Types for Evolving Communication in Safe Ambients

Francesca Levi

Dipartimento di Informatica e Scienze dell'Informazione
Università di Genova, Italy
levifran@disi.unige.it

Abstract. We extend the evolving types of [10] to full Safe Ambients following the lines of the single-threaded types of [12] for communication. Then, we introduce more flexible evolving types which permit to define ambients, where the type of the exchanged data may vary as the internal computation proceeds or after the ambient is opened.

1 Introduction

Mobile Ambients (MA) [5] has recently emerged as a core programming language for the Web and, at the same time, as a model for reasoning about properties of mobile processes. In contrast with previous formalisms for mobile processes such as the π-calculus [14], whose computational model is based on the notion of *communication*, the MA computational model is based on the notion of *movement*. An ambient, which may be thought of as a named location, is the unit of movement. Processes within the same ambient may exchange messages; ambients may be nested, so to form a hierarchical structure. The three primitives for movement allow: an ambient to enter another ambient, an ambient to exit another ambient, a process to dissolve an ambient boundary thus obtaining access to its contents. *Safe Ambients* (SA) [12] is a modification of MA, where a movement, in or out, or an ambient dissolution can take place only if both participants agree. This is achieved by means of coactions $\overline{\mathtt{in}}\,a$, $\overline{\mathtt{out}}\,a$ and $\overline{\mathtt{open}}\,a$, which complement the capabilities $\mathtt{in}\,a$, $\mathtt{out}\,a$ and $\mathtt{open}\,a$, respectively. This variation does not change the expressiveness of MA[1], yet makes it easier to write programs, to define static techniques [12, 3, 9, 8], and to formally prove their correctness, especially by using behavioural equivalences [12, 13]. The types in [12] introduce basic, immobile and single-threaded (ST) ambients. Basic and immobility types simply adapt to SA the types [6, 4] for MA; the ST types define ambients that, at any time, have at most one capability ready to fire. Both immobility and ST types guarantee the absence of grave interferences, which is very useful, for instance for proving the correctness of programs algebraically.

The ST types introduce an interesting feature: the type of an ambient can *evolve*, meaning that the type of its local process can be different from that of the

[1] There is an obvious encoding of MA into SA.

process unleashed, when the ambient is opened. The form of evolution allowed in [12] is, however, very restrictive. For instance, the following process cannot be typed with b immobile, although a, when opened inside b, unleashes only the immobile capability $\overline{\texttt{out}}\, b$

$$a[\,\texttt{in}\, b.\, \overline{\texttt{open}}\, a.\, \overline{\texttt{out}}\, b\,] \mid b[\,\overline{\texttt{in}}\, b.\, \texttt{open}\, a.\, Q\,]. \tag{1}$$

Ambient a can be typed only ST or basic (multi-threaded), and immobile ambients cannot open ST or basic ambients; they can indeed unleash capabilities of movement. Guan et al. [10] have modified the types of [12] for a variant of SA[2] without communication. They let the thread and mobility being orthogonal concepts and, consequently, achieve more flexible types, where both the mobility and the thread rights can evolve after opening. With this modification, the problem in (1) is solved, saying that a is a ST and mobile ambient which, when opened, liberates a ST and immobile process.

Another limitation of most of the types of MA [4, 7, 6, 15] and SA [12, 3, 9] is that the type of the messages exchanged inside an ambient is fixed during its lifetime and also after its dissolution. There are however cases in which the type of messages can *safely* vary as the computation proceeds. This happens, for instance, in the following processes (assuming proper types for M_1 and M_2 and $W_1 \neq W_2$)

$$a[\,\texttt{in}\, c.\, \langle M_1\rangle \mid (x : W_1)\overline{\texttt{open}}\, a.\, \langle M_2\rangle\,] \mid c[\,\overline{\texttt{in}}\, c.\, \texttt{open}\, a.\, (y : W_2)Q\,] \tag{2}$$

$$a[\,\langle M_1\rangle \mid (x : W_1)(\langle M_2\rangle \mid (y : W_2)Q)\,] \tag{3}$$

In (2) the local process of a has exchanges of type W_1 and the process unleashed, when a is opened inside c, has exchanges of type W_2. Similarly in (3) the type of the data exchanged inside a is initially W_1, and becomes W_2 after the reception of M_1.

