Ambient Groups and Mobility Types

Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon

Abstract. We add name groups and group creation to the typed ambient calculus. Group creation is surprisingly interesting: it has the effect of statically preventing certain communications, and can thus block the accidental or malicious escape of capabilities that is a major concern in practical systems. Moreover, ambient groups allow us to refine our earlier work on type systems for ambient mobility. We present type systems in which groups identify the set of ambients that a process may cross or open.

1 Introduction

The Ambient Calculus is a process calculus based on local communication and on process mobility. The basic, untyped, calculus can be decorated with static information to restrict either local communication, or mobility, or both.

Exchange control systems can be used to restrict communication. In [CG99] we have investigated *exchange types*, which subsume standard type systems for processes and functions, but do not impose restrictions on mobility.

Mobility control systems can be used to restrict mobility. In [CGG99] we investigate *immobility* and *locking* annotations, which are simple predicates about mobility.

The goal of this paper is to refine our previous work on mobility control, by including in the type of a process static descriptions of the set of ambients it may cross, and the set of ambients it may open. To do so, we adopt a new construction of independent interest. Among the types, we introduce collections of names that we call *groups*; names belong to groups in the same sense that values belong to types.

To understand how name groups arise, consider a typical static property we may want to express in a type system for the ambient calculus, informally:

> The ambient named n can enter the ambient named m.

This could be expressed as a typing $n : \mathit{CanEnter}(m)$ stating that n is a member of the collection $\mathit{CanEnter}(m)$ of names that can enter m. However, this would bring us straight into the domain of dependent types, since the type $\mathit{CanEnter}(m)$ depends on the name m. Instead, we introduce type-level groups of names, G, H, and restate our property as:

> The name n belongs to group G; the name m belongs to group H. Any ambient of group G can enter any ambient of group H.

Cardelli and Gordon are at Microsoft Research. Ghelli is at Pisa University.

This idea leads to typing judgments of the form:

process P may cross ambients of group G
process P may open ambients of group G

The former reduces to immobility assertions when a process can cross no groups; the latter reduces to locking assertions, when members of a group can be opened by no process [CGG99].

Among the processes, we then introduce an operation, $(\nu G)P$, for creating new groups. Within P we can introduce new names of group G. The binders for new groups, (νG), extrude in much the same way as binders for new names, $(\nu n{:}G)$. Because of extrusion, group binders do not impede the mobility of ambients that are enclosed in the initial scope of fresh groups. However, simple scoping restrictions prevent names of a fresh group from ever being received outside the initial scope of the group.

Therefore, we obtain a flexible way of protecting the propagation of names. This is to be contrasted with the situation in the untyped π-calculus and ambient calculus, where names can (intentionally, accidentally, or maliciously) be extruded arbitrarily far, by the automatic and unrestricted application of extrusion rules.

We organise the paper as follows. In the remainder of this opening section we review the basic untyped ambient calculus. Section 2 describes the typed ambient calculus with groups—obtained by enriching our exchange type system [CG99] with groups. Section 3 enriches the system of Section 2 to control ambient opening. In Section 4, we define a system in which the type of a process records both the groups it may open and the groups it may cross. Section 5 formalizes safety properties guaranteed by typing. Section 6 concludes and discusses related work.

1.1 The Untyped Ambient Calculus (Review)

An ambient is a named boundary whose interior contains a collection of running processes, possibly including nested subambients. We explain the untyped ambient calculus elsewhere [CG98] in detail, but here we introduce its central features via a standard example: $a[p[\mathit{out}\ a.\mathit{in}\ b.\langle c\rangle]] \mid b[\mathit{open}\ p.(x).x[]]$.

Intuitively, this example represents a packet named p being sent from a machine a to a machine b. The example consists of the parallel composition (indicated by the $\mid$ operator) of two ambients, named a and b. The brackets $[\ldots]$ represent ambients' boundaries. The process $p[\mathit{out}\ a.\mathit{in}\ b.\langle c\rangle]$ represents the packet, a subambient of ambient a. The name of the packet ambient is p, and its interior is the process $\mathit{out}\ a.\mathit{in}\ b.\langle c\rangle$. This process consists of three sequential actions: exercise the capability $\mathit{out}\, a$, exercise the capability $\mathit{in} b$, and then output the name c. The effect of the two capabilities on the enclosing ambient p is to move p out of a and into b, to reach the state: $a[] \mid b[p[\langle c\rangle] \mid \mathit{open}\ p.(x).x[]]$. The interior of a is now empty. The interior of b consists of two running processes, the subambient $p[\langle c\rangle]$ and the process $\mathit{open}\ p.(x).x[]$. The latter is attempting to exercise the $\mathit{open}\ p$ capability. Previously it was blocked. Now that the p ambient is present,

the effect of *open* p is to dissolve the ambient's boundary. Hence, the interior of b becomes the process $\langle c\rangle \mid (x).x[]$. This is a composition of an output $\langle c\rangle$ with an input $(x).x[]$. The input consumes the output, leaving $c[]$ as the interior of b. Hence, the final state of the whole example is $a[] \mid b[c[]]$.

The $\mathbf{0}$ process represents inactivity; the notation $a[]$ for an empty ambient named a, used above, is actually short for $a[\mathbf{0}]$. There are also replication and restriction constructs. A replication $!P$ behaves the same as an unlimited number of parallel copies of P. A restriction $(\nu n)P$ creates a new name n with scope P.

2 The Typed Ambient Calculus with Groups

We start with the typed ambient calculus of [CG99] and we add a new process construct, $(\nu G)P$, to create a new group G with scope P. Correspondingly we add a new type construct, $G[T]$, for the type of names of group G that name ambients that contain T exchanges.

