as we did above for the π-calculus channel, because c is located into n. To capture the desired behavior we can rely on mobility:

$$c^n\langle M\rangle \triangleq (\boldsymbol{\nu} p)p[\![\, \mathsf{in}\ n.\mathsf{in}\ c.\langle M\rangle^{\uparrow} \,]\!]$$

The output M is encapsulated into a transport ambient p, which enters n and then c to deliver the message (the name of the transport ambient p must be fresh). Thus, the Seal Calculus process $c^n\langle M\rangle \mid n[\![\, c(x)P \mid Q \,]\!]$ is encoded as follows:

$$(\boldsymbol{\nu} p)p[\![\, \mathsf{in}\ n.\mathsf{in}\ c.\langle M\rangle^{\uparrow} \,]\!] \quad\mid\quad n[\![\, !c[\![\, !(x)\langle x\rangle \,]\!] \mid (x)^c P \mid Q \,]\!]$$

By a sequence of reductions, the process above evolves into

$$n[\![\, !c[\![\, !(x)\langle x\rangle \,]\!] \mid c[\![\, !(x)\langle x\rangle \mid (\boldsymbol{\nu} p)p[\![\, \mathbf{0} \,]\!] \,]\!] \mid P\{x := M\} \mid Q \,]\!],$$

which is equivalent to $n[\![\, !c[\![\, !(x)\langle x\rangle \,]\!] \mid P\{x := M\} \mid Q \,]\!]$ by structural congruence.

Remote inputs are slightly more complex, since the transport ambient must fetch the output and bring it back. The intended reduction is $c\langle M\rangle \mid n[\![\, c^{\uparrow}(x)P \mid Q \,]\!] \rightarrow n[\![\, P\{x := M\} \mid Q \,]\!]$. Upward input from within a seal n is simulated in Boxed Ambients as

$$c^{\uparrow}(x)P \triangleq (\boldsymbol{\nu} p)p[\![\, \mathsf{out}\ n.\mathsf{in}\ c.(x)^{\uparrow}\mathsf{out}\ c.\mathsf{in}\ n.\langle x\rangle \,]\!] \mid (x)^p P$$

Note that the definition depends on the name n of the enclosing ambient. For a formal definition, it is enough to keep track of this information, and extend the encoding of the asynchronous π calculus with the following clauses:

$$\begin{array}{lcl}
\langle\!\langle\, c^m\langle x\rangle \,\rangle\!\rangle_n & = & (\boldsymbol{\nu} p)p[\![\, \mathsf{in}\ m.\mathsf{in}\ c.\langle x\rangle^{\uparrow} \,]\!] \\
\langle\!\langle\, c^{\uparrow}\langle x\rangle \,\rangle\!\rangle_n & = & (\boldsymbol{\nu} p)p[\![\, \mathsf{out}\ n.\mathsf{in}\ c.\langle x\rangle^{\uparrow} \,]\!] \\
\langle\!\langle\, c^m(x)P \,\rangle\!\rangle_n & = & (\boldsymbol{\nu} p)p[\![\, \mathsf{in}\ m.\mathsf{in}\ c.(x)^{\uparrow}\mathsf{out}\ c.\mathsf{out}\ m.\langle x\rangle \,]\!] \mid (x)^p \langle\!\langle\, P \,\rangle\!\rangle_n \\
\langle\!\langle\, c^{\uparrow}(x)P \,\rangle\!\rangle_n & = & (\boldsymbol{\nu} p)p[\![\, \mathsf{out}\ n.\mathsf{in}\ c.(x)^{\uparrow}\mathsf{out}\ c.\mathsf{in}\ n.\langle x\rangle \,]\!] \mid (x)^p \langle\!\langle\, P \,\rangle\!\rangle_n \\
\langle\!\langle\, a[\![\, P \,]\!] \,\rangle\!\rangle_n & = & a[\![\, \langle\!\langle\, P \,\rangle\!\rangle_a \,]\!]
\end{array}$$

4 Typing Boxed Ambients

As we stated at the outset, one of the goals in the design of Boxed Ambients was to enhance static reasoning on ambient and process behavior, by enabling focused and precise analyses while preserving the expressive power of the calculus. The definition of the type system, given in this section, proves that the design satisfies these requirements

A rather simple structure of types suffices to provide precise accounts of process behavior. Ambient and process types are defined simply as two-place constructors describing the types of the exchanges that may take place locally and with the enclosing context. Interestingly, this simple type structure is all that is needed to give a full account of ambient interaction. This is a consequence of (i) there being no way for ambients to communicate directly across more than one boundary, and (ii) communication being the only means for ambient to interact.

4.1 Types

The structure of types is defined by the following productions.

Expression Types

W	$::=$	$\mathsf{Amb}[E,F]$	ambient
	$\mid$	$\mathsf{Cap}[E]$	capability
	$\mid$	$W_1 \times \cdots \times W_n$	tuple

Exchange Types

E, F	$::=$	shh	no exchange
	$\mid$	W	exchange

Process Types

T	$::=$	$\mathsf{Pro}[E,F]$	composite exchange

The type structure is superficially similar to that of companion type systems for the Ambient Calculus [CG99,CGG99]. The interpretation, however, is different.

$\mathsf{Amb}[E,F]$	ambients that enclose processes of type $\mathsf{Pro}[E,F]$,
$\mathsf{Cap}[E]$	capabilities exercised within ambients with E upward exchanges,
$\mathsf{Pro}[E,F]$	processes whose local and upward exchanges have types E and F, respectively.

Notice that capability types are defined as one-place constructors, and disregard the local exchanges of the ambients where they are exercised. This is because (i) exercising a capability within an ambient, say a, may only cause a to move, and (ii) the safety of ambient mobility may be established regardless of the ambient's local exchanges.

As for process types, a few examples help explain the intuition about composite exchange. We use a Church style typed syntax, in which all inputs and restrictions specify the type of the bound variable: more precisely we use $(\boldsymbol{\nu} x : W)P$ and $(x : W)P$ instead of $(\boldsymbol{\nu} x)P$ and $(x)P$, respectively.

$(x : W)\langle x\rangle$: $\mathsf{Pro}[W, \mathsf{shh}]$. W is exchanged (read and written) locally, and there is no upward exchange.

$(x : W)^{\uparrow}\langle x\rangle^{n}$: $\mathsf{Pro}[\mathsf{shh}, W]$. W is exchanged (i.e. read from) upwards, and then written to ambient n. There is no local exchange, hence the type shh as the first component of the process type. For the typing to be derivable, one needs $n : \mathsf{Amb}[W, E]$ for some exchange type E.

$(x : W)^{\uparrow}(y : W')(\langle x\rangle^{n} \mid \langle y\rangle)$: $\mathsf{Pro}[W', W]$. W is exchanged (read from) upwards, and then forwarded to ambient n, while W' is exchanged (read and written) locally. Again, for the typing to be derivable, one needs $n : \mathsf{Amb}[W, E]$ for some exchange type E.

$(x : W)\langle x\rangle^{\uparrow}$: $\mathsf{Pro}[W, W]$. W is read locally, and written upwards.

These simple examples give a flavor of the flexibility provided by the constructs for communication: like mobile ambients, boxed ambients are "places of conversation", but unlike ambients they allow more than just one "topic" of conversation. This is made possible by the local nature of (anonymous) channels, and the consequent "directed" forms of input/output. Specifically, every ambient may exchange values of different types with any of its children, as long as the exchange is directed from the ambient to the children. Instead, upward communication is more constrained: all the children must agree on the (unique) type of exchange they may direct to their parent.

4.2 Typing Rules

The judgments of the type system have two forms: $\Gamma \vdash M : W$, read "*expression M has type W*", and $\Gamma \vdash P : T$, read "*process P has type T*". Accordingly we have two sets of typing rules, one for names and capabilities, one for processes. In addition, we introduce a subsumption rule for process types, based on the following definition of subtyping.

Definition 1 (*Subtyping*). *We denote by $\leqslant$ the smallest reflexive and transitive relation over exchange types such that* $\mathsf{shh} \leqslant E$ *for every exchange type E. Exchange subtyping is lifted to process types as follows:* $\mathsf{Pro}[\mathsf{shh}, F] \leqslant \mathsf{Pro}[E, F]$.

Process subtyping is used in conjunction with subsumption, exchange subtyping is not. The intuition for exchange subtyping is that a (locally or upward) shh exchange is always type compatible with a situation in which some exchange is expected: this is useful in the typing of capabilities. As for process subtyping, it would be tempting to extend the subtyping relation so as to allow subtyping over upward exchanges, as well. However, as we explain later in this section, uses of this relation in conjunction with a subsumption rule for process types would not be sound. As a final remark, note that our notion of subtyping is quite shallow: it is "almost equality" as there is no *deep* subtyping. This holds true for the moded types of Section 6, as well.

