Typing and Subtyping Mobility in Boxed Ambients*

Massimo Merro and Vladimiro Sassone

University of Sussex

Abstract

We provide a novel type system for Ambients that combines value subtyping with mobility types. The former is based on read/write exchange types, while the latter builds on the notion of ambient group. Mobility types allow to specify where an ambient is allowed to stay closing existing expressiveness gaps in the literature at no additional costs in terms of complexity. Subtyping is aimed at achieving maximal generality both communication and mobility types. We then introduce co-capabilities to express explicit permissions to access ambients. In this setting, ambient types are refined to specify who is allowed to enter an ambient, making a promising framework to model open systems.

Introduction

The calculus of *Mobile Ambients*, [8], abbreviated MA, is a novel process calculus to describe *mobile agents* which focuses on three fundamental notions: *location awareness* [9], *mobile computation* [4], and *local communication*. Papers such as [8, 5, 7, 3] demonstrate that MA can describe the run-time behaviour of mobile agents very effectively. In MA, the term $n[P]$ represents an agent, or *ambient*, named n, executing the code P. Intuitively $n[P]$ represents a *mobile*, *bounded*, and *protected* space in which the computation P takes place. Process P may: (i) contain other ambients, (ii) cause the ambient n to move, (iii) open subambients, that is dissolve the boundaries of contained ambients and absorb their contents, and (iv) communicate by dropping anonymous messages into the local ether.

Ambient names, such as n, are used to control access to the ambient computation space and may be dynamically created, as in the π-calculus [15], by the construct $(\nu n)P$; here knowledge of n is restricted to P. The ability to move and open ambient n is regulated by *capabilities* that processes possess by prior knowledge or acquire by communication. As an example, the system

$$k[\,\mathtt{in}\,n.R_1 \mid R_2\,] \;\mid\; n[\,\mathtt{open}\,k.P \mid m[\mathtt{out}\,n.Q_1 \mid Q_2]\,]$$

contains two ambients, k and n, running concurrently. The first, k, can migrate to n by virtue of its capability $\mathtt{in}\,n$. The second, n, contains a sub-ambient $m[\ldots]$, in addition to the capability $\mathtt{open}\,k$ which allows the opening of any ambient named k if any such ambient exists in the computation space of n.

Unlike other process calculi, MA focuses primarily on *process mobility* rather than *process communication*. As a consequence, the need emerged soon of studying static techniques for constraining the mobility behaviour of ambients. In [6], the authors exhibit a type system to control whether or not an ambient is mobile or may be opened. Subsequently, [5] refined that type system, enriching the type of ambient name n with a description of the ambients n may cross. In order to circumvent the need for *dependent types*, [5] introduced the idea of partitioning ambients in programmer-defined *groups*. The type of an ambient is then annotated with the capability of crossing – either from outside, via `in`, or from inside, via `out` – the border of ambients belonging to certain groups.

In our opinion, the current mobility types for Ambients have some limitations:

▷ Types in [5] are rather complex, due to the presence of the `open` capability through which ambients may acquire new capabilities by opening incoming ambients.

*Research supported by 'MyThS: Models and Types for Security in Mobile Distributed Systems', EU FET-GC IST-2001-32617. The first author is funded by EPSRC grant GR/M71169.

▷ No *subtyping* on mobility has been achieved.

▷ Mobility in MA is modelled by means of *subjective moves* triggered by the process running inside an ambient. However, the interaction of subjective moves with the open construct tends to produce typings where all ambients are typed mobile. This obliges the authors of [5] to extend MA with *objective moves*, which move ambients from the outside.

▷ Types in [5] allow an undesirable phenomenon in crossing control that is best illustrated by an example. Consider the system

$$\mathit{Odysseus}[\,\mathtt{in}\,\mathit{Horse}.\mathtt{out}\,\mathit{Horse}.\mathit{DESTROY\_TROY}\,] \;\big|\; \mathit{Horse}[\,\mathtt{in}\,\mathit{Troy}] \;\big|\; \mathit{Troy}[\,\mathit{TROJANS}]$$

where Odysseus plans to enter Troy for the well-known deed, with the well-known method. This system is typeable in [5] under type assumptions of the form

$$\begin{aligned}&\mathit{Odysseus} : \mathtt{amb}[\mathtt{Achaean}, \mathtt{cross}[\mathtt{Toy}]]\\ &\mathit{Horse} : \mathtt{amb}[\mathtt{Toy}, \mathtt{cross}[\mathtt{City}]]\\ &\mathit{Troy} : \mathtt{amb}[\mathtt{City}, \_]\end{aligned}$$

They express that *Odysseus* is an ambient of group Achaean which is allowed to cross the boundary of ambients of group Toy; *Horse* is an ambient of group Toy, which may cross ambients of group City; finally, *Troy* is an ambient of group City. Suppose now *Odysseus* moves into the *Horse*, which subsequently moves into *Troy*, so that the system evolves to

$$\mathit{Troy}[\,\mathit{TROJANS} \;\mid\; \mathit{Horse}[\mathit{Odysseus}[\mathtt{out}\,\mathit{Horse}.\mathit{DESTROY\_TROY}]]].$$

Odysseus may then move out of the *Horse* and take the *TROJANS* by surprise whom believed he did not have permission to traverse *Troy*'s walls.

▷ In MA, ambients cannot determine which agents are allowed to traverse their boundaries.

This paper aims at finding an appropriate formalism, within the ambient framework, that addresses the questions above. It is well-known that many of the difficulties of MA are caused by the open capability [2]; it thus seems a commendable endeavour to investigate variants of the ambient calculus that drop open. *Boxed Ambients* [2], abbreviated BA, is a variant of MA, from which it inherits the primitives in and out with exactly the same semantics. As for communication, BA relies on a completely different model which results from dropping the open capability, and adopting communication across ambient boundaries, between parent and children, similar to those found in the Seal calculus [17]. As pointed out in [2], BA retains much of the expressive power of MA while enhancing the flexibility of typed communication with finer-grained, more effective mechanisms. This makes BA particularly suitable for modelling classical security policies for resource protection and access control. For instance, in [1] BA is used to model *mandatory access control* policies within a multilevel security system, including both *military security* (no read-up, no write-down) and *commercial security* (no read-up, no write-up).

We introduce a type system to control mobility in BA which profits of the absence of the open capability. As in [5], we use *ambient groups* to express and enforce properties of names indirectly, as properties of groups of names, but avoiding dependent types. The absence of open simplifies types considerably, as the topic of conversation inside children is not directly relevant to an ambient and the processes it contains: effects inside the children cannot be unleashed in the current ambient. Our types for ambient names consist of four components: the group to which the ambient belongs; a component that describes the mobility constraints of the ambient, viz. where the ambient is allowed to go/reside; one that characterises the communications in which the ambient is involved; and, finally, a set of markers that specify what the ambient name can be used for. Processes and capabilities have similar types, though not identical. In particular, their mobility types express where processes and capabilities may move to, rather than where they are allowed to stay.

We also equip our type system with a non-trivial form of *subtyping* which deals with both mobility and communication aspects. Subtyping on mobility is based on the idea that a process can be used wherever processes with a more liberal behaviour (in terms of potential moves) is expected. For communication, generality is again our motor. We employ read/write types which specify separately the type of legal inputs and outputs for processes, and allow a general subsumption rule, so as to push both kinds of subtyping as far as possible. For such a system of inference we prove a set of properties expressing formally the intuitions behind our types. These, together with a subject reduction theorem, make explicit the safety guarantees upheld by the present framework.

Our types for BA meet the first four requirements discussed previously. To go further and attack the last requirement, in §4 we extend the syntax of BA with co-capabilities, which express explicit permission to traverse boundaries. The idea of introducing co-actions is borrowed from process calculi such as CCS [14] or the π-calculus [15]. Co-capabilities in ambients have been introduced first in *Safe Ambients* [12]. Our co-capabilities, however, are inspired by [18] and [13], where explicit information about the crossing ambient is required. Thus, for example, in

$$m[\,\mathtt{in}\,n.Q_1 \mid l[\,\mathtt{out}\,m.Q_2]\,] \mid n[P] \mid R,$$

m may migrate into n if P has the form $\overline{\mathtt{in}}\,m.P_1 \mid P_2$, in which case the system evolves to

$$n[\,m[\,Q_1 \mid l[\,\mathtt{out}\,m.Q_2]\,] \mid P_1 \mid P_2\,] \mid R$$

Alternatively, l may emigrate from m if R has the form $\overline{\mathtt{out}}\,l.R_1 \mid R_2$, and then the system evolves to

$$m[\,\mathtt{in}\,n.Q_1] \mid l[Q_2] \mid n[P] \mid R_1 \mid R_2.$$

We call the calculus with co-capabilities BSA, for *Boxed Safe Ambients*. Co-capabilities in BSA are always exercised by the target computation by explicitly naming the ambient allowed to traverse a boundary. In this manner, they allow to enhance ambient types by augmenting the type of n with the set of groups of the ambients allowed to cross n's boundary. The results for BA are smoothly lifted to BSA. We believe that BSA is a promising framework for the analysis of *open systems*, because no ambients can be entered – and thus attacked – by an intruder unless the intruder's group name appears in the type of the ambient.