One solution to deal with this form of *evolving communication* is to represent explicitly some causality information in the types, as suggested in [1, 2]. In this paper we propose an alternative approach which uses a form of linearity inspired by the ST types of [12]. We observe that the ST types offer control on *how many* abstractions and messages are present inside an ambient, and also on *when* they are consumed. This because the ST types let the thread move between the parallel processes local to an ambient by means of open and also by means of messages. For instance, both processes (2) and (3) can be typed with a and c ST, assuming that messages carry the thread and $W_1 = W_2$. In (2) the thread over a is first held by $\texttt{in}\, c.\, \langle M_1\rangle$, then it is passed to $(\overline{\texttt{open}}\, a.\, \langle M_2\rangle)\{{}^{M_1}\!/x\}$; the thread over c is first held by $\overline{\texttt{in}}\, c.\, \texttt{open}\, a.\, (y : W_2)Q$, then it is passed to the process $\langle M_2\rangle\{{}^{M_1}\!/x\}$ unleashed by a, and finally to $(y : W_2)Q\{{}^{M_2\{{}^{M_1}\!/x\}}\!/y\}$.

In cases (2) and (3), the movements of the thread, carried by messages, show that all the messages and abstractions of type W_1 have been consumed, when the message M_2 and the abstraction $(y : W_2)Q$ are ready to interact inside a or

[2] Where in coactions $\overline{\texttt{in}}\, n$ and $\overline{\texttt{out}}\, n$ the name n refers to the ambient that moves, and where $\overline{\texttt{open}}$ has no parameters.

inside c. In general, however, the conditions guaranteed by the ST types are too weak to deal with evolving communication. The problem is the limited control on the reception of messages, which carry the thread, as the following processes illustrate

$$a[\,(z:W_1)P \mid \mathtt{in}\,c.\,\overline{\mathtt{open}}\,a.\,\langle M_2\rangle\,] \mid c[\,\overline{\mathtt{in}}\,c.\,\mathtt{open}\,a.\,(y:W_2)Q\,] \tag{4}$$

$$a[\,\langle M_1\rangle \mid (z:W_1)P \mid (x:W_1)(\langle M_2\rangle \mid (y:W_2)Q)\,]. \tag{5}$$

Both processes can be typed with a and c ST similarly to processes (2) and (3). In (5) the thread over a is passed non-deterministically to either $(\langle M_2\rangle \mid (y:W_2)Q)\{M_1/x\}$ or to $P\{M_1/z\}$; in (4) the thread over a is held by $\mathtt{in}\,c.\,\overline{\mathtt{open}}\,a.\,\langle M_2\rangle$ until a is opened. The subtle difference is that processes (4) and (5) are unsafe, when $W_1 \neq W_2$, as the abstraction $(z:W_1)P$ may end up in parallel with the emission of M_2.

In Section 3 we extend the approach of [10] to full SA following the lines of [12] for communication. In this paper, we focus on the ST types; the inclusion of basic (multi-threaded) ambients is orthogonal and can be found in the complete version [11]. Then, in Section 4, we define new types for evolving communication (EC) by introducing, in ST ambients, a proper typing discipline for communication. This is obtained by imposing stronger conditions on the movements of the thread, carried by messages, according to the discussion above. With the EC types we can type ambients where the type of the exchanged data may vary when the internal computation proceeds, as in process (3), or after the ambient is opened, as in process (2).