Typing of Expressions

(PROJECTION)
$$\frac{\Gamma(n) = W}{\Gamma \vdash n : W}$$

(TUPLE)
$$\frac{\Gamma \vdash M_i{:}W_i \quad \forall i \in 1..k}{\Gamma \vdash (M_1, ..., M_k) : W_1 \times ... \times W_k}$$

(PATH)
$$\frac{\Gamma \vdash M_1{:}\mathsf{Cap}[F] \quad \Gamma \vdash M_2{:}\mathsf{Cap}[F]}{\Gamma \vdash M_1.M_2 : \mathsf{Cap}[F]}$$

(IN)
$$\frac{\Gamma \vdash M : \mathsf{Amb}[F, E] \quad F' \leqslant F}{\Gamma \vdash \mathsf{in}\ M : \mathsf{Cap}[F']}$$

(OUT)
$$\frac{\Gamma \vdash M : \mathsf{Amb}[E, F] \quad F' \leqslant F}{\Gamma \vdash \mathsf{out}\ M : \mathsf{Cap}[F']}$$

The (PROJECTION), (TUPLE), and (PATH) rule are standard. The rules (IN) and (OUT) define the constraints for safe ambient mobility, and explain why capability types are built around a single component. The intuition is as follows: take a capability, say $\mathsf{in}\ n$ with $n : \mathsf{Amb}[F, E]$, and suppose that this capability is exercised within ambient, say, m. If m has upward exchanges of type F', then $\mathsf{in}\ n : \mathsf{Cap}[F']$. Now, for the move of m into n to be safe, one must ensure that the type F of the local exchanges of n be equal to the type F' of the upward exchanges of m. In fact, the typing can be slightly more flexible, for if m has no upward exchange, then $F' = \mathsf{shh} \leqslant F$, and m may safely move into n. Dual reasoning applies to the (OUT) rule: the upward exchanges of the exiting ambient must have type $\leqslant$-compatible with the type of the upward exchanges of the ambient being exited.

Typing of Processes

(DEAD) $$\frac{}{\Gamma \vdash \mathbf{0} : T}$$

(NEW) $$\frac{\Gamma, x : W \vdash P : T}{\Gamma \vdash (\boldsymbol{\nu} x : W)P : T}$$

(PARALLEL) $$\frac{\Gamma \vdash P : \mathsf{Pro}[E,F] \quad \Gamma \vdash Q : \mathsf{Pro}[E,F]}{\Gamma \vdash P \mid Q : \mathsf{Pro}[E,F]}$$

(PREFIX) $$\frac{\Gamma \vdash M : \mathsf{Cap}[F] \quad \Gamma \vdash P : \mathsf{Pro}[E,F]}{\Gamma \vdash M.P : \mathsf{Pro}[E,F]}$$

(AMB) $$\frac{\Gamma \vdash M : \mathsf{Amb}[E,F] \quad \Gamma \vdash P : \mathsf{Pro}[E,F]}{\Gamma \vdash M[\![\, P \,]\!] : \mathsf{Pro}[F,G]}$$

(SUBSUM PROC) $$\frac{\Gamma \vdash P : T \quad T \leqslant T'}{\Gamma \vdash P : T'}$$

(REPLICATION) $$\frac{\Gamma \vdash P : \mathsf{Pro}[E,F]}{\Gamma \vdash\, !P : \mathsf{Pro}[E,F]}$$

(DEAD), (NEW), (PARALLEL), (REPLICATION) and the subsumption rule are standard[5]. In the (PREFIX) rule, the typing of the capability M ensures, via the (IN), (OUT), and (PATH) rules introduced earlier, that each of the ambients being traversed as a result of exercising M have local exchanges of type compatible with the upward exchanges of the current ambient.

The rule (AMB) establishes the constraints that must be satisfied by P to be enclosed in M: specifically, the exchanges declared for M must have the same types E and F as the exchanges of P. In fact, P could be locally silent, and the typing of $M[\![\, P \,]\!]$ be derivable from $\Gamma \vdash P : \mathsf{Pro}[\mathsf{shh}, F]$ by subsumption. In addition, if $\Gamma \vdash M : \mathsf{Amb}[E, \mathsf{shh}]$, and $\Gamma \vdash P : \mathsf{Pro}[E, \mathsf{shh}]$, then by (AMB) and subsumption one derives $\Gamma \vdash M[\![\, P \,]\!] : \mathsf{Amb}[F, G]$ for any F and G, as the rule imposes no constraint on the upward exchanges of the process $M[\![\, P \,]\!]$.

(INPUT $\star$) $$\frac{\Gamma, x : W \vdash P : \mathsf{Pro}[W,E]}{\Gamma \vdash (x : W)P : \mathsf{Pro}[W,E]}$$

(OUTPUT $\star$) $$\frac{\Gamma \vdash M : W \quad \Gamma \vdash P : \mathsf{Pro}[W,E]}{\Gamma \vdash \langle M \rangle P : \mathsf{Pro}[W,E]}$$

(INPUT M) $$\frac{\Gamma \vdash M : \mathsf{Amb}[W,E] \quad \Gamma, x : W \vdash P : T}{\Gamma \vdash (x : W)^M P : T}$$

(OUTPUT M) $$\frac{\Gamma \vdash M : \mathsf{Amb}[W,E] \quad \Gamma \vdash N : W \quad \Gamma \vdash P : T}{\Gamma \vdash \langle N \rangle^M P : T}$$

(INPUT $\uparrow$) $$\frac{\Gamma, x : W \vdash P : \mathsf{Pro}[E,W]}{\Gamma \vdash (x : W)^{\uparrow} P : \mathsf{Pro}[E,W]}$$

(OUTPUT $\uparrow$) $$\frac{\Gamma \vdash M : W \quad \Gamma \vdash P : \mathsf{Pro}[E,W]}{\Gamma \vdash \langle M \rangle^{\uparrow} P : \mathsf{Pro}[E,W]}$$

The rules for input/output are not surprising. In all cases, the type of the exchange must comply with the local exchange type of the target ambient, as expected. Also note that input/output exchanges with children, in the rules (INPUT M) and (OUTPUT M), do not impose any constraint on local and upward exchanges.

As we noted earlier, type soundness requires that subtyping between upward silent and upward non-silent processes be disallowed. To see why, consider for example allowing the relation $\mathsf{Pro}[E, \mathsf{shh}] \leqslant \mathsf{Pro}[E, F]$, implying that upward-silent processes may be subsumed to non-silent

[5] The reason why in (PARALLEL) and (REPLICATION) we used $\mathsf{Pro}[E,F]$ rather than T will become clear in Section 6

processes with any upward exchange type F. While this form of subsumption seems reasonable, it is unsound in the presence of parallel composition. Consider the ambient $a[\![\,\mathsf{in}\ b.\mathbf{0} \mid \langle M\rangle^{\uparrow} P\,]\!]$ with, say $M : W$ for some type W, and note that $\mathsf{in}\ b.\mathbf{0}$ can be typed as $\mathsf{Pro}[\mathsf{shh}, \mathsf{shh}]$ regardless of the type of b. If the suggested subtyping were available, then the parallel composition could be typed as $\mathsf{Pro}[\mathsf{shh}, W]$. However, if $b : \mathsf{Amb}[W', F]$ for some $W' \neq W$, the ambient a could move into b and have unsound upward exchanges after the move. By forbidding subtyping on the upper component of process types, instead, the types that can be deduced for the process $\mathsf{in}\ b.\mathbf{0}$ above may only be of the form $\mathsf{Pro}[E\,, W']$ or $\mathsf{Pro}[E, \mathsf{shh}]$ for some exchange E.

The type system rules ensures that communication inside and across ambients never leads to type mismatches. The latter result is a consequence of the subject reduction property stated next.

Theorem 1 (Subject Reduction). *If $\Gamma \vdash P : T$ and and $P \rightarrow Q$, then $\Gamma \vdash Q : T$.*

Proof. By induction on the derivation of $P \rightarrow Q$, and appeal to standard lemma of substitution and subject congruence.

5 Mobile Ambients versus Boxed Ambients

We now look at the impact of typing on mobility and communication, and contrast it with mobility and communication of Mobile Ambients.

We already noted that type safety for ambient mobility can be established irrespective of local exchanges. On the other hand, upward communication does impose somewhat restrictive constraints over ambient mobility. Specifically, ambients with upward exchanges of type W may only traverse ambients whose local exchanges also have type W. However, when we compare the flexibility of mobility and communication in Boxed Ambients versus the corresponding constructs found in Mobile Ambients, we find that typed Mobile Ambients have, in fact, even more severe constraints.

To see that, it is instructive to note that the type system of the previous section can be specialized to only allow upward-silent ambient types in the form $\mathsf{Amb}[E, \mathsf{shh}]$. This effectively corresponds to inhibiting all forms of upward exchanges: this follows from the format of the (AMB) rule. On the other hand, it provides full flexibility for mobility, while still allowing powerful forms of communication. We may note the following of the specialized type system.

- *Mobility for Boxed Ambients is as flexible as mobility for typed Mobile Ambients.* This follows by the (IN) and (OUT) rules discussed in § 4.2. Capabilities exercised within upward silent ambients have type $\mathsf{Cap}[\mathsf{shh}]$, and $\mathsf{shh} \leqslant F$ for every F: consequently, upward silent ambients have full freedom of moving across ambient boundaries. Furthermore, since Boxed Ambients may not be opened, they may move regardless of the local exchanges of the ambients they traverse. As a consequence, with the specialized type system, an ambient can move independently of its type, and of the type of its (intermediate and final) destinations.
- *Communication is more flexible than in the Ambient Calculus, even in the absence of upward exchanges*. "Upward silent" does not imply "non communicating": an upward-silent ambient may very well move to a target ambient a, and communication between a and the incoming ambient may rely on a accessing the incoming ambient by downward requests. Indeed, an ambient may access all of its children's anonymous channels as well as those of any incoming ambient: all these exchanges may be of different types. Besides, the ambient may hold local exchanges of yet a different type. The encoding of channels given § 3.1 can also be used for encoding local exchanges of different types: the ambient $c[\![\,!(x\,{:}W)\langle x\rangle\,]\!]$ can be viewed as a local channel c of type W, whose input output operators are $(x\,{:}W)^c$ and $\langle M\rangle^c$: the type system allows (encoded) channels of different types to be used in the same ambient.