The paper proceeds as follows. In §1 we review the untyped Boxed Ambients and their standard semantics; §2 introduces the typed variant of Boxed Ambients and presents the relative subject reduction theorem; §3 show how our types tackle the *Troy's war* example above. In §4, we introduce Boxed Safe Ambients and their typing rules, illustrating their expressiveness again on the example of §3. In this extended abstract all proof are omitted, as is much of the discussion.

1 The Boxed Ambients

In this section we review the Boxed Ambient calculus as introduced in [2]. The syntax of processes is given in Table 1, where $\mathbf{N}$ denotes an infinite set of names. Inactivity, composition, restriction and replication are inherited from mainstream concurrent calculi, most notably the π-calculus [15]. The inactive process, $\mathbf{0}$, does nothing. Parallel composition is denoted by a binary operator, $P \mid Q$, that is commutative and associative. The restriction operator, $(\boldsymbol{\nu} n)P$, creates a new (unique) name n within a scope P. Replication, $!P$, allows to create as many parallel replicas as needed and it is equivalent to $P \mid !P$. Specific of the ambient calculus are the *ambient*, $V[P]$, and the *prefix* via capabilities, $V.P$. In $V[P]$, V is the name of the ambient and P is the process running inside the ambient. The process $V.P$ executes an action regulated by the capability V, and then continues as the process P. Capabilities are obtained from names; given a name n, the capability $\mathtt{in}\,n$ allows entry into n, the capability $\mathtt{out}\,n$ allows exit out of n. Meaningless terms such as $\mathtt{in}\,n[P]$ or $n.P$ are ruled out by the type system in the next section.

Communication is (i) *synchronous* (though an asynchronous version has been considered [2]); (ii) *polyadic*, we use boldface when appropriate to represent tuples concisely, as in $(\boldsymbol{x})^{\eta}.P$ and $\langle \boldsymbol{V} \rangle^{\eta}.P$; and most importantly, (iii) *located* and *across* boundaries. Syntactically, located communication is obtained by means of tags specifying the location where the communication has to take place. More precisely, outputs (and similarly inputs) can take one of three forms: (i) $\langle \boldsymbol{V} \rangle^{n}.P$, a message for sub-ambient n from its parent; (ii) $\langle \boldsymbol{V} \rangle^{\uparrow}.P$, a message for the parent from a sub-ambient; (iii) $\langle \boldsymbol{V} \rangle^{\star}.P$, a local communication

Table 1 The Boxed Ambients

Names: $n,m,\ldots,x,y,\ldots \in \mathbf{N}$

Locations:

η ::=	n	names
$\mid$	$\uparrow$	enclosing ambient
$\mid$	$\star$	local

Processes:

P ::=	$\mathbf{0}$	nil process
$\mid$	$P_1 \mid P_2$	composition
$\mid$	$(\nu n)P$	restriction
$\mid$	$!P$	replication
$\mid$	$V[P]$	ambient
$\mid$	$V.P$	prefixing
$\mid$	$(\boldsymbol{x})^{\eta}.P$	input
$\mid$	$\langle \boldsymbol{V} \rangle^{\eta}.P$	output

Values:

V,U ::=	n	name
$\mid$	$\mathtt{in}\,V$	may enter into V
$\mid$	$\mathtt{out}\,V$	may exit out of V
$\mid$	$V_1.V_2$	path

within the current ambient boundaries. We will omit $\star$ in both $\langle \boldsymbol{V} \rangle^{\star}.P$ and $(\boldsymbol{x})^{\star}.P$, so recovering the usual MA notation.

We use a number of notational conventions. Parallel composition has the lowest precedence among the operators. The process $V.V'.P$ is read as $V.(V'.P)$. As usual, we omit trailing dead processes, writing V for $V.\mathbf{0}$, $\langle \boldsymbol{V} \rangle$ for $\langle \boldsymbol{V} \rangle.\mathbf{0}$, and $n[\,]$ for $n[\mathbf{0}]$. Restriction $(\nu n)P$ and input prefix $(\boldsymbol{x})^{\eta}.P$ acts as binders for names n and $\boldsymbol{x}$, respectively. The set of *free names* of P, $\mathrm{fn}(P)$, is defined accordingly.

The dynamics of the calculus is given in the form of a reduction relation as in Table 2. As customary in process calculi, the *reduction semantics* is based on an auxiliary relation called *structural congruence* which brings the participants of a potential interaction to contiguous positions. The formal definition can be found in the Appendix in Table 10.

The reduction rules are divided in two groups: the mobility rules and the the communication rules. The former are exactly as in MA; the latter add to the usual local communication of MA the location-based communication across boundaries. In both rules (RED COMM INPUT n) and (RED COMM OUTPUT n) the underlying anonymous channel is allocated at sub-ambient n; in both (RED COMM INPUT $\uparrow$) and (RED COMM OUTPUT $\uparrow$) the channel is allocated in the parent of n; in (RED COMM LOCAL) the channel is allocated locally. We refer to [2] for a full explanation and justification of these rules. In the communication rules we assume that tuples have the same arity.

2 Typing Mobility in Boxed Ambients

The original types for BA of [2] control that communication is well-typed, wherever it happens. As processes in BA can communicate both locally and upwards, each ambient needs to state explicitly both the local topic of conversation and the topic of the conversation in its parent ambient. There is no need to keep explicit track of communications in the children, as this information is available as the topic of conversation locally to each children.

In this section we extend the types of [2] to control both *communication* and *mobility*. The two aspects are quite orthogonal to each other and interact only in well determined places. The Typed Boxed Ambients are obtained by adding type annotations to the syntax of Table 1 and inheriting the construct $(\nu \mathtt{G})P$ of [5] for *group creation*. The formal definition can be found in the Appendix in Table 11 The operator $(\nu \mathtt{G})P$ binds $\mathtt{G}$ in P; the set of *free groups* of P, $\mathrm{fg}(P)$, is defined accordingly. As in [5], groups may appear in the type annotations of both input and name restriction, and the definition of structural congruence must be extended to keep into account the new group operator. In Table 12 of the Appendix we give the new rules to be added to those in Table 10.

Table 2 Reduction Rules

$\rightarrow$ is the least relation on processes closed under $\_ \mid Q$, $(\boldsymbol{\nu} n)\_$, $n[\_]$ such that:

mobility

$n[\mathtt{in}\, m.P \mid Q] \mid m[R]$	$\rightarrow$	$m[n[P \mid Q] \mid R]$	(RED IN)
$m[n[\mathtt{out}\, m.P \mid Q] \mid R]$	$\rightarrow$	$n[P \mid Q] \mid m[R]$	(RED OUT)

communication

$(\boldsymbol{x}).P \mid \langle \boldsymbol{V} \rangle.Q$	$\rightarrow$	$P\{\boldsymbol{V}/\boldsymbol{x}\} \mid Q$	(RED COMM LOCAL)
$(\boldsymbol{x})^n.P \mid n[\langle \boldsymbol{V} \rangle.Q \mid R]$	$\rightarrow$	$P\{\boldsymbol{V}/\boldsymbol{x}\} \mid n[Q \mid R]$	(RED COMM INPUT n)
$(\boldsymbol{x}).P \mid n[\langle \boldsymbol{V} \rangle^{\uparrow}.Q \mid R]$	$\rightarrow$	$P\{\boldsymbol{V}/\boldsymbol{x}\} \mid n[Q \mid R]$	(RED COMM OUTPUT $\uparrow$)
$\langle \boldsymbol{V} \rangle^n.P \mid n[(\boldsymbol{x}).Q \mid R]$	$\rightarrow$	$P \mid n[Q\{\boldsymbol{V}/\boldsymbol{x}\} \mid R]$	(RED COMM OUTPUT n)
$\langle \boldsymbol{V} \rangle.P \mid n[(\boldsymbol{x})^{\uparrow}.Q \mid R]$	$\rightarrow$	$P \mid n[Q\{\boldsymbol{V}/\boldsymbol{x}\} \mid R]$	(RED COMM INPUT $\uparrow$)

congruence

$P \equiv Q \quad Q$	$\rightarrow$	$R \quad R \equiv S$ implies $P \rightarrow S$	(RED STRUCT)

2.1 The types

The type system revolves around three main types: *ambient types*, *process types* and *capability types*; these are all interrelated to each other (actually mutually recursive) and annotated with information on both communication and mobility. A grammar for the types is given in Table 3.

The first component of ambient types is the group G which the ambient belongs to. Process (resp. capability) types have a similar component that establishes the group of the ambients where the process (resp. capability) can be executed.

The second component of ambient types describes the *mobility* constraints within the ambient, expressed in terms of groups rather than actual names. In the mobility type $\mathtt{mob}[\mathscr{S}]$, $\mathscr{S}$ represents the (set of groups of) ambients where the ambient in question may go/stay. This allows us to state properties such as 'the ambient n can reside in any ambient of group G'. As ambients are finally moved by processes, information about mobility must be attached to processes too. Specifically, a process must declare the set of groups of ambients it may *drive* the enclosing ambient to. This will be needed to type the ambient formation construct. For the same reason, since capabilities can be used to form processes, we need to add the same component to capability types as well.