The construct $G[T]$ is actually a refinement of the construct $Amb[T]$ of [CG99], where Amb can now be seen as the group of all names. It is conceivable to introduce a subtype ordering on groups, with Amb as the maximal element. However, subtyping may help capabilities escape, particularly in the presence of a maximal element; we do not consider these extensions in this paper.

We can now write, for example, the following typed process:

$$(\nu Ch)(\nu Msg)(\nu c{:}Ch[Msg[Shh]])(\nu m{:}Msg[Shh])c[\langle m\rangle \mid (x{:}Msg[Shh]).x[]]$$

This creates two groups Ch and Msg and two names c and m belonging to those groups. The types ensure that only messages, that is, names of type $Msg[Shh]$, can be exchanged inside an ambient named c, as happens in the rest of the process. (The type Shh prohibits exchanges; names of type $Msg[Shh]$ are in group Msg, and name ambients in which exchanges are prohibited.)

The types of the ambient calculus with groups are the same as in [CG99], except that $G[T]$ replaces $Amb[T]$. We have types W for messages. Messages can be either names of type $G[T]$, or capabilities of type $Cap[T]$. We also have types for processes, T, that classify processes according to the type of message tuples they exchange (if any).

Types:

$W ::=$	message type
$\quad G[T]$	$\quad$ ambient name in group G with T exchanges
$\quad Cap[T]$	$\quad$ capability unleashing T exchanges
$S, T ::=$	exchange type
$\quad Shh$	$\quad$ no exchange
$\quad W_1 \times \cdots \times W_k$	$\quad$ tuple exchange ($\mathbf{1}$ is the null product)

Expressions (messages) and processes are also the same as in [CG99], except that we add processes $(\nu G)P$ and include the objective moves of [CGG99].

The movement primitives of the untyped calculus, illustrated by the process $p[out\,a.in\,b.\langle c\rangle]$ from Section 1.1, are called *subjective moves*; the capabilities $out\,a$ and $in\,b$ move the ambient p from the inside. In the typed calculus, we also take *objective moves* as primitive. In an objective move $go\,N.M[P]$, the capability N moves an ambient $M[P]$ from the outside by following the path encoded by N, and once there starts the ambient $M[P]$. In the untyped calculus, we can define an objective move $go\,N.M[P]$ to be short for the process $(\nu k)k[N.M[out\,k.P]]$ where k is not free in P. As we found in our previous work [CGG99], a primitive typing rule for objective moves allows more refined typings than are possible with only subjective moves.

Expressions and processes:

$M, N ::=$	expression	$P, Q, R ::=$	process
n	name	$(\nu G)P$	group creation
$in\,M$	can enter M	$(\nu n{:}W)P$	restriction
$out\,M$	can exit M	$\mathbf{0}$	inactivity
$open\,M$	can open M	$P \mid Q$	composition
ϵ	null	$!P$	replication
$M.M'$	path	$M[P]$	ambient
		$M.P$	action
		$(x_1{:}W_1, \ldots, x_k{:}W_k).P$	input
		$\langle M_1, \ldots, M_k\rangle$	output
		$go\,N.M[P]$	objective move

This grammar allows the formation of certain nonsensical processes, where a capbility is used in place of a name, as in $(in\,n)[\mathbf{0}]$, or vice versa, as in $(\nu n{:}W)n.\mathbf{0}$. Such garbled processes are not typable in any of our type systems.

In the processes $(\nu G)P$ and $(\nu n{:}W)P$, the group G and the name n, respectively, are bound, with scope P. In the process $(x_1{:}W_1, \ldots, x_k{:}W_k).P$, the names $x_1, \ldots, x_k$ are bound, with scope P. We identify processes up to consistent renaming of bound names and bound groups. We write $fn(P)$ for the set of names free in process P, and we write $fg(P)$, $fg(W)$, and $fg(T)$ for the sets of groups free in process P, message type W, and exchange type T, respectively.

The following tables describe the structural congruence rules and the reduction rules. The bottom four rules of structural congruence describe the extrusion behavior of the (νG) binders. Side conditions on these rules prevent violation of lexical scoping. The notation $P\{x_1 \leftarrow M_1, \ldots, x_k \leftarrow M_k\}$ used below in the reduction rule for I/O denotes the outcome of a capture-avoiding simultaneous substitution, for each $i \in 1..k$, of the expression M_i for each free occurrence of the corresponding name x_i in the process P.

Structural Congruence:

$P \equiv Q \Rightarrow (\nu n{:}W)P \equiv (\nu n{:}W)Q$	$P \equiv P$
$P \equiv Q \Rightarrow (\nu G)P \equiv (\nu G)Q$	$Q \equiv P \Rightarrow P \equiv Q$
$P \equiv Q \Rightarrow P \mid R \equiv Q \mid R$	$P \equiv Q, Q \equiv R \Rightarrow P \equiv R$

$P \equiv Q \Rightarrow !P \equiv !Q$
$P \equiv Q \Rightarrow M[P] \equiv M[Q]$ $\quad P \mid Q \equiv Q \mid P$
$P \equiv Q \Rightarrow M.P \equiv M.Q$ $\quad (P \mid Q) \mid R \equiv P \mid (Q \mid R)$
$P \equiv Q \Rightarrow go\ N.M[P] \equiv go\ N.M[Q]$
$P \equiv Q \Rightarrow (x_1{:}W_1, \ldots, x_k{:}W_k).P \equiv (x_1{:}W_1, \ldots, x_k{:}W_k).Q$

$n_1 \neq n_2 \Rightarrow (\nu n_1{:}W_1)(\nu n_2{:}W_2)P \equiv (\nu n_2{:}W_2)(\nu n_1{:}W_1)P$
$n \notin fn(P) \Rightarrow (\nu n{:}W)(P \mid Q) \equiv P \mid (\nu n{:}W)Q$
$n \neq m \Rightarrow (\nu n{:}W)m[P] \equiv m[(\nu n{:}W)P]$