In the ambient calculus, instead, parent-child communication requires the parent to first open the child (or else a "messenger" ambient [CG98] exiting the child). As a consequence, either the parent, and all the children's exchanges have the same type, or there is no way that the parent may communicate with all of its children.

5.1 Security and Resource Access Control

The communication model over which Boxed Ambients are defined has other interesting payoffs when it comes to security and resource protection policies.

As we have argued, the primitives for communication have immediate and very natural interpretations as access requests: for example, the input prefix $(x)^n$ can be seen as a request to read from (the anonymous channel located into) child ambient n and, dually, $\langle M\rangle^{\uparrow}$ can be interpreted as write request to the parent ambient (equivalently, its local channel). Based on that, Boxed Ambients provide for a direct characterization of classical resource access control mechanisms, such as *Mandatory Access Control* or MAC policies. In addition, *multilevel security*, and the associated *Military* (no read-up, no write-down) and *Commercial* (no read-up, no write-up) security models may directly be accounted for by embedding security levels into types, and using typing rules to statically enforce the desired constraints on access. For an thorough discussion of MAC multilevel security for Boxed Ambients the reader is referred to [BCC01]. What is interesting to note here, instead, is that the mechanisms for ambient interaction and communication fit nicely and complement the security model of Mobile Ambients, which predicates in/out access to ambients on possession of appropriate passwords or cryptokeys.

The download example, revisited. To exemplify, consider again the download example in § 1. With Mobile Ambients, security relies solely on authorization based on knowledge of names: the agent $a[\![\, Q \,]\!]$ acquires authorization to enter the host h by knowing the name h and embedding it into the capability $\mathsf{in}\ b$: the capability, or the name, may thus be seen as passwords that enable the access to h, as in $h[\![\, a[\![\, Q \,]\!] \mid P \,]\!]$. Once inside h, the ambient a (or a messenger ambient exiting a) is dissolved to enable interaction. As we argued, this may be upsetting to the host, as it grants Q (or the messenger inside a) indiscriminate access to whatever is inside h.

Instead, if a and h are Boxed Ambients, authorization by possession of capabilities can be complemented by finer-grained control over the access requests by Q to the contents of h. Assume, for the purpose of the example, that h encapsulates its resources in a set of subambients $r_1, \ldots, r_n$. Then P inside h could mediate the access requests by a to each of the r_i's by means of an interface process of the form $(x{:}W)^a\langle x\rangle^{r_i}$. In addition, the incoming agent could be forced to be upward silent to prevent it from interfering with the local exchanges held within h: this can be accomplished by imposing a suitable security policy, based on typing, as shown in [BCC01].

5.2 Discussion

Having argued in favor of the communication model of Boxed Ambients with specialized type system, it is obvious that giving up upward exchanges is a problem: for instance, we would not be able to type-check "transport" ambients, such as those used in the encoding of the Seal Calculus' channeled communications of § 3.2, whose function is to silently carry a process to a certain destination where the process will eventually deliver its output to and/or receive input from its enclosing context. As we show in the next section, it is actually possible to refine and extend the type system to support a smoother and type safe interaction of upward communication and mobility.

6 Moded Typing

The typing technique we develop in this section is based on a refinement of the observation we just made of the specialized type system, namely that ambients enclosing upward-silent processes may safely move across other ambients, regardless of the types of the latter. The new type system uses type modifiers to characterize the computation progress of processes, and in particular, to identify the silent and non-silent phases in the computation of the processes enclosed within ambients: based on that, it enhances the typing of mobility during the ambients' silent phases.

6.1 Moded Types

The new type system is built around the extended classes of process and expression types defined below:

$$\textit{Process Types} \quad T ::= \mathsf{Pro}[E, F] \mid \mathsf{Pro}[E, {}^{\bullet}F] \mid \mathsf{Pro}[E, {}^{\circ}F] \mid \mathsf{Pro}[E, {}^{\vartriangle}F]$$

$$\textit{Expression Types}\ W ::= \mathsf{Amb}[E, F] \mid \mathsf{Amb}^{\circ}[E, F] \mid \mathsf{Cap}[E] \mid W_1 \times \cdots \times W_n$$

Ambient types of the form $\mathsf{Amb}[E, F]$ are exactly as in § 4, and their enclosed processes have "regular" process types $\mathsf{Pro}[E, F]$, deduced by the same rules. On the other hand, ambient types of the form $\mathsf{Amb}^{\circ}[E, F]$ are associated with "transport" (or moded) ambients, whose enclosed processes are assigned moded types, according to the following intuitions:

- $\mathsf{Pro}[E, {}^{\bullet}W]$: upward silent processes with local exchanges of type E. The type W signals that processes with this type may be safely run in parallel with processes with upward exchanges of type W.
- $\mathsf{Pro}[E, {}^{\circ}W]$: processes with local exchanges of type E and upward exchanges of type W. The upward exchanges are temporarily inactive since the process is moving.
- $\mathsf{Pro}[E, {}^{\vartriangle}W]$: processes with local exchanges of type E and that, after performing upward exchanges of type W, evolve into processes of type $\mathsf{Pro}[E, {}^{\circ}W]$ or $\mathsf{Pro}[E, {}^{\vartriangle}W]$.

The syntax allows the formation of process types of the form $\mathsf{Pro}[E, {}^{\bullet}\mathsf{shh}]$, $\mathsf{Pro}[E, {}^{\circ}\mathsf{shh}]$ and $\mathsf{Pro}[E, {}^{\vartriangle}\mathsf{shh}]$. These types are convenient in stating definitions and typing rules: to make sense of them, we stipulate that ${}^{\bullet}\mathsf{shh} = {}^{\circ}\mathsf{shh} = {}^{\vartriangle}\mathsf{shh} = \mathsf{shh}$. To exemplify moded types, consider the following process, where we assume $\Gamma \vdash M : W$.

$$(x{:}W')\langle x\rangle^{m} \mid \mathsf{in}\ n.\langle M\rangle^{\uparrow}.\mathsf{out}\ n : \mathsf{Pro}[W', {}^{\circ}W].$$

The left component of this process does not have upward exchanges, hence it can be assigned the type $\mathsf{Pro}[W', {}^{\bullet}W]$ provided, of course, that $m : \mathsf{Amb}[W', E]$ for some E. On the other hand, the right component does have upward exchanges, but is currently silent because the output prefix is blocked by the move: thus $\mathsf{in}\ n.\langle M\rangle^{\uparrow}.\mathsf{out}\ n : \mathsf{Pro}[W', {}^{\circ}W]$, provided that $n : \mathsf{Amb}[W, W]$. The type $\mathsf{Pro}[W', {}^{\circ}W]$ can also be assigned to the parallel composition which is, in fact, currently silent. Interestingly, the type $\mathsf{Pro}[W', {}^{\circ}W]$ can *not* be assigned to the continuation process $\langle M\rangle^{\uparrow}.\mathsf{out}\ n$ (nor to the parallel composition $(x{:}W')\langle x\rangle^{m} \mid \langle M\rangle^{\uparrow}.\mathsf{out}\ n$), because, after consuming the capability $\mathsf{in}\ b$, the upward exchanges of this process are active. At this stage, a legal type for the process is $\mathsf{Pro}[W', {}^{\vartriangle}W]$, signaling, that after the upward exchange, the process enters again an upward-silent phase.

As the example shows, processes that are subject to moded typing may have different types at different stages of their computation. This does not break subject reduction, as it would seem, as reductions involving the consumption of capabilities only involve the ambients enclosing the capabilities being consumed: as a consequence, while the process enclosed in an ambient changes its type according to the process' progress, the type of the ambient itself is invariant through reduction.

The reader may wonder whether the new class of "transport" ambients is really necessary, and why the same effect can not be obtained by solely relying on "regular" ambient types. The problem is that moded typing is not powerful enough to control mobility: in particular, moded types can not be employed to prevent non-silent ambients to exit their parent during the upward-silent phases of the latter. To see the problem, assume that ambient, say a, is currently silent and moving across ambients with local exchanges, say W. Also assume that a contains a non-silent ambient b with upward exchanges of type W' incompatible with W. As long as b is enclosed in a, its upward exchanges do not interfere with the local exchanges W of the ambients traversed by a. But if b exits a, then its upward exchanges may cause a type mismatch. In our system[6], the problem is solved by providing guarantees that transport ambients can only be exited by (regular or transport) ambients whose upward exchanges have type shh.

6.2 Capabilities and Moded Judgments

The modes attached to process types also affect the typing of capabilities. This is accounted for by a new form of judgment, denoted by $\Gamma \vdash\!\circ\, M : \mathsf{Cap}[E]$. This notation indicates a "silent mode" for typing the capability M, which is useful when typing capability paths: if typed in silent mode, every intermediate move on the path may safely disregard the type of the ambient traversed along the move.

6.3 Typing Rules

The new type system includes all the typing rules from § 4.2. In addition, we have a richer subtype structure for process types, and new rules for deriving silent typings of capabilities, and moded types for processes.