The third component of ambient types disciplines communication along the lines of [2]. In a *communication type* $\mathtt{com}[E,F]$, the *exchange type* E represents the local topic of conversation, and F the one in the parent. As processes communicate with each other both inside and across ambients, type safety requires that the exchange of messages must be compatible with the communication type associated to the ambient. It thus follows that processes have a communication type exactly as the above. Things are slightly different for capabilities. Capabilities determine where a process carries its enclosing ambient. Although ambients cannot be opened, the processes they contain can still interact with the receiving ambients by means of upward communication. As a consequence, in order to guarantee safety in mobility, capability types must contain the information about the type of the conversation at destination.

The final component of ambient types – that we denote as an index attached to the keyword amb – is a set $\chi \subseteq \{\mathtt{i},\mathtt{o},\mathtt{c},\mathtt{r},\mathtt{w}\}$ to denote what the ambient name can be used for, viz. in and out actions, creation of an ambient, reading from and writing to a subambient. This allows to give out names together with restricted capabilities, and refines the capability-passing mechanism of the ambient calculus in that it transmits full knowledge of ambient names, yet selectively releasing rights to act on it. As a matter of notation, we write $\mathtt{amb}_{\mathtt{iow}}$ for $\mathtt{amb}_{\{\mathtt{i},\mathtt{o},\mathtt{w}\}}$ and similarly for all χ.

Values in BA consist of tuples of names and capabilities. Since our intention is to have subtyping on values, we must separate the read and write capabilities of exchange types, as initiated in [16]. We do this in a simple, standard way similar to [21, 19]. Our *exchange types* have the form $\mathtt{rw}[I,O]$ and represent the capability of reading and writing values of, respectively, the message types I and O. Message types contain tuple types $W_1 \times \ldots \times W_k$ to exchange tuples of values. For $k = 0$, the empty tuple type is called $\mathbf{1}$;

Table 3 Types for Boxed Ambients

Groups:	$\mathtt{G},\mathtt{H},\ldots$	
Sets of groups:	$\mathscr{G},\mathscr{D},\mathscr{S},\ldots$	$\mathfrak{U}$ *The universal set of groups*
Ambients types:		
	$A \ ::= \ \mathtt{amb}_\chi[\mathtt{G},M,C]$	ambient name of group $\mathtt{G}$, usable for χ-actions, $\chi \subseteq \{\mathtt{i},\mathtt{o},\mathtt{c},\mathtt{r},\mathtt{w}\}$, with mobility type M, and communication type C
Process types:		
	$\Pi \ ::= \ \mathtt{proc}[\mathtt{G},M,C]$	process that can be enclosed in an ambient of group $\mathtt{G}$, can drive its enclosing ambients to ambients whose groups are in M, and communicates as described by type C
Capability types:		
	$K \ ::= \ \mathtt{cap}[\mathtt{G},M,F]$	capability that can appear in an ambient of group $\mathtt{G}$, can drive it to ambients whose groups are in M, with exchange type F for local communication
Mobility types:		
	$M \ ::= \ \mathtt{mob}[\mathscr{G}]$	mobility specs
Communication types:		
	$C \ ::= \ \mathtt{com}[E,F]$	E is the exchange type for local communications, F is the exchange type for upward communications
Exchange types:		
	$E,F \ ::= \ \mathtt{rw}[I,O]$	read/write values of type I and O (valid if $O \lessdot I$)
Message types:		
	$I,O \ ::= \ \bot$	bottom message type
	$\mid \ W_1 \times \ldots \times W_k$	tuple (**1** is the null product)
	$\mid \ \top$	top message type
Value types:		
	$W,Y \ ::= \ A$	ambient name
	$\mid \ K$	capability

it allows the exchange of empty rules, that is, it allows pure synchronization. Besides tuple types, message types include useful bottom ($\bot$) and top ($\top$) types that allow to express interesting types. One of these is $\mathtt{zero} \triangleq \mathtt{rw}[\top,\bot]$, the least exchange type, which describes processes that engage in no communication. As customary with input/output types, we require in $\mathtt{rw}[I,O]$ that O is a subtype of I, so that a process output is compatible with (and more specific than) its input. Because of such restriction, that will be assumed throughout the paper, the set of valid types is defined simultaneously with the subtype relation $\lessdot$, whose formal definition is given in Table 4. We will discuss the details in §2.3 below. For the purposes of this section it suffices to say what follows.

Subtyping on communication allows to have processes with a communication type C inside ambients with a communication type C' if $C \lessdot C'$. For instance, a process that exchanges no messages can reside in ambients regardless of their topic of conversation.

Subtyping on mobility is essentially based on subsets of set of groups, as formalised by rule (sMOB). So, a process driving to ambients with groups in $\mathscr{D}$ is allowed to reside in ambients which can move into ambients whose groups belongs to $\mathscr{D}'$, with $\mathscr{D} \subseteq \mathscr{D}'$.

In Section 2.3 we will give a better account about subtyping.

Table 4 Subtype Relation

$$(\textsc{sAmb})\ \frac{\chi_1 \subseteq \chi_0 \subseteq \{\mathtt{i},\mathtt{o},\mathtt{c},\mathtt{r},\mathtt{w}\}}{\mathtt{amb}_{\chi_0}[\mathtt{G},M,C] \lessdot \mathtt{amb}_{\chi_1}[\mathtt{G},M,C]} \qquad (\textsc{sProc})\ \frac{M_0 \lessdot M_1; \quad C_0 \lessdot C_1}{\mathtt{proc}[\mathtt{G},M_0,C_0] \lessdot \mathtt{proc}[\mathtt{G},M_1,C_1]}$$

$$(\textsc{sCap})\ \frac{M_0 \lessdot M_1; \quad F_0 \lessdot F_1}{\mathtt{cap}[\mathtt{G},M_0,F_0] \lessdot \mathtt{cap}[\mathtt{G},M_1,F_1]} \qquad (\textsc{sMob})\ \frac{\mathscr{G}_0 \subseteq \mathscr{G}_1}{\mathtt{mob}[\mathscr{G}_0] \lessdot \mathtt{mob}[\mathscr{G}_1]}$$

$$(\textsc{sCom})\ \frac{E_0 \lessdot E_1; \quad F_0 \lessdot F_1}{\mathtt{com}[E_0,F_0] \lessdot \mathtt{com}[E_1,F_1]} \qquad (\textsc{sExc})\ \frac{I_1 \lessdot I_0; \quad O_0 \lessdot O_1}{\mathtt{rw}[I_0,O_0] \lessdot \mathtt{rw}[I_1,O_1]}$$

$$(\textsc{sMsg})\ \frac{-}{\bot \lessdot W_1 \times \ldots \times W_k \lessdot \top} \qquad (\textsc{sTuple})\ \frac{W_i \lessdot T_i; \quad i \in 1..k}{W_1 \times \ldots \times W_k \lessdot T_1 \times \ldots \times T_k}$$

Table 5 Good Values

$$(\textsc{Val}\ n)\ \frac{\Gamma, n : W, \Gamma' \vdash \diamond}{\Gamma, n : W, \Gamma' \vdash n : W}$$

$$(\textsc{Val pfx})\ \frac{\Gamma \vdash V_0 : K;\ \Gamma \vdash V_1 : K}{\Gamma \vdash V_0.V_1 : K} \qquad (\textsc{Val in})\ \frac{\Gamma \vdash V : \mathtt{amb}_{\mathtt{i}}[\mathtt{G},M,\mathtt{com}[E,F]]}{\Gamma \vdash \mathtt{in}V : \mathtt{cap}[\mathtt{H},\mathtt{mob}[\{\mathtt{G}\}],E]}\ \ \mathtt{H} \in \mathrm{dom}(\Gamma)$$

$$(\textsc{Val sub})\ \frac{\Gamma \vdash V : W; \quad W \lessdot W'}{\Gamma \vdash V : W'} \qquad (\textsc{Val out})\ \frac{\Gamma \vdash V : \mathtt{amb}_{\mathtt{o}}[\mathtt{G},M,\mathtt{com}[E,F]]}{\Gamma \vdash \mathtt{out}V : \mathtt{cap}[\mathtt{H},M,F]}\ \ \mathtt{H} \in \mathrm{dom}(\Gamma)$$

2.2 Type assignment

A type environment Γ is a list of assumptions about groups, names and their types of one of two forms: $\mathtt{G}$, declaring the existence of group $\mathtt{G}$, or $n : A$, declaring a name n of type A. Since types contain references to groups, the order of assumptions in Γ is relevant; to describe valid environments we use judgements of the form $\Gamma \vdash \diamond$ that make sure that names and groups are not repeated and that groups are defined before being referred to. The formal definition of type environments and domain of an environment, and the inference rules for *well-formed environments* can be found in the appendix in Table 13.