$P \mid \mathbf{0} \equiv P$ $\quad !P \equiv P \mid !P$
$(\nu n{:}W)\mathbf{0} \equiv \mathbf{0}$ $\quad \epsilon.P \equiv P$
$(\nu G)\mathbf{0} \equiv \mathbf{0}$ $\quad (M.M').P \equiv M.M'.P$
$!\mathbf{0} \equiv \mathbf{0}$ $\quad go\ \epsilon.M[P] \equiv M[P]$

$(\nu G_1)(\nu G_2)P \equiv (\nu G_2)(\nu G_1)P$
$G \notin fg(W) \Rightarrow (\nu G)(\nu n{:}W)P \equiv (\nu n{:}W)(\nu G)P$
$G \notin fg(P) \Rightarrow (\nu G)(P \mid Q) \equiv P \mid (\nu G)Q$
$(\nu G)m[P] \equiv m[(\nu G)P]$

Reduction:

$n[in\ m.P \mid Q] \mid m[R] \to m[n[P \mid Q] \mid R]$ $\quad P \to Q \Rightarrow (\nu G)P \to (\nu G)Q$
$m[n[out\ m.P \mid Q] \mid R] \to n[P \mid Q] \mid m[R]$ $\quad P \to Q \Rightarrow (\nu n{:}W)P \to (\nu n{:}W)Q$
$open\ n.P \mid n[Q] \to P \mid Q$ $\quad P \to Q \Rightarrow P \mid R \to Q \mid R$
$\langle M_1, \ldots, M_k \rangle \mid (x_1{:}W_1, \ldots, x_k{:}W_k).P$ $\quad P \to Q \Rightarrow n[P] \to n[Q]$
$\quad \to P\{x_1 \leftarrow M_1, \ldots, x_k \leftarrow M_k\}$ $\quad P' \equiv P, P \to Q, Q \equiv Q' \Rightarrow P' \to Q'$
$go\ (in\ m.N).n[P] \mid m[Q] \to m[go\ N.n[P] \mid Q]$
$m[go\ (out\ m.N).n[P] \mid Q] \to go\ N.n[P] \mid m[Q]$

In the tables below, we introduce the five basic judgments and the typing rules. Apart from minor adaptations, the main novelty with respect to [CG99] is the rule with conclusion $E \vdash (\nu G)P : T$. The assumptions of this rule are that $E, G \vdash P : T$ and $G \notin fg(T)$. The latter assumption prevents G from going out of scope in the conclusion. Typing environments, E, are given by the grammar $E ::= \varnothing \mid E, G \mid E, n{:}W$. For each E, we inductively define $dom(E)$ by the equations $dom(\varnothing) = \varnothing$, $dom(E, G) = dom(E) \cup \{G\}$, and $dom(E, n{:}W) = dom(E) \cup \{n\}$.

Judgments:

$E \vdash \diamond$	good environment
$E \vdash W$	good message type W
$E \vdash T$	good exchange type T
$E \vdash M : W$	good expression M of message type W
$E \vdash P : T$	good process P with exchange type T

Typing Rules:

$$\frac{}{\varnothing \vdash \diamond} \qquad \frac{E \vdash W \quad n \notin dom(E)}{E, n{:}W \vdash \diamond} \qquad \frac{E \vdash \diamond \quad G \notin dom(E)}{E, G \vdash \diamond}$$

$$\frac{G \in dom(E) \quad E \vdash T}{E \vdash G[T]} \qquad \frac{E \vdash T}{E \vdash Cap[T]} \qquad \frac{E \vdash \diamond}{E \vdash Shh} \qquad \frac{E \vdash W_1 \quad \cdots \quad E \vdash W_k}{E \vdash W_1 \times \cdots \times W_k}$$

$$\frac{E', n{:}W, E'' \vdash \diamond}{E', n{:}W, E'' \vdash n : W} \qquad \frac{E \vdash Cap[T]}{E \vdash \epsilon : Cap[T]} \qquad \frac{E \vdash M : Cap[T] \quad E \vdash M' : Cap[T]}{E \vdash M.M' : Cap[T]}$$

$$\frac{E \vdash n : G[S] \quad E \vdash T}{E \vdash in\ n : Cap[T]} \qquad \frac{E \vdash n : G[S] \quad E \vdash T}{E \vdash out\ n : Cap[T]} \qquad \frac{E \vdash n : G[T]}{E \vdash open\ n : Cap[T]}$$

$$\frac{E \vdash M : Cap[T] \quad E \vdash P : T}{E \vdash M.P : T} \qquad \frac{E \vdash M : G[S] \quad E \vdash P : S \quad E \vdash T}{E \vdash M[P] : T}$$

$$\frac{E, n{:}G[S] \vdash P : T}{E \vdash (\nu n{:}G[S])P : T} \qquad \frac{E, G \vdash P : T \quad G \notin fg(T)}{E \vdash (\nu G)P : T} \qquad \frac{E \vdash T}{E \vdash \mathbf{0} : T} \qquad \frac{E \vdash P : T}{E \vdash !P : T}$$

$$\frac{E \vdash P : T \quad E \vdash Q : T}{E \vdash P \mid Q : T} \qquad \frac{E, n_1{:}W_1, \ldots, n_k{:}W_k \vdash P : W_1 \times \cdots \times W_k}{E \vdash (n_1{:}W_1, \ldots, n_k{:}W_k).P : W_1 \times \cdots \times W_k}$$

$$\frac{E \vdash M_1 : W_1 \quad \cdots \quad E \vdash M_k : W_k}{E \vdash \langle M_1, \ldots, M_k \rangle : W_1 \times \cdots \times W_k}$$

$$\frac{E \vdash N : Cap[S'] \quad E \vdash M : G[S] \quad E \vdash P : S \quad E \vdash T}{E \vdash go\ N.M[P] : T}$$

We obtain a standard subject reduction result. A subtle point, though, is the need to account for the appearance of new groups (G_1, ..., G_k, below) during reduction. This is because reduction is defined up to structural congruence, and structural congruence does not preserve the set of free groups of a process. The culprit is the rule $(\nu n{:}W)\mathbf{0} \equiv \mathbf{0}$, in which groups free in W are not free in $\mathbf{0}$.