Definition 2 (*Process Subtyping).*
Let $\leqslant$ denote the same relation of exchange subtyping of Definition 1. Process subtyping is the smallest reflexive and transitive relation such that $\mathsf{Pro}[\mathsf{shh}, {}^{*}F] \leqslant \mathsf{Pro}[E, {}^{*}F]$ *and in addition, satisfies the diagram on the right for all E and F.*

$$
\begin{tikzcd}
& \mathsf{Pro}[E, {}^{\vartriangle}F] & \\
\mathsf{Pro}[E, F] \arrow[ur] & & \mathsf{Pro}[E, {}^{\circ}F] \arrow[ul] \\
& \mathsf{Pro}[E, {}^{\bullet}F] \arrow[ul] \arrow[ur] &
\end{tikzcd}
$$

The intuition underlying process subtyping is as follows. As we said, the type $\mathsf{Pro}[\_, {}^{\bullet}E]$ identifies upward-silent processes that move their enclosing ambient only through locations with local exchanges of type E. Clearly, any such process can always be considered as a process of type $\mathsf{Pro}[\_, E]$ that is, as a process whose all upward exchanges are of type E and that moves the enclosing ambient only through locations with local communications of type E. In fact, it can also be considered as a process of type $\mathsf{Pro}[\_, {}^{\circ}E]$, that is as a temporary upward-silent process that guarantees its enclosing ambient that whenever it performs an upward communication it will be in a context with local exchanges of type E. The two types $\mathsf{Pro}[\_, E]$ and $\mathsf{Pro}[\_, {}^{\circ}E]$ are incompatible, as processes of the first type may not be assumed to be (even temporary) upward-silent, while processes of the second type may move across ambients regardless of the types of the latter and therefore across ambients whose local exchanges are of a type different from E. Nevertheless, the two types have a common supertype $\mathsf{Pro}[\_, {}^{\vartriangle}E]$, as this type identifies processes that may be currently upward-active, and whose enclosing ambients are guaranteed to reside in contexts with local exchanges of type E.

Typing for Expressions. We use the following notation and conventions: ${}^{*}W$ denotes any of the exchanges ${}^{\vartriangle}W, {}^{\bullet}W, {}^{\circ}W$, while ${}^{?}W$ denotes either ${}^{*}W$ or W; when occurring in definitions and typing rules, the notations ${}^{*}W$ and ${}^{?}W$ are intended to be used uniformly (i.e., all the

[6] A different solution would be possible by extending the calculus with co-capabilities à la *Safe Ambients* [LS00]. In that case, an ambient would be in a silent phase when its enclosed process does not perform upward exchanges and does not offer a *co-out* capability for nested ambients to exit.

occurrences of $*$ and ? in a rule or in the definition denote the same symbol, unless otherwise stated).

The key rules that characterize moded ambients are those that govern mobility into and out from a moded ambient:

$$\text{(OUT} \circ) \quad \frac{\Gamma \vdash M : \mathsf{Amb}^{\circ}[E,F]}{\Gamma \vdash \mathsf{out}\ M : \mathsf{Cap}[\mathsf{shh}]} \qquad \text{(IN} \circ) \quad \frac{\Gamma \vdash M : \mathsf{Amb}^{\circ}[F,E] \quad F' \leqslant F}{\Gamma \vdash \mathsf{in}\ M : \mathsf{Cap}[F']}$$

Note that while there is no constraint for entering a moded ambient —as the rule (IN $\circ$) imposes exactly the same restrictions as the rule (IN)— the rule (OUT $\circ$) requires that if M is moded, then out M can only be exercised in ambients that are upward silent.
The next rules are those that relate and differentiate $\vdash\!\circ$ from $\vdash$.

$$\text{(POLYCAP)} \quad \frac{\Gamma \vdash M : \mathsf{Cap}[E]}{\Gamma \vdash\!\circ M : \mathsf{Cap}[E]} \qquad \text{(POLYPATH)} \quad \frac{\Gamma \vdash\!\circ M_1 : \mathsf{Cap}[E_1] \quad \Gamma \vdash\!\circ M_2 : \mathsf{Cap}[E_2]}{\Gamma \vdash\!\circ M_1.M_2 : \mathsf{Cap}[E_2]}$$

The rule *PolyCap* states that for all capabilities, typing and moded-typing coincide. In addition, for capability paths —that is, for sequences of in and out moves— we have the special, and more flexible rule (POLYPATH) stating that we may disregard intermediate steps, as no communication takes place during those steps: we only need to trace precise information on the last move on the path. This effectively corresponds to interpreting $\mathsf{Cap}[E]$ as the type of capability paths whose *last* move requires upward exchanges of type E.

Moded typing of capabilities helps derive moded process types for prefixed processes as illustrated by the rules below[7].

Typing of Processes
As we said, the new type system includes all the typing rules for processes in § 4.2. In addition, we have the following rules. We start with the typing of prefixes.

$$\text{(PREFIX} \circ) \quad \frac{\Gamma \vdash\!\circ M : \mathsf{Cap}[G] \quad \Gamma \vdash P : \mathsf{Pro}[E, {}^{\circ}F]}{\Gamma \vdash M.P : \mathsf{Pro}[E, {}^{\circ}F]} \qquad \text{(PREFIX} \vartriangle) \quad \frac{\Gamma \vdash\!\circ M : \mathsf{Cap}[F] \quad \Gamma \vdash P : \mathsf{Pro}[E, {}^{\vartriangle}F]}{\Gamma \vdash M.P : \mathsf{Pro}[E, {}^{\circ}F]}$$

$$\text{(PREFIX} \bullet) \quad \frac{\Gamma \vdash M : \mathsf{Cap}[F] \quad \Gamma \vdash P : \mathsf{Pro}[E, {}^{\bullet}F]}{\Gamma \vdash M.P : \mathsf{Pro}[E, {}^{\bullet}F]}$$

(PREFIX $\circ$) and (PREFIX $\vartriangle$) state that prefixing a process P with a move capability always yields "moving" types, that is types with mode $\circ$. In particular, (PREFIX $\circ$) says that we may disregard the type of M (as long as M is a capability) if P is also a moving process.[8] This rule has the same rationale as the (POLYPATH) rule above: both rules are necessary for subject congruence — specifically, for the congruence rule $(M_1.M_2).P \equiv M_1.(M_2.P)$. On the other hand, by (PREFIX $\vartriangle$), the upward exchanges of M and P must be consistent (equal) when P is not moving. In other

[7] The reader may wonder why we introduced a new turnstile symbol rather then adding a mode to capabilities types, as in Cap°. In fact, the two choices are almost equivalent, in terms of expressive power, while the current is slightly less verbose.
[8] This characterization is possible because our syntax does not include the empty path.

words, the *last* move of the prefix must be compatible with the upward exchanges that the process will have right after. Notice, to this regard, that by subsumption, (PREFIX $\vartriangle$) also accounts for the case of prefixing a process P of type $\mathsf{Pro}[E,F]$.

The rule (PREFIX $\bullet$) types silent processes running in a context whose upward exchanges (if any) have type F. In this case, the type of the path M in the premise guarantees that P is type compatible with the local exchanges of the ambients hit on the move. Hence the typing of the capability must be "standard", as in the (PREFIX) rule from § 4.

The next two rules apply to parallel compositions.

(PARALLEL $*$ LEFT)

$$\frac{\Gamma \vdash P : \mathsf{Pro}[E, {}^{*}W] \quad \Gamma \vdash Q : \mathsf{Pro}[E, {}^{\bullet}W]}{\Gamma \vdash P \mid Q : \mathsf{Pro}[E, {}^{*}W]}$$

(PARALLEL $*$ RIGHT)

$$\frac{\Gamma \vdash P : \mathsf{Pro}[E, {}^{\bullet}W] \quad \Gamma \vdash Q : \mathsf{Pro}[E, {}^{*}W]}{\Gamma \vdash P \mid Q : \mathsf{Pro}[E, {}^{*}W]}$$

Two rules, and an appeal to subsumption, suffice to capture all cases. If P and Q are upward-silent (i.e. with upward exchanges ${}^{\bullet}W$), then $P \mid Q$ is also upward silent (with upward exchanges ${}^{\bullet}W$). $P \mid Q$ can be typed as moving (that is, with upward exchanges ${}^{\circ}W$), only when (i) either P or Q is moving and (ii) the other process is upward silent and type compatible with the exchanges of the moving process. The same reasoning applies when $P \mid Q : \mathsf{Pro}[E, {}^{\vartriangle}W]$, i.e. when $P \mid Q$ perform some upward exchange and then eventually move, hence the types $\mathsf{Pro}[E, {}^{\vartriangle}W]$ are derived with the same rules. We need two rules because we have to handle the two cases when the moving subprocess is P or Q.

The rules (DEAD) and (NEW) from § 4 handle also the cases for moded types (of course, save the fact that now T ranges over the extended class of process types). This is not true of the rule (REPL). In fact, if P and Q are both moving, then $P \mid Q$ may not be typed as moving, as either of the two could start its upward exchanges before the other. For this reason, there is no way to type a replicated process as a moving process: the only two possible types for a replicated process are a "regular" type (deduced by the rule REPL from § 4) or a silent type, as stated by the following new rule [9]:

(REPL $\bullet$)

$$\frac{\Gamma \vdash P : \mathsf{Pro}[E, {}^{\bullet}F]}{\Gamma \vdash !P : \mathsf{Pro}[E, {}^{\bullet}F]}$$

For processes of the form $M[\![P]\!]$, we need new rules. The rule (AMB) from § 4 is modified so that it now deduces an upward-silent type, compatible with all the other modes. Two new rules handle the case when M is a transport ambient, distinguishing the cases when the enclosed process is moving or not.