2 Mobile Safe Ambients

We briefly recall the *Safe Ambients* ([12]) calculus. Let $\mathcal{N}$ be set of *names* (ranged over by $a, b, \ldots n, m$), $\mathcal{V}$ be set of *variables* (ranged over by $x, y, z, \ldots$) and $\mathcal{V}_R$ be a set of recursion variables (ranged over by $X, Y, Z, \ldots$). The syntax of (typed) SA processes is shown in Table 1. Standard syntactical conventions are used: trailing zeros are omitted, and parallel composition has the least syntactic precedence. Also, we may omit the type annotations in restrictions and abstractions whenever not important. The rules for the reduction relation of SA are shown in Table 2. Reductions are labelled with σ ranging over $\{\lambda, \mathtt{com}\}$ in order to better formulate subject reduction (Theorem 5.1). Rule `Struct` uses the auxiliary relation $\equiv$ of structural congruence to bring the participants of a potential interaction into contiguous positions; the definition of $\equiv$ is standard (we refer the reader to [11]).

3 Evolving Types for full SA

We adapt the evolving types to the full language by integrating the approach of [10] and the typing of communication in [12]. To retain the expressive power of

$M, N ::=$		(Expressions)	$P, Q ::=$		(Processes)
	$\mathtt{in}\, M$	(enter)		0	(inactivity)
	$\mathtt{out}\, M$	(exit)		$(\boldsymbol{\nu} a : A)\, P$	(typed restriction)
	$\mathtt{open}\, M$	(open)		$P \mid Q$	(parallel composition)
	$\overline{\mathtt{in}}\, M$	(allow enter)		$\mathtt{rec}\, X.\, P$	(replication)
	$\overline{\mathtt{out}}\, M$	(allow exit)		$a[\, P\,]$	(ambient)
	$\overline{\mathtt{open}}\, M$	(allow open)		$M.\, P$	(prefix)
	ϵ	(empty path)		$\langle M \rangle$	(message)
	$M.\, N$	(path)		$(x : W)P$	(typed abstraction)
	x	(variable)		X	(recursion variable)
	a	(name)			

Table 1. Syntax of Expressions and Processes.

$$\mathtt{In}:\ a[\,\mathtt{in}\, b.\, P_1 \mid P_2\,] \mid b[\,\overline{\mathtt{in}}\, b.\, Q_1 \mid Q_2\,] \longrightarrow_\lambda b[\, a[\, P_1 \mid P_2\,] \mid Q_1 \mid Q_2\,]$$

$$\mathtt{Out}:\ b[\, a[\,\mathtt{out}\, b.\, P_1 \mid P_2\,] \mid \overline{\mathtt{out}}\, b.\, Q_1 \mid Q_2\,] \longrightarrow_\lambda a[\, P_1 \mid P_2\,] \mid b[\, Q_1 \mid Q_2\,]$$

$$\mathtt{Open}:\ \mathtt{open}\, a.\, P \mid a[\,\overline{\mathtt{open}}\, a.\, Q_1 \mid Q_2\,] \longrightarrow_\lambda P \mid Q_1 \mid Q_2$$

$$\mathtt{Com}:\ \langle M \rangle \mid (x : W)P \longrightarrow_{\mathtt{com}} P\{^{M}\!/_{x}\}$$

$$\mathtt{Rec}:\ \mathtt{rec}\, X.\, P \longrightarrow_\lambda P\{^{\mathtt{rec}\, X.\, P}\!/_{X}\} \qquad \mathtt{Res}:\ \frac{P \longrightarrow_\sigma P'}{(\boldsymbol{\nu} n : A)\, P \longrightarrow_\sigma (\boldsymbol{\nu} n : A)\, P'}$$

$$\mathtt{Par}:\ \frac{P \longrightarrow_\sigma P'}{P \mid Q \longrightarrow_\sigma P' \mid Q} \qquad \mathtt{Amb}:\ \frac{P \longrightarrow_\sigma P'}{a[\, P\,] \longrightarrow_\lambda a[\, P'\,]}$$

$$\mathtt{Struct}:\ \frac{P \equiv P' \quad P' \longrightarrow_\sigma P'' \quad P'' \equiv P'''}{P \longrightarrow_\sigma P'''}$$

Table 2. Reduction Rules for SA.

both [10] and [12], we introduce two main modifications: (i) we slightly modify the rule of parallel composition of [10]; (ii) we extend the syntax of the exchange types of [12] by letting messages carry the mobility annotation as well as the thread annotation.