Theorem 1. *If $E \vdash P : T$ and either $P \equiv Q$ or $P \to Q$ then there are G_1, ..., G_k such that $G_1, \ldots, G_k, E \vdash Q : T$.*

3 Opening Control

In this section, to control usage of the *open* capability, we add attributes to the ambient types, $G[T]$, and the capability types, $Cap[T]$, of the previous type system. (In the next section, to control usage of the *in* and *out* capabilities, we add further attributes.)

To control the opening of ambients, we formalize the constraint that the name of any ambient opened by a process is in one of the groups G_1, ..., G_k,

but in no others. To do so, we add an attribute $^\circ\{G_1, \ldots, G_k\}$ to ambient types, which now take the form $G[^\circ\{G_1, \ldots, G_k\}, T]$. A name of this type is in group G, and names ambients within which processes may exchange messages of type T and may only open ambients in the groups $G_1, \ldots, G_k$. We need to add the same attribute to capability types, which now take the form $\mathit{Cap}[^\circ\{G_1, \ldots, G_k\}, T]$. Exercising a capability of this type may unleash exchanges of type T and openings of ambients in groups $G_1, \ldots, G_k$. The typing judgment for processes acquires the form $E \vdash P : {}^\circ\{G_1, \ldots, G_k\}, T$. The pair $^\circ\{G_1, \ldots, G_k\}, T$ constrains both the *opening effects* (what ambients the process opens) and the *exchange effects* (what messages the process exchanges). We call such a pair an *effect*, and introduce the metavariable F to range over effects. It is also convenient to introduce metavariables $\mathbf{G}$, $\mathbf{H}$ to range over finite sets of name groups. The following table summarizes these metavariable conventions and our enhanced syntax for types:

Group Sets and Types:

$\mathbf{G}, \mathbf{H} ::= \{G_1, \ldots, G_k\}$	finite set of name groups
$W ::=$	message type
$\quad G[F]$	$\quad$ ambient name in group G (contains processes with F effects)
$\quad \mathit{Cap}[F]$	$\quad$ capability (unleashes F effects)
$F ::=$	effect
$\quad {}^\circ\mathbf{H}, T$	$\quad$ may open $\mathbf{H}$, may exchange T
$S, T ::=$	exchange type
$\quad \mathit{Shh}$	$\quad$ no exchange
$\quad W_1 \times \cdots \times W_k$	$\quad$ tuple exchange

The following tables define the type system in detail. There are five basic judgments as before. They have the same format except that the judgment $E \vdash F$, meaning that the effect F is good given environment E, replaces the previous judgment $E \vdash T$. We omit the three rules for deriving good environments; they are exactly as in the previous section. There are two main differences between the other rules below and the rules of the previous section. First, effects, F, replace exchange types, T, throughout. Second, in the rule ascribing a type to *open* n, the condition $G \in \mathbf{H}$ constrains the opening effect $\mathbf{H}$ of the capability *open* n to include the group G, the group of the name n.

Judgments:

$E \vdash \diamond$	good environment
$E \vdash W$	good message type W
$E \vdash F$	good effect F
$E \vdash M : W$	good expression M of message type W
$E \vdash P : F$	good process P with F effects

Typing Rules:

$$\frac{G \in dom(E) \quad E \vdash F}{E \vdash G[F]} \qquad \frac{E \vdash F}{E \vdash Cap[F]} \qquad \frac{\mathbf{H} \subseteq dom(E) \quad E \vdash \diamond}{E \vdash {}^{\circ}\mathbf{H}, Shh}$$

$$\frac{\mathbf{H} \subseteq dom(E) \quad E \vdash W_1 \quad \cdots \quad E \vdash W_k}{E \vdash {}^{\circ}\mathbf{H}, W_1 \times \cdots \times W_k} \qquad \frac{E', n{:}W, E'' \vdash \diamond}{E', n{:}W, E'' \vdash n : W}$$

$$\frac{E \vdash Cap[F]}{E \vdash \epsilon : Cap[F]} \qquad \frac{E \vdash M : Cap[F] \quad E \vdash M' : Cap[F]}{E \vdash M.M' : Cap[F]}$$

$$\frac{E \vdash n : G[F] \quad E \vdash {}^{\circ}\mathbf{H}, T}{E \vdash in\ n : Cap[{}^{\circ}\mathbf{H}, T]} \qquad \frac{E \vdash n : G[F] \quad E \vdash {}^{\circ}\mathbf{H}, T}{E \vdash out\ n : Cap[{}^{\circ}\mathbf{H}, T]}$$

$$\frac{E \vdash n : G[{}^{\circ}\mathbf{H}, T] \quad G \in \mathbf{H}}{E \vdash open\ n : Cap[{}^{\circ}\mathbf{H}, T]} \qquad \frac{E \vdash M : Cap[F] \quad E \vdash P : F}{E \vdash M.P : F}$$

$$\frac{E \vdash M : G[F] \quad E \vdash P : F \quad E \vdash F'}{E \vdash M[P] : F'} \qquad \frac{E, n{:}G[F] \vdash P : F'}{E \vdash (\nu n{:}G[F])P : F'}$$

$$\frac{E, G \vdash P : F \quad G \notin fg(F)}{E \vdash (\nu G)P : F} \qquad \frac{E \vdash F}{E \vdash \mathbf{0} : F} \qquad \frac{E \vdash P : F}{E \vdash !P : F}$$