(AMB)

$$\frac{\Gamma \vdash M : \mathsf{Amb}[E,F] \quad \Gamma \vdash P : \mathsf{Pro}[E,F]}{\Gamma \vdash M[\![P]\!] : \mathsf{Pro}[F, {}^{\bullet}H]}$$

(AMB $\vartriangle$)

$$\frac{\Gamma \vdash M : \mathsf{Amb}^{\circ}[E,F] \quad \Gamma \vdash P : \mathsf{Pro}[E, {}^{\vartriangle}F]}{\Gamma \vdash M[\![P]\!] : \mathsf{Pro}[F, {}^{\bullet}H]}$$

(AMB $\circ$)

$$\frac{\Gamma \vdash M : \mathsf{Amb}^{\circ}[E,F] \quad \Gamma \vdash P : \mathsf{Pro}[E, {}^{\circ}F]}{\Gamma \vdash M[\![P]\!] : \mathsf{Pro}[G, {}^{\bullet}H]}$$

[9] This is due to the particular semantics of replication we use, which yields to an unrestrained generation of copies. It is clear that the use of guarded replication or call by need would make replication compatible with moded types (see also next section).

In (AMB $\vartriangle$) P is not moving, and the rule imposes type constraints equivalent to those imposed by the (AMB) rule: note, in fact, that the judgment $\Gamma \vdash P : \mathsf{Pro}[E, {}^{\vartriangle}F]$ could be derived by subsumption from $\Gamma \vdash P : \mathsf{Pro}[E, F]$. If, instead, P is moving, as in (AMB $\circ$), its upward exchanges are blocked by the move, and we have freedom to chose the type of local exchanges of the process $M[\![P]\!]$. Once again, subject reduction does not break if exercising the capability in P activates upward exchanges: (AMB $\vartriangle$) can be used to type the reductum .

We conclude with the rules for input-output.

(INPUT $\star *$)
$$\frac{\Gamma, x{:}W \vdash P : \mathsf{Pro}[W, {}^{*}F]}{\Gamma \vdash (x{:}W)P : \mathsf{Pro}[W, {}^{*}F]}$$

(OUTPUT $\star *$)
$$\frac{\Gamma \vdash M : W \quad \Gamma \vdash P : \mathsf{Pro}[W, {}^{*}F]}{\Gamma \vdash \langle M \rangle P : \mathsf{Pro}[W, {}^{*}F]}$$

(INPUT $\uparrow \vartriangle$)
$$\frac{\Gamma, x{:}W \vdash P : \mathsf{Pro}[F, {}^{\vartriangle}W]}{\Gamma \vdash (x{:}W)^{\uparrow}P : \mathsf{Pro}[F, {}^{\vartriangle}W]}$$

(OUTPUT $\uparrow \vartriangle$)
$$\frac{\Gamma \vdash M : W \quad \Gamma \vdash P : \mathsf{Pro}[F, {}^{\vartriangle}W]}{\Gamma \vdash \langle M \rangle^{\uparrow} P : \mathsf{Pro}[F, {}^{\vartriangle}W]}$$

Local communications are not affected by modes: it is the mode of the continuation process that determines the moded type of the input/output process itself.

Upward exchanges have only non-moving types, for obvious reasons. The particular type they have —that is, either $\mathsf{Pro}[F, {}^{\vartriangle}W]$ or the more informative $\mathsf{Pro}[F, W]$— depends on the type of their continuation. If their continuation is of type $\mathsf{Pro}[F, {}^{\bullet}W]$ or $\mathsf{Pro}[F, W]$, then the process —which is clearly not silent— can be typed as $\mathsf{Pro}[F, W]$. These cases are captured by the rule (INPUT/OUTPUT $\uparrow$) of § 4 (together with subsumption for the case $\mathsf{Pro}[F, {}^{\bullet}W]$). If instead the continuation has type $\mathsf{Pro}[F, {}^{\vartriangle}W]$ or $\mathsf{Pro}[F, {}^{\circ}W]$, as in (INPUT/OUTPUT $\uparrow \vartriangle$), we can just say that the process may eventually evolve into a moving process, hence the type $\mathsf{Pro}[F, {}^{\vartriangle}W]$ in the conclusion.

Finally, downward communications are not affected by whether the target ambient is moded or not. The rules from § 4 work just as well for the new system: two new rules, with the same format, handle the case when target ambient is moded:

(INPUT $M \circ$)
$$\frac{\Gamma \vdash M : \mathsf{Amb}^{\circ}[W, E] \quad \Gamma, x{:}W \vdash P : T}{\Gamma \vdash (x{:}W)^{M} P : T}$$

(OUTPUT $M \circ$)
$$\frac{\Gamma \vdash M : \mathsf{Amb}^{\circ}[W, E] \quad \Gamma \vdash N : W \quad \Gamma \vdash P : T}{\Gamma \vdash \langle N \rangle^{M} P : T}$$

Note that in all output rules, the typing of the expression M being output is subject to "regular" typing. As a consequence, capability paths may be communicated only if well-typed under regular typing. This restriction could be lifted, had we employed moded capability types as suggested in § 6.3 (cf. footnote 7), but with no significant additional expressive power.

6.4 Subject Reduction

The results of § 4 hold for the new system as well. As a matter of fact, subject reduction for the type system of § 4 is a direct consequence of the subject reduction for moded typing. The theorem, and its proof are standard.

Lemma 1 (Substitution). *Let $\vdash^{?}$ denote either $\vdash$ or $\vdash\!\circ$.*

– *Assume $\Gamma, x : W \vdash^{?} M : W'$ and $\Gamma \vdash N : W$. Then $\Gamma \vdash^{?} M\{x := N\} : W'$.*
– *Assume $\Gamma, x{:}W \vdash P : T$ and $\Gamma \vdash N : W$. Then $\Gamma \vdash P\{x := N\} : T$.*

Proof. *Standard: by induction on the derivations of the two judgments $\Gamma, x : W \vdash {}^{?} M : W'$ and $\Gamma \vdash P : T$.*

Lemma 2 (Subject Congruence). *If $\Gamma \vdash P : T$ and $P \equiv Q$ then $\Gamma \vdash Q : T$.*

Proof. *By simultaneous induction on the derivations of $P \equiv Q$ and $Q \equiv P$.*

Theorem 2 (Subject Reduction). *If $\Gamma \vdash P : T$ and $P \rightarrow Q$ then $\Gamma \vdash Q : T$.*

Proof. *By induction on the derivation of $P \rightarrow Q$.*

We conclude this section with an example showing how moded typing help type-check the transport ambients used in § 3.2 to encode communication on named channels à la Seal Calculus. We give the case of downward input on a channel of type W, as in $c^m(x : W)P$, as representative.

In the typed encoding, the channel c is expressed by the Boxed Ambient $c[\![\, !(x{:}W)\langle x\rangle \,]\!]$. Now, the typed encoding of downward input is as follows:

$$\langle\!\langle\, c^m(x{:}W)P \,\rangle\!\rangle = (\boldsymbol{\nu} p{:}\mathsf{Amb}^{\circ}[W, W])p[\![\, \mathsf{in}\ m.\mathsf{in}\ c.(x{:}W)^{\uparrow}\mathsf{out}\ c.\mathsf{out}\ m.\langle x\rangle \,]\!] \mid (x{:}W)^p \langle\!\langle\, P \,\rangle\!\rangle$$

We give a type derivation under the most general assumptions, that is: $P{:}\mathsf{Pro}[E, {}^{?}F], m{:}\mathsf{Amb}^{?}[G, H]$ (where E, F, G, H can be any type), and $c{:}\mathsf{Amb}[W, \mathsf{shh}]$. The fact that the ambient p is typed as a transport ambient is essential for the typed encoding to type-check. This is shown by the following analysis that also illustrate the interplay between the modes $\circ$ and $\vartriangle$.

Let Γ be a type environment where $m{:}\mathsf{Amb}^{?}[G, H], c{:}\mathsf{Amb}[W, \mathsf{shh}]$ and $p{:}\mathsf{Amb}^{\circ}[W, W]$. First observe that, for the encoding to be typable we need

$$\Gamma \vdash p[\![\, \mathsf{in}\ m.\mathsf{in}\ c.(x{:}W)^{\uparrow}\mathsf{out}\ c.\mathsf{out}\ m.\langle x\rangle \,]\!] : \mathsf{Pro}[E, {}^{\bullet}F]$$

This judgment may be derived by the rule (AMB $\circ$), provided that the process enclosed in p can be typed with mode $\circ$, that is, if

$$\Gamma \vdash \mathsf{in}\ m.\mathsf{in}\ c.(x{:}W)^{\uparrow}\mathsf{out}\ c.\mathsf{out}\ m.\langle x\rangle : \mathsf{Pro}[W, {}^{\circ}W].$$

This follows by (PREFIX $\circ$) from $\Gamma \vdash^{\circ} \mathsf{in}\ m{:}\mathsf{Cap}[G]$ and

$$\Gamma \vdash \mathsf{in}\ c.(x{:}W)^{\uparrow}\mathsf{out}\ c.\mathsf{out}\ m.\langle x\rangle : \mathsf{Pro}[W, {}^{\circ}W]$$

Note that here we use the flexibility of moded typing as no relation is required between G and W. The last judgment follows again by (PREFIX $\circ$) from $\Gamma \vdash^{\circ} \mathsf{in}\ c{:}\mathsf{Cap}[W]$ and from

$$\Gamma \vdash (x{:}W)^{\uparrow}\mathsf{out}\ c.\mathsf{out}\ m.\langle x\rangle : \mathsf{Pro}[W, {}^{\vartriangle}W],$$