Table 5 presents the typing rules for *well-formed values*. Rules (Val n) and (Val sub) are straightforward. Rule (Val pfx) decrees that the type of a path of capabilities is the common type of its components. Notice that, in the light of (Val sub) and of the subtyping rule (sCap) of Table 4, this actually means that paths are assigned a common super type of their component capabilities.

In rule (Val in), exploiting the subtyping rule (sAmb), we require the in-capability on ambient name n to form $\mathtt{in}n$. The conclusion states that the capability $\mathtt{in}V$ can reside inside any ambient of group $\mathtt{H}$, for any $\mathtt{H} \in \mathrm{dom}(\Gamma)$, and can drive its enclosing ambient into an ambient of group $\mathtt{G}$ (viz. V's group) with local exchange type E (viz. V's local exchange type). Dually, rule (Val out) requires the out-capability on the ambient name. Its conclusion says that the capability $\mathtt{out}V$ can reside inside any ambient of group $\mathtt{H}$, for any $\mathtt{H} \in \mathrm{dom}(\Gamma)$. It also states that $\mathtt{out}V$ can drive the enclosing ambient into ambients with groups mentioned in M (viz. V's mobility type) and local exchange type F (viz. V's upward exchange type).

The inference rules in Tables 6 and 7 are for *well-typed processes* as expressed by judgements of the form $\Gamma \vdash P : \Pi$. We divide the rules in two groups, dealing separately with mobility and communication. Table 6 focuses on mobility. Rule (Pro pfx) builds on the subtyping rules (sCap) and (sProc) to state that the characteristics of the capabilities present in a process must be subsumed in its type. Precisely, all the capabilities in P must share with it the group $\mathtt{G}$. Also, P shares with V the mobility behaviour M, and the local exchange types of ambients it can reach must be compatible with its upward exchanges, F.

Rule (Pro Amb) is crucial. It says that a process $V[P]$ can be formed only if we have the capability to create an ambient called V. Furthermore, P can run inside V only if P and V agree on the group, the mobility type, and the communication type. The resulting ambient $V[P]$ is a process that (i) can be executed in any ambient of group $\mathtt{G}$, provided that $\mathtt{G}$ is a place where V is allowed to reside; (ii) will not drive an

Table 6 Good Processes - Mobility

$$(\text{PRO PFX})\ \frac{\Gamma \vdash V : \mathtt{cap}[\mathtt{G}, M, F]; \quad \Gamma \vdash P : \mathtt{proc}[\mathtt{G}, M, \mathtt{com}[E, F]]}{\Gamma \vdash V.P : \mathtt{proc}[\mathtt{G}, M, \mathtt{com}[E, F]]}$$

$$(\text{PRO AMB})\ \frac{\Gamma \vdash V : \mathtt{amb_c}[\mathtt{H}, \mathtt{mob}[\mathscr{S}], \mathtt{com}[E, F]]; \quad \Gamma \vdash P : \mathtt{proc}[\mathtt{H}, \mathtt{mob}[\mathscr{S}], \mathtt{com}[E, F]]}{\Gamma \vdash V[P] : \mathtt{proc}[\mathtt{G}, \mathtt{mob}[\varnothing], \mathtt{com}[F, \mathtt{zero}]]} \quad \mathtt{G} \in \mathscr{S}$$

$$(\text{PRO RES})\ \frac{\Gamma, n : A \vdash P : \Pi}{\Gamma \vdash (\boldsymbol{\nu} n : A)P : \Pi} \qquad (\text{PRO GRES})\ \frac{\Gamma, \mathtt{G} \vdash P : \Pi}{\Gamma \vdash (\boldsymbol{\nu}\mathtt{G})P : \Pi} \quad \mathtt{G} \notin \mathrm{fg}(\Pi)$$

$$(\text{PRO } \mathbf{0})\ \frac{\mathtt{G} \in \mathrm{dom}(\Gamma)}{\Gamma \vdash \mathbf{0} : \mathtt{proc}[\mathtt{G}, \mathtt{mob}[\varnothing], \mathtt{com}[\mathtt{zero}, \mathtt{zero}]]} \qquad (\text{PRO PAR})\ \frac{\Gamma \vdash P : \Pi; \ \Gamma \vdash Q : \Pi}{\Gamma \vdash P \mid Q : \Pi}$$

$$(\text{PRO REP})\ \frac{\Gamma \vdash P : \Pi}{\Gamma \vdash !P : \Pi} \qquad (\text{PRO SUB})\ \frac{\Gamma \vdash P : \Pi \quad \Pi \lessdot \Pi'}{\Gamma \vdash P : \Pi'}$$

Table 7 Good Processes - Communication

$$(\text{PRO INPUT} \star)\ \frac{\Gamma, x_1{:}W_1, \dots, x_k{:}W_k \vdash P : \mathtt{proc}[\mathtt{G}, M, \mathtt{com}[\mathtt{rw}[I, O], F]]}{\Gamma \vdash (x_1{:}W_1, \dots, x_k{:}W_k).P : \mathtt{proc}[\mathtt{G}, M, \mathtt{com}[\mathtt{rw}[I, O], F]]} \quad I \lessdot W_1 \times \dots \times W_k$$

$$(\text{PRO OUTPUT} \star)\ \frac{\Gamma \vdash V_1{:}W_1, \dots, V_k{:}W_k; \quad \Gamma \vdash P : \mathtt{proc}[\mathtt{G}, M, \mathtt{com}[\mathtt{rw}[I, W_1 \times \dots \times W_k], F]]}{\Gamma \vdash \langle V_1, \dots, V_k \rangle.P : \mathtt{proc}[\mathtt{G}, M, \mathtt{com}[\mathtt{rw}[I, W_1 \times \dots \times W_k], F]]}$$

$$(\text{PRO INPUT} \uparrow)\ \frac{\Gamma, x_1{:}W_1, \dots, x_k{:}W_k \vdash P : \mathtt{proc}[\mathtt{G}, M, \mathtt{com}[E, \mathtt{rw}[I, O]]]}{\Gamma \vdash (x_1{:}W_1, \dots, x_k{:}W_k)^{\uparrow}.P : \mathtt{proc}[\mathtt{G}, M, \mathtt{com}[E, \mathtt{rw}[I, O]]]} \quad I \lessdot W_1 \times \dots \times W_k$$

$$(\text{PRO OUTPUT} \uparrow)\ \frac{\Gamma \vdash V_1{:}W_1, \dots, V_k{:}W_k; \quad \Gamma \vdash P : \mathtt{proc}[\mathtt{G}, M, \mathtt{com}[E, \mathtt{rw}[I, W_1 \times \dots \times W_k]]]}{\Gamma \vdash \langle V_1, \dots, V_k \rangle^{\uparrow}.P : \mathtt{proc}[\mathtt{G}, M, \mathtt{com}[E, \mathtt{rw}[I, W_1 \times \dots \times W_k]]]}$$

$$(\text{PRO INPUT } V)\ \frac{\Gamma \vdash V : \mathtt{amb_r}[\mathtt{G}, M, \mathtt{com}[\mathtt{rw}[I, O], F]]; \quad \Gamma, x_1{:}W_1, \dots, x_k{:}W_k \vdash P : \Pi}{\Gamma \vdash (x_1{:}W_1, \dots, x_k{:}W_k)^{V}.P : \Pi} \quad I \lessdot W_1 \times \dots \times W_k$$

$$(\text{PRO OUTPUT } V)\ \frac{\Gamma \vdash V : \mathtt{amb_w}[\mathtt{G}, M, \mathtt{com}[\mathtt{rw}[I, W_k \times \dots \times W_k], F]]; \quad \Gamma \vdash U_1{:}W_1, \dots, U_k{:}W_k; \quad \Gamma \vdash P : \Pi}{\Gamma \vdash \langle U_1, \dots, U_k \rangle^{V}.P : \Pi}$$

enclosing ambients anywhere; and (iii) inherits P's upward exchange type as its local exchange type and, finally, has a minimal upward exchange type. As a consequence of the requirement $\mathtt{G} \in \mathscr{S}$, the set $\mathscr{S}$ is never empty; indeed an ambient, even if immobile, it always resides somewhere. Notice the role played here by the subtyping rules (SAMB) and (SPROC). The remaining rules in Table 6 are straightforward.

Table 7 adapts the type rules for communication of [2] to our setting taking subtyping into account. Essentially, the 'output rules' state that message types must be accounted for in the corresponding process types, so that no bad surprises may arise when a process containing an output is deployed in contexts. On the contrary, the 'input rules' allow process types to be subtypes of those declared in the input prefix, so that P can be deployed more generally, yet without risking exposure to messages of unexpected types.

2.3 Subtyping

Subtyping has a truly relevant impact on type systems for Ambients. As already remarked, subtyping on communication allows to have processes that exchange no messages in ambients regardless of the local topic of conversation. This idea has been developed in [21]. In the current paper we put forward the use of subtyping for mobility types, unveiling its beneficial impact and general usefulness for the purpose. The prime idea here is to allow a process that may drive ambients to $\mathscr{D}$ wherever a processes that travels to $\mathscr{D}'$, with $\mathscr{D} \subseteq \mathscr{D}'$, is expected.