$$\frac{E \vdash P : F \quad E \vdash Q : F}{E \vdash P \mid Q : F} \qquad \frac{E, n_1{:}W_1, \ldots, n_k{:}W_k \vdash P : {}^{\circ}\mathbf{H}, W_1 \times \cdots \times W_k}{E \vdash (n_1{:}W_1, \ldots, n_k{:}W_k).P : {}^{\circ}\mathbf{H}, W_1 \times \cdots \times W_k}$$

$$\frac{E \vdash M_1 : W_1 \quad \cdots \quad E \vdash M_k : W_k \quad \mathbf{H} \subseteq dom(E)}{E \vdash \langle M_1, \ldots, M_k \rangle : {}^{\circ}\mathbf{H}, W_1 \times \cdots \times W_k}$$

$$\frac{E \vdash N : Cap[{}^{\circ}\mathbf{H}, T] \quad E \vdash M : G[F] \quad E \vdash P : F \quad E \vdash F'}{E \vdash go\ N.M[P] : F'}$$

Theorem 2. *If $E \vdash P : F$ and either $P \equiv Q$ or $P \to Q$ then there are $G_1, \ldots, G_k$ such that $G_1, \ldots, G_k, E \vdash Q : F$.*

Here is a simple example of a typing derivable in this system:

$$G, n{:}G[{}^{\circ}\{G\}, Shh] \vdash n[\mathbf{0}] \mid open\ n.\mathbf{0} : {}^{\circ}\{G\}, Shh$$

This asserts that the whole process $n[\mathbf{0}] \mid open\ n.\mathbf{0}$ is well-typed and opens only ambients in the group G.

On the other hand, one might expect the following variant to be derivable, but it is not:

$$G, n{:}G[{}^{\circ}\varnothing, Shh] \not\vdash n[\mathbf{0}] \mid open\ n.\mathbf{0} : {}^{\circ}\{G\}, Shh$$

This is because the typing rule for *open* n requires the effect unleashed by the *open* n capability to be the same as the effect contained within the ambient n.

But the opening effect $^\circ\varnothing$ specified by the type $G[^\circ\varnothing, \mathit{Shh}]$ of n cannot be the same as the effect unleashed by $\mathit{open}\, n$, because the rule also requires the latter to at least include the group G of n.

We have not found this feature to be problematic, and indeed it has a positive side-effect: the type $G[^\circ\mathbf{G}, T]$ of an ambient name n not only tells which opening effects may happen inside the ambient, but also tells whether n may be opened from outside: it is openable only if $G \in \mathbf{G}$, since this is the only case when $\mathit{open}\, n.\mathbf{0} \mid n[P]$ can be well typed. Hence, the presence of G in the set $\mathbf{G}$ may either mean that n is meant to be an ambient within which other ambients in group G may be opened, or that it is meant to be an openable ambient.

4 Crossing Control

This section presents the third and final type system of the paper, obtained by enriching the type system of Section 3 with attributes to control mobility.

Movement operators enable an ambient n to cross the boundary of another ambient m either by entering it via an *in* m capability or by exiting it via an *out* m capability. In the type system of this section, the type of n lists those groups that may be crossed; the ambient n may only cross the boundary of another ambient m if the group of m is included in this list. In our typed calculus, there are two kinds of movement, subjective moves and objective moves. Therefore, we separately list those groups that may be crossed by objective moves and those groups that may be crossed by subjective moves.

We add new attributes to the syntax of ambient types, effects, and capability types. An ambient type acquires the form $G\,{}^\frown\mathbf{G}'[{}^\frown\mathbf{G}, {}^\circ\mathbf{H}, T]$. An ambient of this type is in group G, may cross ambients in groups $\mathbf{G}'$ by objective moves, may cross ambients in groups $\mathbf{G}$ by subjective moves, may open ambients in groups $\mathbf{H}$, and may contain exchanges of type T. An effect, F, of a process is now of the form ${}^\frown\mathbf{G}, {}^\circ\mathbf{H}, T$. It asserts that the process may exercise *in* and *out* capabilities to accomplish subjective moves across ambients in groups $\mathbf{G}$, that the process may open ambients in groups $\mathbf{H}$, and that the process may exchange messages of type T. Finally, a capability type retains the form $\mathit{Cap}[F]$, but with the new interpretation of F. Exercising a capability of this type may unleash F effects.

Types:

$W ::=$	message type
$\quad G\,{}^\frown\mathbf{G}[F]$	ambient name in group G, crosses $\mathbf{G}$ objectively, contains processes with F effects
$\quad \mathit{Cap}[F]$	capability (unleashes F effects)
$F ::=$	effect
$\quad {}^\frown\mathbf{G}, {}^\circ\mathbf{H}, T$	crosses $\mathbf{G}$, opens $\mathbf{H}$, exchanges T
$S, T ::=$	exchange type
$\quad \mathit{Shh}$	no exchange
$\quad W_1 \times \cdots \times W_k$	tuple exchange

The format of the five judgments making up the system is the same as in Section 3. We omit the three rules defining good environments; they are as in Section 2. There are two main changes to the previous system to control mobility. First, the rules for typing *in* n and *out* n change to assign a type $Cap[\frown\mathbf{G},\mathrm{o}\mathbf{H},T]$ to the capabilities *in* n and *out* n only if $G \in \mathbf{G}$ where G is the group of n. Second, the rule for objective moves changes to allow an objective move of an ambient of type $G\frown\mathbf{G}'[F]$ by a capability of type $Cap[\frown\mathbf{G},\mathrm{o}\mathbf{H},T]$ only if $\mathbf{G} = \mathbf{G}'$.