This judgment can be derived by (INPUT $\uparrow \vartriangle$) from

$$\Gamma, x{:}W \vdash \mathsf{out}\ c.\mathsf{out}\ m.\langle x\rangle{:}\mathsf{Pro}[W, {}^{\vartriangle}W].$$

Again, we rely on moded typing: the whole process type-checks since the move that precedes the upward output brings the ambient in an environment with the right exchange type. Deriving the last judgment is not difficult. From $\Gamma, x{:}W \vdash^{\circ} \mathsf{out}\ m{:}\mathsf{Cap}[H]$ and from $\Gamma, x{:}W \vdash \langle x\rangle{:}\mathsf{Pro}[W, {}^{\circ}W]$, we have $\Gamma, x{:}W \vdash \mathsf{out}\ m.\langle x\rangle : \mathsf{Pro}[W, {}^{\circ}W]$. Now, from the last judgment and from $\Gamma, x{:}W \vdash^{\circ} \mathsf{out}\ c{:}\mathsf{Cap}[\mathsf{shh}]$ an application of (PREFIX $\circ$) yields $\Gamma, x{:}W \vdash \mathsf{out}\ c.\mathsf{out}\ m.\langle x\rangle{:}\mathsf{Pro}[W, {}^{\circ}W]$ as desired. To conclude, we can apply subsumption, based on the subtyping $\mathsf{Pro}[W, {}^{\circ}W] \leqslant \mathsf{Pro}[W, {}^{\vartriangle}W]$, and then (INPUT $\uparrow \vartriangle$) to obtain the desired typing.

7 Asynchronous communications

As noted in [Car99], mobile and distributed computation can hardly rely on synchronous input-output as the only mechanism of communication. Also, experience with implementation of distributed calculi [BV02,FLA00] shows that the form of consensus required for synchronous communication is quite hard to implement in a distributed environment.

In § 2 we said that asynchronous communication can be recovered in our calculus in two possible ways: (i) either by coding it with synchronous output and null continuations, or (ii) by introducing the additional equivalence $\langle M\rangle^\eta P \equiv \langle M\rangle^\eta \mid P$. The first solution allows synchronous and asynchronous output to coexist. An asynchronous output-prefix $\langle M\rangle^\eta$ followed by a continuation P can be expressed in terms of synchronous output by the parallel composition $\langle M\rangle^\eta \mathbf{0} \mid P$. The second solution takes this idea to its extreme, and leads to a purely asynchronous calculus.

Neither alternative is entirely satisfactory. One problem with the first is that $\langle M\rangle^\eta P$ and $\langle M\rangle^\eta \mathbf{0} \mid P$ are only equivalent under the type system of § 4, not with moded types. In fact, for $\eta = \uparrow$, it is not difficult to find situations where $\langle M\rangle^\eta P$ is well-typed and $\langle M\rangle^\eta \mathbf{0} \mid P$ is not (with moded typing). An immediate consequence of this observation is that the congruence law $\langle M\rangle^\uparrow P \equiv \langle M\rangle^\uparrow \mid P$ is not preserved by moded typing, hence the second alternative is not sound for the system of § 6.

A further reason for being unsatisfied with the first solution is that the use of null continuations to code asynchronous output has the effect of essentially defeating moded typing. Moded typing is possible, and effective, only along a single thread, while the coding of asynchronous output introduces parallel compositions and leaves no residual following an output. Notice, however, that the problem is not a consequence of moded typing and asynchrony being inherently incompatible. To see that, observe that in $\langle M\rangle^\uparrow P$ the continuation P could be typed with a mode independently of whether the prefix denotes synchronous or asynchronous output. All that matters for P to receive a (sound) "moving" type is that $\langle M\rangle$ gets delivered to the parent ambient before unleashing P: once delivered, whether or not $\langle M\rangle$ also synchronizes with local input is irrelevant.

Based on this observation, a smoother integration of asynchronous output and moded typing may be achieved by re-stating the congruence law as a reduction rule, and making it location-aware so that the output is delivered to the appropriate ambient.

Different formulations of the asynchronous version of the calculus are possible. A first solution, given below, is to replace the reductions (*output n*) and (*output* $\uparrow$) of § 2 with the reductions (*asynch output n*) and (*asynch output* $\uparrow$) below, and to introduce the new reduction (*asynch output* $\star$):

(*asynch output* $\star$)	$\langle M\rangle P \rightarrow \langle M\rangle \mid P$
(*asynch output n*)	$\langle M\rangle^n P \mid n[\![\, Q \,]\!] \rightarrow P \mid n[\![\, \langle M\rangle \mid Q \,]\!]$
(*asynch output* $\uparrow$)	$n[\![\, \langle M\rangle^\uparrow P \mid Q \,]\!] \rightarrow \langle M\rangle \mid n[\![\, P \mid Q \,]\!]$

With these reductions, the problem with moded types is solved: an upward output followed by a move, as in $\langle N\rangle^\uparrow M.P$ may safely be typed with mode $\vartriangle$ (based on the mode $\circ$ for $M.P$) irrespective of whether the output synchronizes or not. More generally, we may prove that subject reduction holds for this form of asynchronous reduction and the moded type system presented in the previous section: no further modification is needed.

A second possibility, is to combine synchrony and asynchrony. Cardelli [Car00], advocates that local exchanges can be synchronous, while remote communication ought to be asynchronous. This is a sound choice for our calculus: in fact, the reduction (*asynch output* $\star$) for local exchanges may be dispensed with, as local asynchronous output may be coded by $\langle M\rangle\mathbf{0} \mid P$ without affecting moded typing. Although this is sound, it would introduce some form of asymmetry in the implementation since non-local read accesses on local synchronous output would be synchronous with this solution.

A third possibility arises from the observation that the new output rules described for the first solution, together with the reduction rules for input prefixes of § 2 derive the following new set of reductions for input:

$$
\begin{array}{ll}
(\textit{asynch input} \star) & (x)P \mid \langle M\rangle \rightarrow P\{x := M\} \\
(\textit{asynch input } n) & (x)^n P \mid n[\![\, \langle M\rangle \mid Q \,]\!] \rightarrow P\{x := M\} \mid n[\![\, Q \,]\!] \\
(\textit{asynch input} \uparrow) & \langle M\rangle \mid n[\![\, (x)^{\uparrow} P \mid Q \,]\!] \rightarrow n[\![\, P\{x := M\} \mid Q \,]\!]
\end{array}
$$

One could then take the *asynch input* rules as primitive, and use them instead of the corresponding rules of § 2. In other words the third solution consists in replacing all the reduction rules of Section 2 by the six *asynch*-rules defined in this section. Although this solution is very close to the first one (but more "inefficient" since it adds new intermediate reduction steps), the result is rather interesting, as it suggests a novel interpretation of the process form $\langle M\rangle$ as a *memory cell*. Indeed, one may view $\langle M\rangle\mathbf{0}$ and $\langle M\rangle$ as denoting two very distinct processes, the former being a local output with a null continuation, the latter being a memory cell (more precisely a one-place buffer)[10]. Taking this view, every communication becomes a two-step protocol and the reductions have new interpretations. To exemplify, (*asynch output* $\star$) describes how a writer process $\langle M\rangle P$ writes a memory cell $\langle M\rangle$ and then continues as P; (*asynch input* $\star$) shows a reader that makes a destructive access to a memory cell $\langle M\rangle$. The same reasoning applies to downward and upward exchanges. As a result, memory cells, that is the output form $\langle M\rangle$, take the role of the resources of the calculus, which are bound to their location.

Whatever solution we choose in this section, they are all compatible with the moded typing of § 6 and, *a fortiori*, with the type system of § 4.

7.1 Synchrony versus asynchrony: security trade-offs

The choice of synchronous versus asynchronous communication has other consequences on the calculus, specifically, in terms of the security guarantees that can be made for it.

On one side, it is well known that synchronous communication generates hard-to-detect information flows based on synchronization. Our definition of synchronous input-output of § 2 also has this problem. For example, in the system $a[\![\, Q \mid b[\![\, \langle M\rangle P \,]\!] \,]\!]$, the sub-ambient b, gets to know exactly when (and if) Q makes a downward read access to its contents. Therefore one bit of information flowed by a read access from the reader to the writer. This makes non-interference [GM82,FG97] quite hard to satisfy.

On the other hand, by asynchronous communication we effectively give up *mediation* (see § 1.2), that is, control over interaction between sibling ambients. With synchronous input-output no ambient can be "spoiled" with unexpected (and possibly unwanted) output by its enclosing or enclosed ambients. As an example, consider the system $a[\![\, (x{:}W)^b P \mid b[\![\, c[\![\, \langle M\rangle^{\uparrow} \mid Q \,]\!] \,]\!] \,]\!]$ which is typable in our system provided that $M{:}W$ and the b is declared of type $\mathsf{Amb}[W, F]$ for some F. With synchronous reductions there is no way for the upward output in c and the downward input in a to synchronize. Instead, in the asynchronous case, the initial configuration

[10] To make it more explicit, for this last solution we could have used a different syntax for a memory cell containing M, say $\lbrace\!\mid M \mid\!\rbrace$, so that for example the local reduction rules would be written as

$$
\begin{array}{ll}
(\textit{asynch output} \star) & \langle M\rangle P \rightarrow \lbrace\!\mid M \mid\!\rbrace \mid P \\
(\textit{asynch input} \star) & (x)P \mid \lbrace\!\mid M \mid\!\rbrace \rightarrow P\{x := M\}
\end{array}
$$

an asynchronous output produce a cell, and a process reads from a cell.

would evolve into $a[\![\, (x{:}W)^b P \mid b[\![\, \langle M\rangle \mid c[\![\, Q\,]\!]\,]\!]\,]\!]$; by a further reduction step the ambient a gets hold of the message $\langle M\rangle$ without any mediation of b.