Observe that capability-passing is an important mechanism for mobility, and once we allow subtyping on capabilities, we also have to allow it on messaging and split communication types in their input and output components. We therefore adopt input/output types and take on as the second objective of the paper to integrate them with the subtyping on mobility. The key rule for messages is (sEXC) that decrees that inputs behaves contravariantly and output covariantly; rule (sTUPLE) lift subtyping to tuple types in the obvious way; (sMSG) rules the existence of a bottom, $\perp$, and a top, $\top$. Our aim of fully integrating subtyping in both mobility and communication types is achieved in rules (sPROC) and (sCAP), that state the covariance of process and capability types with their mobility and communication components.

A special name with a relevant role in mobility and subtyping is $\mathfrak{U}$, the universe of all groups, whose nature is defined by assuming $\mathtt{G} \in \mathfrak{U}$, for all groups $\mathtt{G}$. This allows, for instance, to express the mobility type $\mathtt{mob}[\mathfrak{U}]$ that describes ambients which are allowed to stay everywhere. Although the other rules contain little surprise, their global equilibrium is very delicate and deserves explanation. Below we discuss the most interesting aspects.

(sAMB). Remarkably, ambient types must be kept invariant, apart from the capabilities χ that – as typical of these 'may-use-for' situations – follow a contravariant typing. It is indeed easy to see that no kind of variance is admissible for mobility or communication, as they both give rise to phony capabilities that trick processes into runtime errors. Let us focus on mobility. If ambient types were to be covariant, their mobility assumptions could be easily defied. Processes might falsely believe that, say, any ambient n at type $\mathtt{amb}_\chi[\_,\mathtt{mob}[\mathscr{S}],\_]$, with $\mathtt{G} \notin \mathscr{S}$, is allowed to stay in any (ambient of group) $\mathtt{G}$. In this manner, n might end up sitting where it should not. The same net effect is obtained by assuming contravariance, due to the interplay with covariance of process types in their mobility components. In the example below, for instance, m tricks itself into believing that n will never go to k, and again it ends up where it should not. Assume n, m and k have types $\mathtt{amb}_{\chi_n}[\mathtt{N},\mathtt{mob}[\{\mathtt{T},\mathtt{K}\}],\_]$, $\mathtt{amb}_{\chi_m}[\mathtt{M},\mathtt{mob}[\{\{\mathtt{T},\mathtt{N}\}],\_]$, $\mathtt{amb}_{\chi_k}[\mathtt{K},\mathtt{mob}[\{\mathtt{T}\}],\_]$, respectively. Then the term $m[\mathtt{in}\,n.\mathtt{out}\,n] \mid n[\mathtt{in}\,k] \mid k[\,]$ would be typeable, while it becomes evident after reduction to $k[m[\,] \mid n[\,]]$, that it should not be so, as m ends up in k even though $\mathtt{K}$ is not mentioned in m's mobility type.

Similar arguments can be used to show that (sAMB) must be invariant also in the communication component. In particular, if C were allowed to vary either covariantly or contravariantly, we could assign a type to the process

$$n[(x{:}W).\mathbf{0}] \mid m[\mathtt{in}\,n.\langle V\rangle^{\uparrow}.\mathbf{0}]$$

with $V : Y$ and $Y \not\leqslant W$ which, after reducing to $n[(x{:}W).\mathbf{0} \mid m[\langle V\rangle^{\uparrow}.\mathbf{0}]]$ manifests a runtime error and, therefore, a flaw in the type system. The derivations, of course different in the two cases are left to the reader; we limit ourselves to comment that in the case of contravariance, the problems arise from a conflict with the (covariant) capability rules, while for covariance it is possible to use subtyping to give n and m two different, incompatible views of the exchange type in n.

It is interesting to observe that the approach of [21] does not suffer from mis-interactions between rules for ambient names and rules for capabilities, even though subtyping is contravariant for the former and covariant for the latter. This results from Zimmer's ambients coming equipped with two (otherwise equal) types: a covariant and a contravariant one. As an effect, the covariant type will at any time be an upper bound of the original ambient type, and is used to turn subtyping 'off' in sensitive spots, viz. typing the open capability. Our subtyping relations suffers the same difficulties when we replace 'open' with 'upward communication.' In the present paper we have renounced to subtyping on names, but the double typing approach of [21] could be adapted, mutatis mutandis, to our framework.

Although in [2] there is no explicit subtyping on names or capabilities, the rules corresponding to our (VAL IN) and (VAL OUT) allow $\mathtt{in}\,n$ and $\mathtt{out}\,n$ in the conclusion an exchange type that is a subtype of the one declared for n. This is effectively a form of contravariant subtyping on names. We believe that our approach has advantages over that, as it allows deep, general subtyping on exchange types which, as discussed below, constitutes a major difference with the approach of [2].

(sCOM). Differently from [2], our subtyping is covariant in both components of communication types. Expressed in our terms, the process subtyping in *loc. cit.* is limited to decreeing $\mathtt{com}[\mathtt{zero},F] \lessdot \mathtt{com}[E,F]$.

The key of the matter is that such limitation buys Bugliesi et al.'s ambients the ability to travel ambients of non-homogeneous types at the price of having no communication once at destination. Our type system surrenders such a privilege in exchange for a non trivial subtyping on the second component of communication types. To exemplify, assume that n_1 and n_2 have communication types $\texttt{com}[E,F_1]$ and $\texttt{com}[E,F_2]$, respectively, with $E \leqslant F_1, F_2$, and F_1 and F_2 incompatible exchange types. Then, contrarily to Bugliesi et al., we cannot type

$$\texttt{in}\,n_1.\texttt{in}\,n_2.\mathbf{0} \quad \text{nor} \quad \texttt{in}\,n_1.\mathbf{0} \mid \texttt{in}\,n_2.\mathbf{0}.$$

However, using subtyping on the upward exchange types, we can type $\texttt{in}\,n_1.\mathbf{0} \mid \langle w\rangle^{\uparrow}.\mathbf{0}$, for say $w : F_2$.

2.4 Properties of typing

As stated below, our subtype relation satisfies a certain number of properties.

Theorem 2.1 (Properties of $\leqslant$) The subtype relation is a partial order with all bounded joins and meets.

Existing joins ($\sqcup$) and meets ($\sqcap$) can be expressed quite easily starting from componentwise set union and intersection for mobility types, e.g., $\texttt{mob}[\mathscr{S}_0] \sqcup \texttt{mob}[\mathscr{S}_1] \triangleq \texttt{mob}[\mathscr{S}_0 \cup \mathscr{S}_1]$, from the obvious roles of $\perp$ and $\top$ among message types, and from their covariant/contravariant extension to exchange types, that is

$$\texttt{rw}[I_0,O_0] \sqcup \texttt{rw}[I_1,O_1] \triangleq \texttt{rw}[I_0 \sqcap I_1, O_0 \sqcup O_1] \qquad \texttt{rw}[I_0,O_0] \sqcap \texttt{rw}[I_1,O_1] \triangleq \texttt{rw}[I_0 \sqcup I_1, O_0 \sqcap O_1].$$

As a whole, our type systems admits a quite simple algorithmic version – i.e., an equivalent type system with no subsumption rules nor reflexivity and transitivity rules for $\leqslant$ – that makes relatively easy to prove the following results, intended to formalise the safety guarantees provided by our type system. In the following, we use '_' to avoid naming irrelevant parts of types.

Theorem 2.2 (Communication Properties) Whenever for some types $W_1 \times \ldots \times W_k$ and Π one of the following holds,

$$\Gamma \vdash (x_1{:}W_1,\ldots,x_k{:}W_k).P \mid \langle V_1,\ldots,V_k\rangle.P : \Pi;$$
$$\Gamma \vdash (x_1{:}W_1,\ldots,x_k{:}W_k).P \mid m[\langle V_1,\ldots,V_k\rangle^{\uparrow}.P] : \Pi; \quad \Gamma \vdash \langle V_1,\ldots,V_k\rangle.P \mid m[(x_1{:}W_1,\ldots,x_k{:}W_k)^{\uparrow}.P] : \Pi,$$
$$\Gamma \vdash (x_1{:}W_1,\ldots,x_k{:}W_k)^m.P \mid m[\langle V_1,\ldots,V_k\rangle.P] : \Pi; \quad \Gamma \vdash \langle V_1,\ldots,V_k\rangle^m.P \mid m[(x_1{:}W_1,\ldots,x_k{:}W_k).P] : \Pi$$

then $\Gamma \vdash V_1 : Y_1,\ldots,V_k : Y_k$ with $Y_1 \times \ldots \times Y_k \leqslant W_1 \times \ldots \times W_k$.