Typing Rules:

$$\frac{G \in dom(E) \quad \mathbf{G} \subseteq dom(E) \quad E \vdash F}{E \vdash G\frown\mathbf{G}[F]} \qquad \frac{E \vdash F}{E \vdash Cap[F]}$$

$$\frac{\mathbf{G} \subseteq dom(E) \quad \mathbf{H} \subseteq dom(E) \quad E \vdash \diamond}{E \vdash \frown\mathbf{G},\mathrm{o}\mathbf{H}, Shh}$$

$$\frac{\mathbf{G} \subseteq dom(E) \quad \mathbf{H} \subseteq dom(E) \quad E \vdash W_1 \quad \cdots \quad E \vdash W_k}{E \vdash \frown\mathbf{G},\mathrm{o}\mathbf{H}, W_1 \times \cdots \times W_k}$$

$$\frac{E', n{:}W, E'' \vdash \diamond}{E', n{:}W, E'' \vdash n : W} \qquad \frac{E \vdash Cap[F]}{E \vdash \epsilon : Cap[F]} \qquad \frac{E \vdash M : Cap[F] \quad E \vdash M' : Cap[F]}{E \vdash M.M' : Cap[F]}$$

$$\frac{E \vdash n : G\frown\mathbf{G}'[F] \quad E \vdash \frown\mathbf{G},\mathrm{o}\mathbf{H}, T \quad G \in \mathbf{G}}{E \vdash in\ n : Cap[\frown\mathbf{G},\mathrm{o}\mathbf{H}, T]}$$

$$\frac{E \vdash n : G\frown\mathbf{G}'[F] \quad E \vdash \frown\mathbf{G},\mathrm{o}\mathbf{H}, T \quad G \in \mathbf{G}}{E \vdash out\ n : Cap[\frown\mathbf{G},\mathrm{o}\mathbf{H}, T]}$$

$$\frac{E \vdash n : G\frown\mathbf{G}'[\frown\mathbf{G},\mathrm{o}\mathbf{H}, T] \quad G \in \mathbf{H}}{E \vdash open\ n : Cap[\frown\mathbf{G},\mathrm{o}\mathbf{H}, T]} \qquad \frac{E \vdash M : Cap[F] \quad E \vdash P : F}{E \vdash M.P : F}$$

$$\frac{E \vdash M : G\frown\mathbf{G}[F] \quad E \vdash P : F \quad E \vdash F'}{E \vdash M[P] : F'} \qquad \frac{E, n{:}G\frown\mathbf{G}[F] \vdash P : F'}{E \vdash (\nu n{:}G\frown\mathbf{G}[F])P : F'}$$

$$\frac{E, G \vdash P : F \quad G \notin fg(F)}{E \vdash (\nu G)P : F} \qquad \frac{E \vdash F}{E \vdash \mathbf{0} : F} \qquad \frac{E \vdash P : F}{E \vdash !P : F}$$

$$\frac{E \vdash P : F \quad E \vdash Q : F}{E \vdash P \mid Q : F} \qquad \frac{E, n_1{:}W_1, \ldots, n_k{:}W_k \vdash P : \frown\mathbf{G},\mathrm{o}\mathbf{H}, W_1 \times \cdots \times W_k}{E \vdash (n_1{:}W_1, \ldots, n_k{:}W_k).P : \frown\mathbf{G},\mathrm{o}\mathbf{H}, W_1 \times \cdots \times W_k}$$

$$\frac{E \vdash M_1 : W_1 \quad \cdots \quad E \vdash M_k : W_k \quad \mathbf{G} \subseteq dom(E) \quad \mathbf{H} \subseteq dom(E)}{E \vdash \langle M_1, \ldots, M_k\rangle : \frown\mathbf{G},\mathrm{o}\mathbf{H}, W_1 \times \cdots \times W_k}$$

$$\frac{E \vdash N : Cap[\frown\mathbf{G},\mathrm{o}\mathbf{H}, T] \quad E \vdash M : G\frown\mathbf{G}[F] \quad E \vdash P : F \quad E \vdash F'}{E \vdash go\ N.M[P] : F'}$$

Theorem 3. *If $E \vdash P : F$ and either $P \equiv Q$ or $P \to Q$ then there are $G_1, \ldots, G_k$ such that $G_1, \ldots, G_k, E \vdash Q : F$.*

Recall the untyped example from Section 1.1. Consider two groups G and H. Let $W = G \frown \varnothing[\frown\varnothing, {}^\circ\varnothing, \mathit{Shh}]$ and set P to be the example process:

$$P = a[p[\mathit{out}\ a.\mathit{in}\ b.\langle c\rangle]] \mid b[\mathit{open}\ p.(x{:}W).x[]]$$

Let $E = G, H, a{:}W, b{:}G \frown \varnothing[\frown\{G\}, {}^\circ\{H\}, W], c{:}W, p{:}H \frown \varnothing[\frown\{G\}, {}^\circ\{H\}, W]$. Then we can derive the typings:

$$\begin{array}{l} E \vdash \mathit{out}\ a.\mathit{in}\ b.\langle c\rangle : \frown\{G\}, {}^\circ\{H\}, W \\ E \vdash \mathit{open}\ p.(x{:}W).x[] : \frown\{G\}, {}^\circ\{H\}, W \\ E \vdash P : \frown\varnothing, {}^\circ\varnothing, \mathit{Shh} \end{array}$$

From the typings $a, c : G \frown \varnothing[\frown\varnothing, {}^\circ\varnothing, \mathit{Shh}]$, we can tell that ambients a and c are immobile ambients in which nothing is exchanged and that cannot be opened. From the typings $p{:}H \frown \varnothing[\frown\{G\}, {}^\circ\{H\}, W], b{:}G \frown \varnothing[\frown\{G\}, {}^\circ\{H\}, W]$, we can tell that the ambients b and p cross only G ambients, open only H ambients, and contain W exchanges; the typing of p also tells us it can be opened. This is good, but is not fully satisfactory, since, if b were meant to be immobile, we would like to express this immobility invariant in its type. However, since b opens a subjectively mobile ambient, then b must be typed as if it were subjectively mobile itself (by the rule for *open*).