I, J	$:= 0 \mid 1$	(**Thread Annotations**)
X, Y	$:= \uparrow \mid \ominus$	(**Mobility Annotations**)
T	$:= {}^{I,X}W \mid \mathtt{Shh}$	(**Types of Messages**)
E, F	$:= T^{I,X}$	(**Pre-types**)
S	$:= E \mid E; J, Y$	(**Simple and Evolving Pre-types**)
A	$:= \mathtt{Amb}[S]$	(**Types of Ambients**)
W	$:= A \mid C$	(**Types of Expressions**)
Z	$:= \mathtt{Proc}[S] \mid \top$	(**Types of Processes**)
C, D	$:= \diamond \mid Z.C \mid (C \mid Z)$	(**Capability Contexts**)

Table 3. Type Grammar.

The type grammar is shown in Table 3. The thread annotation can take value 0 (no thread) or 1 (thread); the mobility annotation can take value $\ominus$ (immobile) or $\uparrow$ (mobile). The intuitive meaning of the ambient types is the following:

- $\mathtt{Amb}[T^{I,X}]$ is the type of a *simple* (S) ambient, that cannot be opened, and inside which messages of type T can be exchanged; it has thread I and mobility X;
- $\mathtt{Amb}[T^{I,X}; J, Y]$ is the type of an *evolving* (E) ambient inside which messages of type T can be exchanged; it has thread I and mobility X before being opened; when opened, it unleashes a process with thread J, mobility Y, and exchanges of type T. We call $T^{I,X}$ its *current* pre-type and $T^{J,Y}$ its *future* pre-type.

Similarly, $\mathtt{Proc}[T^{I,X}]$ and $\mathtt{Proc}[T^{I,X}; J, Y]$ are the types of the processes local to a simple and evolving ambient respectively. The type of messages T can be either $\mathtt{Shh}$ or ${}^{I,X}W$: $\mathtt{Shh}$ means that there is no communication; ${}^{I,X}W$ means that data of type W are exchanged, and that I and X are the thread and mobility annotations carried by messages. The exchanged data can be either ambients or capabilities, which are represented as in [10, 1] by contexts (with hole $\diamond$). The use of contexts in place of standard capability types and of the special process type $\top$, representing an error, simplifies the typing rules.

The typing rules use sub-typing and two auxiliary operators, which realize the sequential and parallel composition of process types. They are needed to derive a process type corresponding to $C[Z]$, when the hole of a context C is filled by a process type Z (specifically in the typing rules for prefixing and parallel composition of Table 6).

$$
\begin{array}{ll}
\ominus \sqsubseteq \uparrow & \\
0 \sqsubseteq 1 \sqsubseteq \omega & \\
\mathtt{Shh} \sqsubseteq^{I,X} W & \\
T_1^{I_1,X_1} \sqsubseteq T_2^{I_2,X_2} & \text{if } I_1 \sqsubseteq I_2, X_1 \sqsubseteq X_2, T_1 \sqsubseteq T_2 \\
E_1; J_1, Y_1 \sqsubseteq E_2; J_2, Y_2 & \text{if } E_1 \sqsubseteq E_2, J_1 \sqsubseteq J_2, Y_1 \sqsubseteq Y_2 \\
\mathtt{Proc}[S_1] \sqsubseteq \mathtt{Proc}[S_2] & \text{if } S_1 \sqsubseteq S_2
\end{array}
$$

Table 4. Sub-typing.

The relations of sub-typing, over any syntactical category, are defined as the minimal reflexive and transitive relations which satisfy the rules of Table 4. Table 5 shows the operators of parallel and sequential composition. For reasons of space, we omit the symmetric cases of $|$ and we assume that the cases, which are not listed, are $\top$. We also use $x \sqcup y$ for the least upper bound (l.u.b.) of x and y, and we let $x \sqcup y = \top$ if the least upper bound does not exist. We assume that the effect of $\top$ is properly propagated to pre-types and process types.