Similarly, two siblings may establish a covert channel: $b[\![\, a[\![\, (x{:}W)^{\uparrow} P\,]\!] \mid c[\![\, \langle M\rangle^{\uparrow} Q\,]\!]\,]\!]$ reduces in two steps into $b[\![\, a[\![\, P\{x := M\}\,]\!] \mid c[\![\, Q\,]\!]\,]\!]$. These kind of covert channels are two examples of security breaches that cannot be prevented by the existing primitives of the calculus. A possible solution is to resort to further synchronization mechanisms, such as those offered by *portals* in the Seal calculus: this however, would essentially defeat asynchrony. A different, and more effective, way to avoid covert channels is by multilevel security, based on types, as we show in [BCC01].

8 Conclusion and related work

We have presented a variant of Mobile Ambients, based on a different choice of communication primitives. The new calculus complements the constructs for mobility of Mobile Ambients with what we believe to be more effective mechanisms for resource protection and access control. In addition, it provides for more flexible typing of communications, and new insight into the relation between synchrony and asynchrony.

As we mentioned, other alternatives for parent-child communication would be possible. One alternative, suggested by the anonymous referees could be based on the following reductions:

$$(x)^n P \mid n[\![\, \langle M\rangle^{\uparrow} Q \mid R\,]\!] \rightarrow P\{x := M\} \mid n[\![\, Q \mid R\,]\!]$$
$$\langle M\rangle^n P \mid n[\![\, (x)^{\uparrow} Q \mid R\,]\!] \rightarrow P \mid n[\![\, Q\{x := M\} \mid R\,]\!]$$

These reductions are similar in spirit to the corresponding reductions adopted in [CGZ01] for the Seal Calculus. We had considered this option for our Boxed Ambients, and initially dismissed it because it appeared to be enforcing an interpretation of channels as shared resources, thus undermining the notion of locality we wished to express. Looking at it retrospectively, it is now only fair to observe that the alternative reductions would still enable the view of an ambient as having two channels: a private channel which is only available for local exchanges, and an "upward channel" which the ambient offers to its enclosing context for read and write access.

In fact, a first analysis shows that there are trade-offs between our solution and the one given above. The latter has a number of security benefits, as it provides ambients with full control of the exchanges they may have with their children. Our solution, instead, enables communication protocols that would be difficult (if at all possible) to express with the above reductions. One example is the possibility for an ambient to "broadcast" a message to *any* entering ambient: $a[\![\, !\,\langle M\rangle\,]\!]$. Here, a could be thought of as an "information site" which any ambient can enter to get a copy of M (reading it from upwards, after having entered a). The same protocol could hardly be expressed with the reductions given above, as they requires an ambient to know the names of its children in order to communicate with them. Nevertheless, a more in-depth analysis of the trade-offs between the two solutions deserves to be made, and is part of our plans of future work.

Besides Mobile Ambients and Seals, whose relationships with Boxed Ambients have been discussed all along, the new calculus shares the same motivations, and is superficially similar to Sewell and Vitek's Boxed-π [SV00]. The technical development, however, is entirely different. We do not provide direct mechanisms for constructing *wrappers*, rather we propose a new construct for ambient interaction in the attempt to provide easier-to-monitor communications. Also, our form of communication is anonymous, and based on a notion of locality which is absent in the Boxed-π Calculus. This latter choice has important consequences in the formalization of classic security models as we discuss in [BCC01]. Finally Boxed-π does not consider mobility which is a fundamental component of this work.

Our type system is clearly also related to other typing systems developed for Mobile Ambients. In [CG99] types guarantees absence of type confusion for communications. The type systems of [CGG99] and [Zim00] provide control over ambients moves and opening. Furthermore, the introduction of *group* names [CGG00] and the possibility of creating fresh group names, give flexible ways to statically prevent unwanted propagation of names. The powerful type discipline for Safe Ambients, presented in [LS00], add a finer control over ambient interactions and remove all *grave interference*, i.e. all non-deterministic choice between logical incompatible interactions.

All those approaches are orthogonal to the particular communication primitives. We believe that similar typing disciplines as well as the use of group names and mobility types (without opening control, of course), can be adapted to Boxed Ambients to obtain similar strong results.

Last, but not least, in [HR01,HR00] Hennessy and Riley discuss resource protection in $D\pi$-calculus, a distributed variant of π-calculus, where processes are placed in movable locations. In spite of the fact that the design choices in the two calculi are different, and largely unrelated (different primitives, no location nesting, . . .) the ideas discussed in [HR01,HR00] were a constant source of inspiration for us.

Acknowledgments. Thanks to Santiago Pericas for his comments of on earlier draft of the paper. Comments by the anonymous referees helped improve the presentation: we gratefully acknowledge their effort.

References

[BC01] M. Bugliesi and G. Castagna. Secure safe ambients. In *Proc. of the 28th ACM Symposium on Principles of Programming Languages*, pages 222–235, London, 2001. ACM Press.

[BCC01] M. Bugliesi, G. Castagna, and S. Crafa. Reasoning about security in mobile ambients. In *CONCUR 2001 (12th. International Conference on Concurrency Theory)*, number 2154 in Lecture Notes in Computer Science, pages 102–120, Aahrus, Danemark, 2001. Springer.

[Bou92] G. Boudol. Asynchrony and the π-calculus. Research Report 1702, INRIA, http://www-sop.inria.fr/mimosa/personnel/Gerard.Boudol.html, 1992.

[BV02] C. Bryce and J. Vitek. The JavaSeal mobile agent kernel. *Autonomous Agents and Multi-Agent Systems*, 2002. To appear.

[Car99] L. Cardelli. *Abstractions for Mobile Computation*, volume 1603 of *Lecture Notes in Computer Science*, pages 51–94. Springer, 1999.

[Car00] L. Cardelli. Global computing. In *IST FET Global Computing Consultation Workshop*, 2000. Slides.

[CG98] L. Cardelli and A. Gordon. Mobile ambients. In *Proceedings of POPL '98*. ACM Press, 1998.

[CG99] L. Cardelli and A. Gordon. Types for mobile ambients. In *Proceedings of POPL '99*, pages 79–92. ACM Press, 1999.

[CGG99] L. Cardelli, G. Ghelli, and A. Gordon. Mobility types for mobile ambients. In *Proceedings of ICALP '99*, number 1644 in Lecture Notes in Computer Science, pages 230–239. Springer, 1999.

[CGG00] L. Cardelli, G. Ghelli, and A. D. Gordon. Ambient groups and mobility types. In *International Conference IFIP TCS*, number 1872 in Lecture Notes in Computer Science, pages 333–347. Springer, 2000.

[CGZ01] G. Castagna, G. Ghelli, and F. Zappa. Typing mobility in the Seal Calculus. In *CONCUR 2001 (12th. International Conference on Concurrency Theory)*, number 2154 in Lecture Notes in Computer Science, pages 82–101, Aahrus, Danemark, 2001. Springer.

[DCS00] M. Dezani-Ciancaglini and I. Salvo. Security types for safe mobile ambients. In *Proceedings of ASIAN '00*, pages 215–236. Springer, 2000.

[DLB00] P. Degano, F. Levi, and C. Bodei. Safe ambients: Control flow analysis and security. In *Proceedins of ASIAN '00*, volume 1961 of *LNCS*, pages 199–214. Springer, 2000.

[FG97] R. Focardi and R. Gorrieri. Non interference: Past, present and future. In *Proceedings of DARPA Workshop on Foundations for Secure Mobile Code*, pages 26–28, march 1997.
[FLA00] C. Fournet, J-J. Levy, and Shmitt. A. An asynchronous, distributed implementation of mobile ambients. In *International Conference IFIP TCS*, number 1872 in Lecture Notes in Computer Science. Springer, 2000.
[GM82] J.A. Goguen and J. Meseguer. Security policy and security models. In *Proceedings of Symposium on Secrecy and Privacy*, pages 11–20. IEEE Computer Society, april 1982.
[HR00] M. Hennessy and J. Riely. Information flow vs. resource access in the asynchronous π-calculus (extended abstract). In *Automata, Languages and Programming, 27th International Colloquium*, volume 1853 of *Lecture Notes in Computer Science*, pages 415–427. Springer, 2000.
[HR01] M. Hennessy and J. Riely. Resource access control in systems of mobile agents. *Information and Computation*, 2001. To appear.
[LS00] F. Levi and D. Sangiorgi. Controlling interference in Ambients. In *POPL '00*, pages 352–364. ACM Press, 2000.
[NN00] H. R. Nielson and F. Nielson. Shape analysis for mobile ambients. In *POPL '00*, pages 135–148. ACM Press, 2000.
[NNHJ99] F. Nielson, H. Riis Nielson, R. R. Hansen, and J. G. Jensen. Validating firewalls in mobile ambients. In *Proc. CONCUR '99*, number 1664 in LNCS, pages 463–477. Springer, 1999.
[SV00] P. Sewell and J. Vitek. Secure composition of untrusted code: Wrappers and causality types. In *13th IEEE Computer Security Foundations Workshop*, 2000.
[VC99] J. Vitek and G. Castagna. Seal: A framework for secure mobile computations. In *Internet Programming Languages*, number 1686 in Lecture Notes in Computer Science. Springer, 1999.
[Zim00] P. Zimmer. Subtyping and typing algorithms for mobile ambients. In *Proceedins of FoSSaCS '99*, volume 1784 of *LNCS*, pages 375–390. Springer, 2000.