As already observed in [CGG99], this problem can be solved by replacing the subjective moves by objective moves. Let $W = G \frown \varnothing[\frown\varnothing, {}^\circ\varnothing, \mathit{Shh}]$, again, and set Q to be the example process with objective instead of subjective moves:

$$Q = a[\mathit{go}\ (\mathit{out}\ a.\mathit{in}\ b).p[\langle c\rangle]] \mid b[\mathit{open}\ p.(x{:}W).x[]]$$

Let $E = G, H, a{:}W, b{:}G \frown \varnothing[\frown\varnothing, {}^\circ\{H\}, W], c{:}W, p{:}H \frown \{G\}[\frown\varnothing, {}^\circ\{H\}, W]$, and we can derive:

$$\begin{array}{l} E \vdash \mathit{out}\ a.\mathit{in}\ b : \mathit{Cap}[\frown\{G\}, {}^\circ\varnothing, \mathit{Shh}] \\ E \vdash \mathit{go}\ (\mathit{out}\ a.\mathit{in}\ b).p[\langle c\rangle] : \frown\varnothing, {}^\circ\varnothing, \mathit{Shh} \\ E \vdash \mathit{open}\ p.(x{:}W).x[] : \frown\varnothing, {}^\circ\{H\}, W \\ E \vdash Q : \frown\varnothing, {}^\circ\varnothing, \mathit{Shh} \end{array}$$

The typings of a and c are unchanged, but the new typings of p and b are more informative. We can tell from the typing $p{:}H \frown \{G\}[\frown\varnothing, {}^\circ\{H\}, W]$ that movement of p is now due to objective rather than subjective moves. We can now tell from the typing $b{:}G \frown \varnothing[\frown\varnothing, {}^\circ\{H\}, W]$ that the ambient b is immobile.

This example suggests that in some situations objective moves lead to more informative typings than subjective moves. Still, subjective moves are essential for moving ambients containing running processes. We need such ambients to model mobile agents, for example.

5 Upper Bounds on Capabilities Imposed by Effects

Like most other type systems for concurrent calculi, ours does not guarantee liveness, for example, the absence of deadlocks. Still, we may regard the effect assigned to a process as a safety property: an upper bound on the capabilities that may be exercised by the process, and hence on its behavior. We formalize this idea in the setting of our third type system, and explain some consequences.

We begin by defining a fragment of a labelled transition system for the ambient calculus [GC99]. We say that a process P exercises a capability M, one of *in* n or *out* n or *open* n, to leave residue P' just if the M-labelled transition $P \xrightarrow{M} P'$ may be derived by the following rules:

Labelled Transitions: $P \xrightarrow{M} P'$ **where** $M \in \{in\ n, out\ n, open\ n\}$

$$\frac{P \equiv M.Q}{P \xrightarrow{M} Q} \qquad \frac{P \xrightarrow{M} P' \quad m \notin fn(M)}{(\nu m{:}W)P \xrightarrow{M} (\nu m{:}W)P'} \qquad \frac{P \xrightarrow{M} P'}{(\nu G)P \xrightarrow{M} (\nu G)P'}$$

$$\frac{P \xrightarrow{M} P'}{P \mid Q \xrightarrow{M} P' \mid Q} \qquad \frac{Q \xrightarrow{M} Q'}{P \mid Q \xrightarrow{M} P \mid Q'}$$

The following asserts that the group of the name contained in any capability exercised by a well-typed process is bounded by the effect assigned to the process. It is a corollary of Theorem 3.

Theorem 4 (Effect Safety). *Suppose that* $E \vdash P : {}^{\frown}\mathbf{G}, {}^{\circ}\mathbf{H}, T$.

(1) *If* $P \downarrow in\ n$ *then* $E \vdash n : G\,{}^{\frown}\mathbf{G}'[F]$ *for some type* $G\,{}^{\frown}\mathbf{G}'[F]$ *with* $G \in \mathbf{G}$.
(2) *If* $P \downarrow out\ n$ *then* $E \vdash n : G\,{}^{\frown}\mathbf{G}'[F]$ *for some type* $G\,{}^{\frown}\mathbf{G}'[F]$ *with* $G \in \mathbf{G}$.
(3) *If* $P \downarrow open\ n$ *then* $E \vdash n : G\,{}^{\frown}\mathbf{G}'[F]$ *for some type* $G\,{}^{\frown}\mathbf{G}'[F]$ *with* $G \in \mathbf{H}$.

To explain the operational significance of this theorem, consider a name $m : H\,{}^{\frown}\mathbf{H}'[{}^{\frown}\mathbf{G}, {}^{\circ}\mathbf{H}, T]$ and a well-typed ambient $m[P]$. Suppose that $m[P]$ is a subprocess of some well-typed process Q. We can show, by adapting standard techniques [GC99], two connections between the M-labelled transitions of the process P and the reductions immediately derivable from the whole process Q.

First, within Q, the ambient $m[P]$ can cross the boundary of another ambient named n of some group G only if either $P \xrightarrow{in\ n} P'$ or $P \xrightarrow{out\ n} P'$ for some P'. The typing rule for ambients implies that P must have effect ${}^{\frown}\mathbf{G}, {}^{\circ}\mathbf{H}, T$. Part (1) or (2) of the theorem implies that the set $\mathbf{G}$ contains G. Second, suppose that P includes a top-level ambient named n. The boundary of n can be dissolved only if $P \xrightarrow{open\ n} P'$ for some P'. Since P has effect ${}^{\frown}\mathbf{G}, {}^{\circ}\mathbf{H}, T$, part (3) of the theorem implies that the set $\mathbf{H}$ contains G. So the set $\mathbf{G}$ includes the groups of all ambients that can be crossed by $m[P]$, and the set $\mathbf{H}$ includes the groups of all ambients that can be opened within $m[P]$.