Both the parallel and the sequential composition of two process types are obtained by the related composition of their pre-types.

Parallel Composition Two S pre-types are combined by taking the l.u.b. of the message types and of the mobility annotations, and a thread annotation which is: ST only if *at most* one is ST; in particular it is no thread only if both are no thread. A E pre-type and a S pre-type are combined by composing the S pre-type with the current pre-type. In [10] a different rule is used: both the current and the future pre-type are updated reflecting the idea that any process executable in parallel with a capability $\overline{\mathtt{open}}$ may be executed either *before* or *after* the opening. Our rule is based on the following property of ST ambients: all processes executable in parallel with a capability $\overline{\mathtt{open}}$ have necessarily thread 0 (and are thus also immobile) so that they cannot affect the future pre-type [3]. The parallel composition of two E pre-types is not permitted as there is no control on which $\overline{\mathtt{open}}$ may be executed first (as in [10]).

Sequential Composition Two S pre-types are combined by taking the l.u.b. of both the types of messages, and the thread and mobility annotations. The sequential composition of a S and a E pre-type depends on which type occurs first. If the S pre-type precedes the E pre-type, it means that the simple process is executed *before* opening. Hence, we modify only the current pre-type. If the E pre-type precedes the S pre-type, it means that the simple process is executed *after* opening. We therefore require the simple pre-type

[3] It is worth mentioning that for the calculus without communication the two rules are equivalent; our rule is however more precise in presence of abstractions and messages (see [11] for an example).

to be consistent with the future pre-type.

$$0 \mid 0 = 0, 0 \mid 1 = 1, 1 \mid 0 = 1$$

$$T_1^{I_1,X_1} \mid T_2^{I_2,X_2} = (T_1 \sqcup T_2)^{I_1 \mid I_2, X_1 \sqcup X_2}$$
$$T_1^{I_1,X_1} . T_2^{I_2,X_2} = T_1^{I_1,X_1} \sqcup T_2^{I_2,X_2}$$

$$E_1 \mid (E_2; J, Y) = (E_1 \mid E_2); J, Y$$
$$E_1 . (E_2; J, Y) = (E_1 . E_2); J, Y$$
$$(T_1^{I_1,X_1}; J, Y) . T_2^{I_2,X_2} = \begin{cases} T_1^{I_1,X_1}; I_2, X_2 & \text{if } T_2^{I_2,X_2} \sqsubseteq T_1^{J,Y} \\ \top & \text{otherwise} \end{cases}$$

$$\mathtt{Proc}\,[S_1] . \mathtt{Proc}\,[S_2] = \mathtt{Proc}\,[S_1 . S_2]$$
$$\mathtt{Proc}\,[S_1] \mid \mathtt{Proc}\,[S_2] = \mathtt{Proc}\,[S_1 \mid S_2]$$

Table 5. Parallel and sequential composition of types.

The typing rules are shown in Table 6. A *type environment* is a finite set of assignments of types to names, variables and recursion variables.

The typing rules for expressions are similar to those in [10], and assign a proper context to any capability. These rules should be read having in mind that the hole in the context will be filled by the continuation of the capability (see rule (pref)). For capabilities the contexts are of the form $Z . C$, where Z is a process type, expressing the contribution of the capability, and C is a context, that permits to take into account the continuation. Rules (inout) and (co-inout) are analogous with the only difference that the former is mobile and the latter is immobile. In rule (open) the context shows that the process type of the continuation ends up in parallel with the future type of the opened ambient. In rule (co-open) the context shows that the capability $\overline{\mathtt{open}}\, M$ has an evolving type, dictated by the type of the evolving ambient M where it can be exercised. Rule (path) realizes context substitution.

The typing rules for processes extend those of [10] to communication. Rule (amb) says that any type can be assigned to an ambient, provided that its local process agrees with its simple or evolving pre-type. We use $S_1 \rhd S_2$ for $S_1 = S_2$. This relation is exploited to compact similar rules together and is extended in Section 4. Rule (pref) says that the context hole has to be filled with the type of the continuation. Rule (par) shows that the two types have to be combined by means of $|$.