Theorem 2.3 (Mobility Properties) Whenever $\Gamma \vdash n[\texttt{in}\,m.P \mid Q] \mid m[R] : \Pi$, then

$$\Gamma \vdash m : \texttt{amb}_{\chi_0}[\texttt{M},\_,\_] \quad \text{and} \quad \Gamma \vdash n : \texttt{amb}_{\chi_1}[\_,\texttt{mob}[\mathscr{S}],\_]$$

with $\texttt{M} \in \mathscr{S}$, $\texttt{i},\texttt{c} \in \chi_0$ and $\texttt{c} \in \chi_1$. Moreover, if for some type Π we have $\Gamma \vdash m[n[\texttt{out}\,m.P \mid Q] \mid R] : \Pi$, then

$$\Gamma \vdash m : \texttt{amb}_{\chi_0}[\texttt{M},\texttt{mob}[\mathscr{S}_m],\_] \quad \text{and} \quad \Gamma \vdash n : \texttt{amb}_{\chi_1}[\texttt{N},\texttt{mob}[\mathscr{S}_n],\_]$$

with $\texttt{o},\texttt{c} \in \chi_0$, $\texttt{c} \in \chi_1$, $\texttt{M} \in \mathscr{S}_n$ and $\mathscr{S}_m \subseteq \mathscr{S}_n$.

Theorem 2.4 (Subject Congruence and Reduction) If $\Gamma \vdash P : \Pi$ and $P \equiv Q$ or $P \rightarrow Q$, then there exist groups $\texttt{G}_0,\ldots\texttt{G}_k$ such that $\texttt{G}_0,\ldots,\texttt{G}_k,\Gamma \vdash Q : \Pi$.

3 Some MyThS

As pointed out in the introduction, the system

$$Odysseus[\mathtt{in}\,Horse.\mathtt{out}\,Horse.DESTROY\_TROY] \mid Horse[\mathtt{in}\,Troy] \mid Troy[TROJANS]$$

can be typed with the types of [5] under hypotheses that say very little about the intentions of *Odysseus* to traverse *Troy*'s walls or indeed about the permission he might hold for doing so. On the contrary, in our system the term above can be only typed under assumptions of the kind

$$\begin{array}{l} Odysseus : \mathtt{amb}_{\mathtt{c}}[\mathtt{Achaean}, \mathtt{mob}[\{\mathtt{Ground}, \mathtt{Toy}, \mathtt{City}\}], \_] \\ Horse : \mathtt{amb}_{\mathtt{ioc}}[\mathtt{Toy}, \mathtt{mob}[\{\mathtt{Ground}, \mathtt{City}\}], \_] \\ Troy : \mathtt{amb}_{\mathtt{ioc}}[\mathtt{City}, \_, \_] \end{array}$$

that make explicit the hypotheses that *Odysseus* is a `Achaean` intentioned to move into a `City`. Such information should be enough to alert the *TROJANS* about an attack on *Troy* by *Odysseus*. On the other hand, under assumptions of the form

$$Odysseus : \mathtt{amb}_{\mathtt{c}}[\mathtt{Achaean}, \mathtt{mob}[\{\mathtt{Ground}, \mathtt{Toy}\}], \_]$$

the *TROJANS* should not fear any attack from *Odysseus*.

So far so good. Provided the *TROJANS* are in a position of believing *Odysseus*'s declared intentions of just 'moving into some `Toy`' everything is in place. But what if they suspect that the wily *Odysseus* might be lying about his real intentions, that is about his real type? The situation would then be perfectly analogous to what happens in the real-world of *open-ended systems*, where we cannot ask – nor afford – to type-check all systems which we interact with and, at the same time, we cannot trust their declarations of goodwill. What precautions can be taken to avoid being attacked by a malicious agent? What precautions can the *TROJANS* take to avoid *Odysseus*'s ravage?

A first solution is to have a type inference algorithms to infer a proper type for the external code. Then, analysing the mobility type of *Odysseus*, *Troy* can check whether its enemy is intentioned to move in. A more robust alternative is to provide *Troy* with a tool to explicitly declare the ambients allowed to move in. For instance, one may extend the mobility types with a second component that records the groups of those agents which are allowed to move into the ambient (in this case *Troy*). This would permit checking `in` movements by simply testing whether the moving ambients is accepted at destination. Unfortunately, we could not do the same test for `out` actions because target ambients in `out` movements may vary at run-time.

So, we extend the syntax with a variant of Levi and Sangiorgi's co-capabilities [12], in order to express explicit access permissions, and provide appropriate ambient types to address the questions raised in the example above.

4 Safe Boxed Ambients

In this section we introduce the Boxed Safe Ambients, abbreviated BSA, a variant of BA enriched with co-capabilities to express permissions to move into ambients. Using co-capabilities, a BSA ambient can discriminate which ambient groups are allowed to enter its computation space, and when. In the following, we first adapt the type theory of BA to BSA, then we lift subject reduction to BSA, and finally we apply the results to refine the treatment of our running example.

The syntax of BSA is obtained by extending the definition of Values of Table 11 as follows:

$$\begin{array}{lcll} V & ::= & \ldots & \text{as in Typed Boxed Ambients} \\ & \mid & \overline{\mathtt{in}}\,\alpha \quad \alpha \in \{n, \star\} & \text{allow enter of } n \text{ or of all} \\ & \mid & \overline{\mathtt{out}}\,\alpha \quad \alpha \in \{n, \star\} & \text{allow exit of } n \text{ or of all} \end{array}$$

The definition of co-capability in BSA, inspired by [12, 13, 18], is a novel combination of existing proposals. In particular, both $\overline{\mathtt{in}}$ and $\overline{out}$ co-capabilities are placed in the target computation, as in [13]. However,

Table 8 Good Values in BSA (overlay on Table 5)

(VAL $\mathtt{in}V$)
$$\frac{\Gamma \vdash V : \mathtt{amb_i}[\mathtt{G}, M, \mathtt{com}[E,F]]}{\Gamma \vdash \mathtt{in}V : \mathtt{cap}[\mathtt{H}, \mathtt{mob}[\{\mathtt{G}\}, \varnothing], E]}\ \mathtt{H} \in \mathrm{dom}(\Gamma)$$

(VAL $\mathtt{out}\,V$)
$$\frac{\Gamma \vdash V : \mathtt{amb_o}[\mathtt{G}, \mathtt{mob}[\mathscr{S},\mathscr{C}], \mathtt{com}[E,F]]}{\Gamma \vdash \mathtt{out}\,V : \mathtt{cap}[\mathtt{H}, \mathtt{mob}[\mathscr{S}, \varnothing], F]}\ \mathtt{H} \in \mathrm{dom}(\Gamma)$$

(VAL $\overline{\mathtt{in}}V$)
$$\frac{\Gamma \vdash V : \mathtt{amb_i}[\mathtt{G}, M, \mathtt{com}[E,F]]}{\Gamma \vdash \overline{\mathtt{in}}V : \mathtt{cap}[\mathtt{H}, \mathtt{mob}[\varnothing, \{\mathtt{G}\}], F]}\ \mathtt{H} \in \mathrm{dom}(\Gamma)$$

(VAL $\overline{\mathtt{out}}\,V$)
$$\frac{\Gamma \vdash V : \mathtt{amb_o}[\mathtt{G}, M, \mathtt{com}[E,F]]}{\Gamma \vdash \overline{\mathtt{out}}\,V : \mathtt{cap}[\mathtt{H}, \mathtt{mob}[\varnothing, \{\mathtt{G}\}], F]}\ \mathtt{H} \in \mathrm{dom}(\Gamma)$$

(VAL $\overline{\mathtt{in}}\star$)
$$\frac{\mathtt{G} \in \mathrm{dom}(\Gamma)}{\Gamma \vdash \overline{\mathtt{in}}\star : \mathtt{cap}[\mathtt{G}, \mathtt{mob}[\varnothing, \mathfrak{U}], \mathtt{zero}]}$$

(VAL $\overline{\mathtt{out}}\star$)
$$\frac{\mathtt{G} \in \mathrm{dom}(\Gamma)}{\Gamma \vdash \overline{\mathtt{out}}\star : \mathtt{cap}[\mathtt{G}, \mathtt{mob}[\varnothing, \mathfrak{U}], \mathtt{zero}]}$$

unlike [13], we allow both explicit and anonymous co-capabilities as in [18] and [12], respectively. The former indicates explicitly the name of the ambient allowed to cross the ambient boundary; the latter, denoted by $\star$, represents a general permission to cross the boundary usable by all.

The definition of structural congruence for BSA is the same as in Table 12. The reduction relation is obtained by replacing the rules (Red In) and (Red Out) of Table 2 with the following rules.

$$n[\mathtt{in}\,m.P \mid Q] \ \mid\ m[\overline{\mathtt{in}}\,\alpha.R \mid S] \ \rightarrow\ m[n[P \mid Q] \mid R \mid S] \quad \text{for } \alpha \in \{\star, n\} \qquad (\text{RED IN})$$

$$m[n[\mathtt{out}\,m.P \mid Q] \mid R] \ \mid\ \overline{\mathtt{out}}\,\alpha.S \ \rightarrow\ n[P \mid Q] \mid m[R] \mid S \quad \text{for } \alpha \in \{\star, n\} \qquad (\text{RED OUT})$$

The types for BSA are obtained by enriching the mobility types of Table 3 with an extra information about the groups of ambients allowed to stay in the ambient in question:

$$M \ ::= \ \mathtt{mob}[\mathscr{S},\mathscr{C}]$$

The rules for subtyping of Table 4 are adapted to BSA by replacing the rule (sMOB) with:

$$(\text{sMOB})\ \frac{\mathscr{G}_0 \subseteq \mathscr{G}_1, \quad \mathscr{C}_0 \subseteq \mathscr{C}_1}{\mathtt{mob}[\mathscr{G}_0, \mathscr{C}_0] \leqslant \mathtt{mob}[\mathscr{G}_1, \mathscr{C}_1]}$$

The remaining rules are unchanged.