A corollary of Theorem 3 is that these bounds on ambient behavior apply not just to ambients contained within Q, but to ambients contained in any process reachable by a series of reductions from Q.

6 Conclusions

Our contribution is a new type system for tracking the behavior of mobile computations. We introduced the idea of a *name group*. A name group represents a collection of ambient names; ambient names belong to name groups in the same sense that values belong to types. We studied the properties of a new process operator $(\nu G)P$ that lexically scopes groups. Using groups, our type system can impose behavioral constraints like "this ambient crosses only ambients in one set of groups, and only dissolves ambients in another set of groups". Our previous type system for mobility [CGG99] cannot express such constraints.

In an extended version of this paper[1] we revisit an encoding of a distributed programming language that we first reported in the technical report version of our earlier work [CGG99]. In the encoding, ambients model both network nodes and the threads that may migrate between the nodes. The encoding can be typed in all three of the systems presented in this paper. The encoding illustrates how ambient groups can be used to partition the set of ambient names according to their intended usage, and how opening and crossing control allows the programmer to state some of those programming invariants which are the most interesting when programming mobile computation. For example, the typing allows threads to cross node boundaries, but not mistakenly the other way round, and guarantees that neither threads nor nodes may be opened. We use (νG) to make fresh groups for certain synchronization ambients in the encoding. The benefit of (νG) is that we can be statically assured that these synchronization ambients are known only to the processes we intend to synchronize, and propagate no further.

Our groups are similar to the *sorts* used as static classifications of names in the π-calculus [Mil99]. Our basic system of Section 2 is comparable to Milner's sort system for π, except that a *new sort* operator does not seem to have been considered in the π-calculus literature. Another difference is that sorts in the π-calculus are mutually recursive; we would have to add a recursion operator to achieve a similar effect. Our systems of Sections 3 and 4 depend on groups to constrain the opening and crossing behavior of processes. We are not aware of any uses of Milner's sorts to control process behavior beyond controlling the sorts of communicated names.

Apart from Milner's sorts, other static classifications of names occur in derivatives of the π-calculus. We mention two examples. In the type system of Abadi [Aba97] for the spi calculus, names are classified by three static *security levels*—*Public*, *Secret*, and *Any*—to prevent insecure information flows. In the flow analysis of Bodei, Degano, Nielson, and Nielson [BDNN98] for the π-calculus, names are classified by static *channels* and *binders*, again with the purpose of establishing security properties. (A similar flow analysis now exists for the ambient calculus [NNHJ99].) Although there is a similarity between these notions and groups, and indeed to sorts, nothing akin to our (νG) operator appears to have been studied.

[1] A draft technical report is at *http://research.microsoft.com/users/adg/Publications*

There is a connection between name groups and the region variables in the work of Tofte and Talpin [TT97] on region-based implementation of the λ-calculus. The store is split into a set of stack-allocated regions, and the type of each stored value is labelled with the region in which the value is stored. The scoping construct *letregion* ρ *in* e allocates a fresh region, binds it to the region variable ρ, evaluates e, and on completion, deallocates the region bound to ρ. The constructs *letregion* ρ *in* e and $(\nu G)P$ are similar in that they confer static scopes on the region variable ρ and the group G, respectively. One difference is that in our operational semantics $(\nu G)P$ is simply a scoping construct; it allocates no storage. Another is that scope extrusion laws do not seem to have been explicitly investigated for *letregion*. Still, we can interpret *letregion* in terms of (νG), and intend to report this in a future paper.

Levi and Sangiorgi's type system for a generalization of the ambient calculus [LS00] can guarantee immobility and single-threadedness. It would be interesting to consider extensions of their type system with groups.

Acknowledgements Silvano Dal Zilio commented on a draft of this paper. Ghelli acknowledges the support of Microsoft Research during the writing of this paper. The same author has also been partially supported by grants from the E.U., workgroups PASTEL and APPSEM, and by "Ministero dell'Università e della Ricerca Scientifica e Tecnologica", project INTERDATA.

References

[Aba97] M. Abadi. Secrecy by typing in security protocols. In *Proceedings TACS'97*, *LNCS* 1281, pages 611–638. Springer, 1997.

[BDNN98] C. Bodei, P. Degano, F. Nielson, and H. Nielson. Control flow analysis for the π-calculus. In *Proceedings Concur'98*, *LNCS* 1466, pages 84–98. Springer, 1998.

[CG98] L. Cardelli and A. D. Gordon. Mobile ambients. In *Proceedings FoSSaCS'98*, *LNCS* 1378, pages 140–155. Springer, 1998. Accepted for publication in *Theoretical Computer Science*.

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

[CGG99] L. Cardelli, G. Ghelli, and A. D. Gordon. Mobility types for mobile ambients. In *Proceedings ICALP'99*, *LNCS* 1644, pages 230–239. Springer, 1999.

[GC99] A. D. Gordon and L. Cardelli. Equational properties of mobile ambients. In *Proceedings FoSSaCS'99*, *LNCS* 1578, pages 212–226. Springer, 1999.

[LS00] F. Levi and D. Sangiorgi. Controlling interference in ambients. In *Proceedings POPL'00*, pages 352–364. ACM, 2000.

[Mil99] R. Milner. *Communicating and Mobile Systems: the π-Calculus*. CUP, 1999.

[NNHJ99] F. Nielson, H.R. Nielson, R.R. Hansen, and J.G. Jensen. Validating firewalls in mobile ambients. In *Proceedings Concur'99*, *LNCS* 1664, pages 463–477. Springer, 1999.

[TT97] M. Tofte and J.-P. Talpin. Region-based memory management. *Information and Computation*, 132(2):109–176, 1997. Preliminary version in *Proceedings POPL'94*.