The rules for communication are an adaptation of those of [12]. Rules (abs)

$$\frac{\Gamma \vdash N : A \qquad M \in \{\overline{\mathtt{in}}\, N, \overline{\mathtt{out}}\, N\}}{\Gamma \vdash M : \mathtt{Proc}\,[\mathtt{Shh}^{1,\ominus}].\,\diamond}\ \text{co-inout}$$

$$\frac{\Gamma \vdash M : \mathtt{Amb}\,[T^{I,X}; J, Y]}{\Gamma \vdash \mathtt{open}\, M : \mathtt{Proc}\,[\mathtt{Shh}^{1,\ominus}].\,(\diamond \mid \mathtt{Proc}\,[T^{J,Y}])}\ \text{open}$$

$$\frac{}{\Gamma \vdash \epsilon : \diamond}\ \epsilon \qquad \frac{\Gamma \vdash N : A \qquad M \in \{\mathtt{in}\, N, \mathtt{out}\, N\}}{\Gamma \vdash M : \mathtt{Proc}\,[\mathtt{Shh}^{1,\uparrow}].\,\diamond}\ \text{inout}$$

$$\frac{\Gamma \vdash M : \mathtt{Amb}\,[T^{I,X}; J, Y]}{\Gamma \vdash \overline{\mathtt{open}}\, M : \mathtt{Proc}\,[T^{1,\ominus}; J, Y].\,\diamond}\ \text{co-open} \qquad \frac{\Gamma \vdash M : C_1 \qquad \Gamma \vdash N : C_2}{\Gamma \vdash M.N : C_1\,[C_2]}\ \text{path}$$

$$\frac{\Gamma \vdash M : \mathtt{Amb}\,[S_1] \qquad \Gamma \vdash P : \mathtt{Proc}\,[S_2] \qquad S_1 \rhd S_2}{\Gamma \vdash M[\,P\,] : Z}\ \text{amb}$$

$$\frac{\Gamma, x : W \vdash P : \mathtt{Proc}\,[({}^{I,X}W)^{I,X}]}{\Gamma \vdash (x : W)P : \mathtt{Proc}\,[({}^{I,X}W)^{0,\ominus}]}\ \text{abs}$$

$$\frac{\Gamma, x : W \vdash P : \mathtt{Proc}\,[({}^{I,X}W)^{I,X}; J, Y]}{\Gamma \vdash (x : W)P : \mathtt{Proc}\,[({}^{I,X}W)^{0,\ominus}; J, Y]}\ \text{e-abs}$$

$$\frac{\Gamma \vdash M : C \qquad \Gamma \vdash P : Z}{\Gamma \vdash M.P : C[Z]}\ \text{pref} \qquad \frac{\Gamma \vdash P_1 : Z_1 \qquad \Gamma \vdash P_2 : Z_2}{\Gamma \vdash P_1 \mid P_2 : Z_1 \mid Z_2}\ \text{par}$$

$$\frac{\Gamma \vdash P : Z_1 \qquad Z_1 \sqsubseteq Z_2}{\Gamma \vdash P : Z_2}\ \text{sub} \qquad \frac{\Gamma \vdash M : W}{\Gamma \vdash \langle M \rangle : \mathtt{Proc}\,[({}^{I,X}W)^{I,X}]}\ \text{msg}$$

$$\frac{}{\Gamma \vdash \mathbf{0} : Z}\ \text{nil} \qquad \frac{\Gamma, X : Z \vdash P : Z}{\Gamma \vdash \mathtt{rec}\, X.\, P : Z}\ \text{rec}$$

$$\frac{\Gamma, n : A \vdash P : Z}{\Gamma \vdash (\boldsymbol{\nu} n : A)\, P : Z}\ \text{res} \qquad \frac{\Gamma(X) = Z}{\Gamma \vdash X : Z}\ \text{varProc}$$

Table 6. Typing Rules.

and (e-abs) say that an abstraction, simple or evolving, has no thread and is immobile; the body of the abstraction dictates the thread and mobility annotations carried by messages. Rule (msg) shows that messages acquire the thread and mobility annotations from the message type. These typing rules guarantee that in a ST ambient, when messages carry the thread, at any moment, at most one capability or message is ready to fire. Therefore, there can be non-determinism on reception of messages but not on emission. This typing discipline for communication is too weak, when considering evolving communication, as they cannot distinguish the processes (2) and (3), (4) and (5) discussed the Introduction.