The inference rules for well-typed values of Table 5 must be adapted to take into account the new component of the capability types, as in Table 8. Rules (VAL $\mathtt{in}V$) and (VAL $\mathtt{out}\,V$) are like the corresponding ones in Table 5. We simply add a second empty component to the mobility type, denoting that these capabilities allow no incoming code.

Dually, rules (VAL $\overline{\mathtt{in}}V$) and (VAL $\overline{\mathtt{out}}\,V$) do not trigger movements, but allow inbound code. Thus, the resulting mobility types in both conclusions have the first component empty, while the second one record the groups of possible visitors (i.e. the group of V). The third component of both capability types is the upward exchange type of V.

As for rules (VAL $\overline{\mathtt{in}}\star$) and (VAL $\overline{\mathtt{out}}\star$), they differ from the previous ones in the second component of the mobility types where the universal set of groups $\mathfrak{U}$ is used to reflect the anonymous nature of the construct. Moreover, in both cases the exchange type is $\mathtt{zero}$, the least exchange type, because nothing is known about the communication types of the incoming ambients. The inference rules for well-typed processes are changed as in Table 9; the remaining rules are changed in the obvious manner. Notice the second mobility component in the conclusion of the rule (PRO AMB).

Our results extends smoothly to BSA. In particular, the Subject Reduction Theorem 2.4 can be restated in formally identical terms for BSA as for BA. The new results concern the added control power granted to BSA by co-capabilities.

Theorem 4.1 (Control Properties) *In addition to the properties of Theorems 2.2 and 2.3, whenever*

$$\Gamma \vdash m[\overline{\mathtt{in}}\alpha.P \mid Q] : \Pi \quad \textit{or} \quad \Gamma \vdash m[\,\overline{\mathtt{out}}\,\alpha.P \mid Q\,] : \Pi \quad \textit{with } \alpha \in \{\star, n\},$$

then $\Gamma \vdash m : \mathtt{amb}_{\chi_0}[\_, \mathtt{mob}[\_, \mathscr{C}], \_]$ *and*

Table 9 Good Processes - Mobility (overlay on Table 6)

(PRO PFX)
$$\frac{\Gamma \vdash V : \mathtt{cap}[\mathtt{G}, M, F]; \quad \Gamma \vdash P : \mathtt{proc}[\mathtt{G}, M, \mathtt{com}[E, F]]}{\Gamma \vdash V.P : \mathtt{proc}[\mathtt{G}, M, \mathtt{com}[E, F]]}$$

(PRO $\mathbf{0}$)
$$\frac{\mathtt{G} \in \mathrm{dom}(\Gamma)}{\Gamma \vdash \mathbf{0} : \mathtt{proc}[\mathtt{G}, \mathtt{mob}[\varnothing, \varnothing], \mathtt{com}[\mathtt{zero}, \mathtt{zero}]]}$$

(PRO AMB)
$$\frac{\Gamma \vdash V : \mathtt{amb}_{\mathtt{c}}[\mathtt{H}, \mathtt{mob}[\mathscr{S}, \mathscr{C}], \mathtt{com}[E, F]]; \quad \Gamma \vdash P : \mathtt{proc}[\mathtt{H}, \mathtt{mob}[\mathscr{S}, \mathscr{C}], \mathtt{com}[E, F]]}{\Gamma \vdash V[P] : \mathtt{proc}[\mathtt{G}, \mathtt{mob}[\varnothing, \{\mathtt{H}\}], \mathtt{com}[F, \mathtt{zero}]]} \quad \mathtt{G} \in \mathscr{S}$$

- *either* $\alpha = \star$ *and* $\mathscr{C} = \mho$,
- *or* $\alpha = n$ *with* $\Gamma \vdash n : \mathtt{amb}_{\chi_1}[\mathtt{N}, \_, \_]$ *and* $\mathtt{N} \in \mathscr{C}$.

4.1 Using co-capabilities to defend *Troy*

The system discussed in §3 can be rewritten in BSA as follows.

$$\begin{aligned} \mathit{THE\_TROJAN\_WAR} \;\triangleq\; & \mathit{Odysseus}[\mathtt{in}\,\mathit{Horse}.\mathtt{out}\,\mathit{Horse}.\mathit{DESTROY\_TROY}] \\ & \mid \mathit{Horse}[\overline{\mathtt{in}}\,\star.\mathtt{in}\,\mathit{Troy}] \\ & \mid \mathit{Troy}[\overline{\mathtt{in}}\,\mathit{Horse}.\mathit{TROJANS} \mid \overline{\mathtt{out}}\,\mathit{Odysseus}.\mathit{SINON}] \end{aligned}$$

Here, although the *TROJANS* let the *Horse* inside the city walls, *Odysseus* still needs a co-capability, such as $\overline{\mathtt{out}}\,\mathit{Odysseus}$ executed by *SINON* from *Troy*, to be able to get out of the *Horse*. However, a behaviour like *SINON*'s from a process running in *Troy* can only be well-typed if *Troy* has a type allowing ambients of group Achaean to enter *Troy*, as for instance the following one

$$\mathit{Troy} : \mathtt{amb}_{\mathtt{ioc}}[\mathtt{City}, \mathtt{mob}[\_, \{\mathtt{Toy}, \mathtt{Achaean}\}], \_]$$

Of course, such a choice would be suicidal for *Troy*!

On the other hand, consider the system below, where we remove the security breach represented by *SINON*.

$$\begin{aligned} \mathit{THE\_TROJAN\_TRAP} \;\triangleq\; & \mathit{Odysseus}[\mathtt{in}\,\mathit{Horse}.\mathtt{out}\,\mathit{Horse}.\mathit{DESTROY\_TROY}] \\ & \mid \mathit{Horse}[\overline{\mathtt{in}}\,\star.\mathtt{in}\,\mathit{Troy}] \mid \mathit{Troy}[\overline{\mathtt{in}}\,\mathit{Horse}.\mathit{TROJANS}] \end{aligned}$$

This situation would be perfectly safe for *Troy* (but dangerous for *Odysseus*!) provided we can type it under the assumptions of the form

$$\begin{aligned} & \mathit{Odysseus} : \mathtt{amb}_{\mathtt{c}}[\mathtt{Achaean}, \_, \_] \\ & \mathit{Horse} : \mathtt{amb}_{\mathtt{ioc}}[\mathtt{Toy}, \_, \mathtt{com}[E, \mathbf{0}]] \\ & \mathit{Troy} : \mathtt{amb}_{\mathtt{ioc}}[\mathtt{City}, \mathtt{mob}[\varnothing, \mathscr{C}], \_] \end{aligned}$$

with $\mathtt{Achaean} \notin \mathscr{C}$. In fact, even though the *Horse* gets inside the walls no *TROJANS* will ever give *Odysseus* permission to get out of it (and he will be stuck to starvation inside it), as assured by the condition for $\mathtt{Achaean} \notin \mathscr{C}$. Furthermore, the upward exchange type of the *Horse* prevents leakages of information from *Troy*.

5 Conclusion and Related Work

We have presented a powerful type system to control communication and mobility within Bugliesi *et al.*'s Boxed Ambients. A major feature of our approach is the extensive use of a subtyping relation well integrated and crucially relied upon within the system. Our type system as a whole enjoys the expected property of subject reduction and provides strong safety guarantees for well-typed terms. These results are formally stated in §2 and §4 and exemplified in §3. We are currently investigating *type inference* algorithms.

Our types and subtyping compare well with related proposals in the literature, as we justified in §2. In particular, while the treatment of exchange types is similar to Zimmer's [21] and Yoshida and Hennessy's [19] – in turn elaborations over Pierce and Sangiorgi's input/output types [16] – we know of no similar approaches to mobility types. Work of Yoshida and Hennessy's [20] and Castagna *et al.*'s [11] use subtyping on interface types for processes of, respectively, a higher-order π calculus and the Seal calculus. These approaches, however, are not directly concerned with control of subjective mobility. On the contrary, De Nicola *et al.*'s [10] focuses on mobility control using subtyping over process capabilities, but for a Linda-like language rather far from ours.

In the secons part of the paper we extended the Boxed Ambients with a version of co-capabilities where the entering/exiting ambient can, though need not, be explicitly mentioned. The information gathered in ambient types, together with the power granted to the calculus by co-capabilities, allow to express properties not expressible in previous frameworks. In particular, they permit to deal with situations typical of open-ended systems, as we exemplified by means of the *Troy's War* example. We believe that our approach here is totally compatible with the *moded types* of [2] and plan in future work to investigate the matter further.