A Moded Typing: the complete type system

Recall that, in order to have a more compact set of rules, we use ${}^{*}W$ to denote any of the exchanges ${}^{\vartriangle}W, {}^{\bullet}W, {}^{\circ}W$, and use ${}^{?}W$ to denote either ${}^{*}W$ or W. Similarly we use $\mathsf{Amb}^{?}[E,F]$ to denote either $\mathsf{Amb}[E,F]$ or $\mathsf{Amb}^{\circ}[E,F]$. The use of such shorthands make it possible to express different rules of § 4 and 6 as instances of a same rule. For this reason the rules here are slightly different from those in the main text.

Expressions

(PROJECTION)
$$\frac{\Gamma(M) = W}{\Gamma \vdash M : W}$$

(TUPLE)
$$\frac{\Gamma \vdash M_i : W_i \quad \forall i \in 1..k}{\Gamma \vdash (M_1, \ldots, M_k) : W_1 \times \cdots \times W_k}$$

(IN)
$$\frac{\Gamma \vdash M : \mathsf{Amb}^{?}[F, E] \quad F' \leqslant F}{\Gamma \vdash \mathsf{in}\ M : \mathsf{Cap}[F']}$$

(OUT)
$$\frac{\Gamma \vdash M : \mathsf{Amb}[E, F] \quad F' \leqslant F}{\Gamma \vdash \mathsf{out}\ M : \mathsf{Cap}[F']}$$

(OUT $\circ$)
$$\frac{\Gamma \vdash M : \mathsf{Amb}^{\circ}[E, F]}{\Gamma \vdash\!\circ\ \mathsf{out}\ M : \mathsf{Cap}[\mathsf{shh}]}$$

(CAP $\circ$)
$$\frac{\Gamma \vdash M : \mathsf{Cap}[E]}{\Gamma \vdash\!\circ\ M : \mathsf{Cap}[E]}$$

(PATH)
$$\frac{\Gamma \vdash M_1 : \mathsf{Cap}[F] \quad \Gamma \vdash M_2 : \mathsf{Cap}[F]}{\Gamma \vdash M_1.M_2 : \mathsf{Cap}[F]}$$

(POLYPATH)
$$\frac{\Gamma \vdash\!\circ\ M_1 : \mathsf{Cap}[E_1] \quad \Gamma \vdash\!\circ\ M_2 : \mathsf{Cap}[E_2]}{\Gamma \vdash\!\circ\ M_1.M_2 : \mathsf{Cap}[E_2]}$$

Processes

(PREFIX $\circ$)
$$\frac{\mathit{\Gamma} \mathrel{\vdash\!\!\circ} M : \mathsf{Cap}[G] \quad \mathit{\Gamma} \vdash P : \mathsf{Pro}[E, {}^{\circ}F]}{\mathit{\Gamma} \vdash M.P : \mathsf{Pro}[E, {}^{\circ}F]}$$

(PREFIX $\vartriangle$)
$$\frac{\mathit{\Gamma} \mathrel{\vdash\!\!\circ} M : \mathsf{Cap}[F] \quad \mathit{\Gamma} \vdash P : \mathsf{Pro}[E, {}^{\vartriangle}F]}{\mathit{\Gamma} \vdash M.P : \mathsf{Pro}[E, {}^{\circ}F]}$$

(PREFIX $\bullet$)
$$\frac{\mathit{\Gamma} \vdash M : \mathsf{Cap}[F] \quad \mathit{\Gamma} \vdash P : \mathsf{Pro}[E, {}^{\bullet}F]}{\mathit{\Gamma} \vdash M.P : \mathsf{Pro}[E, {}^{\bullet}F]}$$

(PREFIX)
$$\frac{\mathit{\Gamma} \vdash M : \mathsf{Cap}[F] \quad \mathit{\Gamma} \vdash P : \mathsf{Pro}[E, F]}{\mathit{\Gamma} \vdash M.P : \mathsf{Pro}[E, F]}$$

(PAR)
$$\frac{\mathit{\Gamma} \vdash P : \mathsf{Pro}[E, F] \quad \mathit{\Gamma} \vdash Q : \mathsf{Pro}[E, F]}{\mathit{\Gamma} \vdash P \mid Q : \mathsf{Pro}[E, F]}$$

(PAR $*$)
$$\frac{\mathit{\Gamma} \vdash P : \mathsf{Pro}[E, {}^{*}W] \quad \mathit{\Gamma} \vdash Q : \mathsf{Pro}[E, {}^{\bullet}W]}{\mathit{\Gamma} \vdash P \mid Q,\ Q \mid P : \mathsf{Pro}[E, {}^{*}W]}$$

(DEAD)
$$\frac{}{\mathit{\Gamma} \vdash \mathbf{0} : T}$$

(NEW)
$$\frac{\mathit{\Gamma}, x : W \vdash P : T}{\mathit{\Gamma} \vdash (\boldsymbol{\nu} x : W)P : T}$$

(REPL)
$$\frac{\mathit{\Gamma} \vdash P : \mathsf{Pro}[E, F]}{\mathit{\Gamma} \vdash !P : \mathsf{Pro}[E, F]}$$

(REPL $\bullet$)
$$\frac{\mathit{\Gamma} \vdash P : \mathsf{Pro}[E, {}^{\bullet}F]}{\mathit{\Gamma} \vdash !P : \mathsf{Pro}[E, {}^{\bullet}F]}$$

(AMB)
$$\frac{\mathit{\Gamma} \vdash M : \mathsf{Amb}[E, F] \quad \mathit{\Gamma} \vdash P : \mathsf{Pro}[E, F]}{\mathit{\Gamma} \vdash M[\![P]\!] : \mathsf{Pro}[F, {}^{\bullet}H]}$$

(AMB $\vartriangle$)
$$\frac{\mathit{\Gamma} \vdash M : \mathsf{Amb}^{\circ}[E, F] \quad \mathit{\Gamma} \vdash P : \mathsf{Pro}[E, {}^{\vartriangle}F]}{\mathit{\Gamma} \vdash M[\![P]\!] : \mathsf{Pro}[F, {}^{\bullet}H]}$$

(AMB $\circ$)
$$\frac{\mathit{\Gamma} \vdash M : \mathsf{Amb}^{\circ}[E, F] \quad \mathit{\Gamma} \vdash P : \mathsf{Pro}[E, {}^{\circ}F]}{\mathit{\Gamma} \vdash M[\![P]\!] : \mathsf{Pro}[G, {}^{\bullet}H]}$$

(INPUT $\star$)
$$\frac{\mathit{\Gamma}, x : W \vdash P : \mathsf{Pro}[W, {}^{?}E]}{\mathit{\Gamma} \vdash (x : W)P : \mathsf{Pro}[W, {}^{?}E]}$$

(OUTPUT $\star$)
$$\frac{\mathit{\Gamma} \vdash M : W \quad \mathit{\Gamma} \vdash P : \mathsf{Pro}[W, {}^{?}E]}{\mathit{\Gamma} \vdash \langle M \rangle P : \mathsf{Pro}[W, {}^{?}E]}$$

(INPUT M)
$$\frac{\mathit{\Gamma} \vdash M : \mathsf{Amb}^{?}[W, E] \quad \mathit{\Gamma}, x : W \vdash P : T}{\mathit{\Gamma} \vdash (x : W)^{M} P : T}$$

(OUTPUT N)
$$\frac{\mathit{\Gamma} \vdash N : \mathsf{Amb}^{?}[W, E]\ \mathit{\Gamma} \vdash M : W\ \mathit{\Gamma} \vdash P : T}{\mathit{\Gamma} \vdash \langle M \rangle^{N} P : T}$$

(INPUT $\uparrow$)
$$\frac{\mathit{\Gamma}, x : W \vdash P : \mathsf{Pro}[E, W]}{\mathit{\Gamma} \vdash (x : W)^{\uparrow} P : \mathsf{Pro}[E, W]}$$

(OUTPUT $\uparrow$)
$$\frac{\mathit{\Gamma} \vdash M : W \quad \mathit{\Gamma} \vdash P : \mathsf{Pro}[E, W]}{\mathit{\Gamma} \vdash \langle M \rangle^{\uparrow} P : \mathsf{Pro}[E, W]}$$

(INPUT $\uparrow \vartriangle$)
$$\frac{\mathit{\Gamma}, x : W \vdash P : \mathsf{Pro}[F, {}^{\vartriangle}W]}{\mathit{\Gamma} \vdash (x : W)^{\uparrow} P : \mathsf{Pro}[F, {}^{\vartriangle}W]}$$

(OUTPUT $\uparrow \vartriangle$)
$$\frac{\mathit{\Gamma} \vdash M : W \quad \mathit{\Gamma} \vdash P : \mathsf{Pro}[F, {}^{\vartriangle}W]}{\mathit{\Gamma} \vdash \langle M \rangle^{\uparrow} P : \mathsf{Pro}[F, {}^{\vartriangle}W]}$$

In addition, we have a standard subsumption rule stating that $\mathit{\Gamma} \vdash P : T'$ whenever $\mathit{\Gamma} \vdash P : T$ and $T \leqslant T'$.