The process (1) discussed in the Introduction can instead be typed. Assume that $\Gamma \vdash a : \mathtt{Amb}[\mathtt{Shh}^{1,\uparrow}; 1, \ominus]$, $\Gamma \vdash b : \mathtt{Amb}[\mathtt{Shh}^{1,\ominus}]$, and $\Gamma \vdash Q : \mathtt{Proc}[\mathtt{Shh}^{0,\ominus}]$. For any type $Z \neq \top$ we have $\Gamma \vdash a[\,\mathtt{in}\, b.\, \overline{\mathtt{open}}\, a.\, \overline{\mathtt{out}}\, b\,] \mid b[\,\overline{\mathtt{in}}\, b.\, \mathtt{open}\, a.\, Q\,] : Z$.

4 Evolving Communication Types

$$
\begin{array}{lll}
{}^{\circ}T & := {}^{\star,X}W & \\
{}^{\star}T & := {}^{\star,X}W \mid \mathtt{Shh} & (\star - \textbf{Messages}) \\
{}^{\circ}L & := {}^{\circ}T \mid {}^{\circ}L_1 \cdot {}^{\circ}L_2 \mid \mathtt{rec}\,{}^{\circ}L & (\textbf{List of Messages}) \\
L & := [{}^{\circ}L] \mid [\mathtt{Shh}] & (\textbf{EC Messages}) \\
{}^{\circ}E, {}^{\circ}F & := {}^{\star}T^{I,X} \mid L^{I,X} & (\star \textbf{ and EC Pre-types}) \\
E, F & := {}^{\circ}E & \\
S & := {}^{\circ}E; F & (\textbf{EEC Pre-types})
\end{array}
$$

Table 7. Type Grammar for EC types

To model evolving communication we add the symbols and productions of Table 7 to the type grammar of Table 3. We introduce a new thread annotation, called $\star$, which is carried by messages similarly to the thread of Section 3. We also introduce two types of messages: (i) $\star$-messages, where ${}^{\star,X}W$ shows that data of type W are exchanged, and that messages carry the thread annotation $\star$ and mobility annotation X; (ii) EC messages, where $[{}^{\circ}L]$ models evolving communication as a (possibly recursive) list of message types of the form ${}^{\star,X}W$. These types permit to better characterize who can pass and who can receive the thread $\star$ carried by messages. In particular, ${}^{\star,X}W$ is the type of a process which can pass the thread; while $[{}^{\circ}L]$ is the type of a process which can receive the thread. They are used precisely to impose stronger conditions on the way $\star$ can be passed, when communication evolve.

In addition, we extend the productions of pre-types S by introducing ${}^{\circ}E$ and ${}^{\circ}E; F$. The resulting new ambient types have the following intuitive meaning. $\mathtt{Amb}[{}^{\circ}E; F]$ is the type of an evolving ambient with evolving communication, where both the thread and mobility annotations and the type of communication can change after opening; ${}^\circ E$ and F are its current and future pre-type. $\texttt{Amb}\,[{}^\circ E]$ is the type of a S ambient with evolving communication, whose local processes behave according to ${}^\circ E$. Similarly for the related process types.