Finally, notice that within our type theory it is possible to analyse immobility properties of ambients. Namely, $n[P]$ is immobile when P can be assigned a type of the form $\mathtt{proc}[\_,\mathtt{mob}[\varnothing,\_],\_]$. However, prescribing that an ambient cannot be moved is a different matter. Recall that in ambient types the mobility component– differently from process and capability types – describes the allowed places for the ambient to *stay*, and not to *travel to*. Therefore, taking $\mathtt{mob}[\varnothing]$ as mobility type will not help. (As a matter of fact, as revealed by inspection of rule (PRO AMB), $\mathscr{S}$ can never be empty; indeed, an ambient will have reside somewhere.) This is, we believe, a minor point that can be handled on top of the existing framework with minimal changes. For instance, it would be enough to think of the set of group names as partitioned in two infinite subsets: the mobile and the immobile groups. Then, we only need to add to (VAL IN) and (VAL OUT) the side condition that $\mathtt{H}$ is mobile. Ambients could then be declared immobile by assigning them – either in Γ or in a name declarations – to an immobile group.

References

[1] M. Bugliesi, G. Castagna and S. Crafa. Reasoning about security in mobile ambients. In *Proc. CONCUR 2001*, volume 2154 Lecture Notes in Computer Science, Springer, 2001.

[2] M. Bugliesi, G. Castagna, and S. Crafa. Boxed ambients. In *Proc. TACS 2001*, volume 2215 of Lecture Notes in Computer Science, Springer, 2001.

[3] L. Cardelli and G. Ghelli. A query language based on the ambient logic. In *Proc. ESOP 2001*, volume 2028 of Lecture Notes in Computer Science, Springer, 2001.

[4] L. Cardelli. Wide area computation, In *Proc. ICALP 1999*, volume 1644 of Lecture Notes in Computer Science, Springer, 1999.

[5] L. Cardelli, G. Ghelli, and A. Gordon. Ambient groups and mobility types. In *Proc. IFIP TCS 2000*, volume 1872 of Lecture Notes in Computer Science, Springer, 2000.

[6] L. Cardelli and A. Gordon. Types for mobile ambients. In *Proc. of POPL'99*, ACM Press, 1999.

[7] L. Cardelli and A. Gordon. Anytime, anywhere: Modal logics for mobile ambients. In *Proc. of POPL 2000*, ACM Press, 2000.

[8] L. Cardelli and A. Gordon. *Mobile ambients*. Theoretical Computer Science, 240(1):177–213, 2000. An extended abstract appeared in *Proc. of FoSSaCS 1998*, volume 1378 of *Lecture Notes in Computer Science*, Springer, 1998.

[9] I. Castellani. Process algebras with localities. In J. Bergstra, A. Ponse, and S. Smolka, (Eds), *Handbook of Process Algebra*, 945–1045, North-Holland, 2001.

[10] R. De Nicola, G. Ferrari, and R. Pugliese. *Klaim: A kernel language for agents interaction and mobility*. IEEE Trans. on Software Engineering, 24(5), IEEE Press, 1998.

[11] G. Ghelli G. Castagna and F. Zappa Nardelli. Typing mobility in the seal calculus. In *Proc. CONCUR 2001*, volume 2154 Lecture Notes in Computer Science, Springer, 2001.

[12] F. Levi and D. Sangiorgi. Controlling interference in ambients. In *Proc. POPL 2000*, ACM Press.

[13] M. Merro and M. Hennessy. Bisimulation congruences in safe ambients. *Proc. POPL'02*, ACM Press, 2002

[14] R. Milner. *Communication and Concurrency*. Prentice Hall, 1989.

[15] R. Milner, J. Parrow, and D. Walker. *A calculus of mobile processes, (Parts I and II)*. Information and Computation, 100:1–77, 1992.

[16] B. Pierce and D. Sangiorgi. *Typing and subtyping for mobile processes*. Journal of Mathematical Structures in Computer Science, 6(5):409–454, 1996.

[17] J. Vitek and G. Castagna. Seal: A framework for secure mobile computations. In *Internet Programming Languages*, volume 1686 of Lecture Notes in Computer Science, Springer, 1999.

[18] Y. Yang, X. Guan, and J. You. *Typing evolving ambients*. Information Processing Letters, 80(5):265–270, 2001.

[19] N. Yoshida and M. Hennessy. Subtyping and locality in distributed higher order processes. In *Proc. CONCUR 1999*, volume 1664 of Lecture Notes in Computer Science, Springer, 1999.

[20] N. Yoshida and M. Hennessy. Assigning types to processes. In *Proc. LICS 2000*, IEEE Press, 2000.

[21] P. Zimmer. Subtyping and typing algorithms for mobile ambients. In *Proc. FoSSaCS 2000*, volume 1784 of Lecture Notes in Computer Science, Springer, 2000.

A Further Definitions

Table 10 Structural Congruence

$\equiv$ is the least congruence relation on processes such that:

$P \mid \mathbf{0} \equiv P$	(STRUCT PAR)
$P \mid Q \equiv Q \mid P$	(STRUCT PAR COMM)
$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$	(STRUCT PAR ASSOC)
$(\boldsymbol{\nu} n)\mathbf{0} \equiv \mathbf{0}$	(STRUCT ZERO RES)
$(\boldsymbol{\nu} n)(\boldsymbol{\nu} m)P \equiv (\boldsymbol{\nu} m)(\boldsymbol{\nu} n)P, \quad \text{if } n \neq m$	(STRUCT RES RES)
$(\boldsymbol{\nu} n)(P \mid Q) \equiv P \mid (\boldsymbol{\nu} n)Q, \quad \text{for } n \notin \text{fn}(P)$	(STRUCT RES PAR)
$(\boldsymbol{\nu} n)(m[P]) \equiv m[(\boldsymbol{\nu} n)P], \quad \text{for } n \neq m$	(STRUCT RES AMB)
$(M.M').P \equiv M.(M'.P)$	(STRUCT PATH ASSOC)
$!\mathbf{0} \equiv \mathbf{0}$	(STRUCT REPL $\mathbf{0}$)
$!P \equiv P \mid\, !P$	(STRUCT REPL PAR)

Table 11 The Typed Boxed Ambients (add-on to Table 1)

Processes:

P	$::=$	$\ldots$	boxed ambients
	$\mid$	$(x_1 : W_1, \ldots, x_k : W_k)^\eta.P$	typed input (replace previous rule)
	$\mid$	$(\boldsymbol{\nu} n : A)P$	typed names (replace previous rule)
	$\mid$	$(\boldsymbol{\nu} \mathtt{G})P$	group creation

Table 12 Extended Structural Congruence (add-on to Table 10)

$(\boldsymbol{\nu}\mathtt{G})(\boldsymbol{\nu}\mathtt{H})P \equiv (\boldsymbol{\nu}\mathtt{H})(\boldsymbol{\nu}\mathtt{G})P$	(Struct GRes GRes)
$(\boldsymbol{\nu}\mathtt{G})(\boldsymbol{\nu}n : W)P \equiv (\boldsymbol{\nu}n : W)(\boldsymbol{\nu}\mathtt{G})P$, if $\mathtt{G} \notin \mathrm{fg}(W)$	(Struct GRes Res)
$(\boldsymbol{\nu}\mathtt{G})(P \mid Q) \equiv P \mid (\boldsymbol{\nu}\mathtt{G})Q$, if $\mathtt{G} \notin \mathrm{fg}(P)$	(Struct GRes Par)
$(\boldsymbol{\nu}\mathtt{G})n[P] \equiv m[(\boldsymbol{\nu}\mathtt{G})P]$	(Struct GRes Amb)

Table 13 Good Environment: $\Gamma \vdash \diamond$

$\Gamma \ ::= \ \emptyset \ \mid \ \Gamma, \mathtt{G} \ \mid \ \Gamma, n : A$

$\mathrm{dom}(\varnothing) \stackrel{\text{def}}{=} \varnothing \qquad \mathrm{dom}(\Gamma, \mathtt{G}) \stackrel{\text{def}}{=} \mathrm{dom}(\Gamma) \cup \{\mathtt{G}\} \qquad \mathrm{dom}(\Gamma, n : A) \stackrel{\text{def}}{=} \mathrm{dom}(\Gamma) \cup \{n\}$

$(\textsc{Env}\ \varnothing)\ \dfrac{-}{\varnothing \vdash \diamond} \qquad (\textsc{Env}\ n)\ \dfrac{n \notin \Gamma;\ \ \mathrm{fg}(A) \subseteq \mathrm{dom}(\Gamma)}{\Gamma, n : A \vdash \diamond} \qquad (\textsc{Env}\ \mathtt{G})\ \dfrac{\Gamma \vdash \diamond; \quad \mathtt{G} \notin \Gamma}{\Gamma, \mathtt{G} \vdash \diamond}$