We introduce a relation of equivalence over lists of messages ${}^\circ L$, which rearranges lists using the associative property and realizing the fold/unfold of recursion. We define $\cong$ as the minimal transitive, reflexive and symmetric relation which satisfies the rules of Table 8. To compact similar rules together we also use

$$\texttt{head}({}^\circ L) = \begin{cases} {}^\circ T & \text{if } {}^\circ L ={}^\circ T \cdot^\circ L_1 \text{ or } {}^\circ L ={}^\circ T \\ \text{undefined} & \text{otherwise} \end{cases}$$

$({}^\circ T \cdot^\circ L_1) \cdot^\circ L_2 \cong^\circ T \cdot ({}^\circ L_1 \cdot^\circ L_2)$	
$\texttt{rec}\,{}^\circ L \cong^\circ L \cdot (\texttt{rec}\,{}^\circ L)$	
${}^\circ L_1 \cdot^\circ L_2 \cong^\circ L_3 \cdot^\circ L_2$	if ${}^\circ L_1 \cong^\circ L_3$
${}^\circ L_1 \cdot^\circ L_2 \cong^\circ L_1 \cdot^\circ L_3$	if ${}^\circ L_2 \cong^\circ L_3$
$\texttt{rec}\,{}^\circ L_1 \cong \texttt{rec}\,{}^\circ L_2$	if ${}^\circ L_1 \cong^\circ L_2$
$\texttt{Shh} \sqsubseteq^{\star,X} W$	
$\texttt{Shh} \sqsubseteq [\texttt{Shh}]$	
$[\texttt{Shh}] \sqsubseteq [L]$	
${}^\circ L_1 \sqsubseteq^\circ L_1 \cdot^\circ L_2$	
${}^\circ L_1 \sqsubseteq^\circ L_2$	if ${}^\circ L_1 \cong^\circ L_2$
$[{}^\circ L_1] \sqsubseteq [{}^\circ L_2]$	if ${}^\circ L_1 \sqsubseteq^\circ L_2$
$L_1^{I_1,X_1} \sqsubseteq L_2^{I_2,X_2}$	if $L_1 \sqsubseteq L_2, I_1 \sqsubseteq I_2, X_1 \sqsubseteq X_2$
${}^\star T_1{}^{I_1,X_1} \sqsubseteq {}^\star T_2{}^{I_2,X_2}$	if ${}^\star T_1 \sqsubseteq^\star T_2, I_1 \sqsubseteq I_2, X_1 \sqsubseteq X_2$
${}^\circ E_1; F_1 \sqsubseteq^\circ E_2; F_2$	if ${}^\circ E_1 \sqsubseteq^\circ E_2, F_1 \sqsubseteq F_2$.
${}^\circ T \cdot^\circ L \rhd^\circ L$	
${}^\circ L_1 \rhd^\circ L_2'$	if ${}^\circ L_1 \cong^\circ L_2, {}^\circ L_2 \rhd^\circ L_2'$
$[{}^\circ L_1]^{I,X} \rhd [{}^\circ L_2]^{I,X}$	if ${}^\circ L_1 \rhd^\circ L_2$
${}^\circ E_1; F \rhd^\circ E_2; F$	if ${}^\circ E_1 \rhd^\circ E_2$

Table 8. Additional relations over EC types

All the notions of Section 3 (sub-typing, least upper bound, composition of types and typing rules) are valid also for the new values. We present the additional cases for sub-typing and for the operators of parallel and sequential composition in Tables 8 and 9. The rules are similar to those of Table 5. As before, the cases which are not listed correspond to $\top$ and the symmetric cases of $|$ are omitted. The main difference is due to the parallel composition of the new message types $\star$-Messages and EC messages. In particular, we permit both the sequential and parallel composition of a EC message type L only with a

$\star$-message type ${}^\star T$, provided that ${}^\star T$ agrees with the first type of the list L. We forbid the parallel composition of two evolving message types L_1 and L_2, because there is no control on when they evolve. In the sequential composition we rule out all the other cases, as they cannot arise in the contexts modeling expressions [4].

$$
\begin{array}{l}
{}^\star T_1{}^{I_1,X_1} \mid {}^\star T_2{}^{I_2,X_2} = ({}^\star T_1 \sqcup^\star T_2)^{I_1|I_2, X_1 \sqcup X_2} \\
{}^\star T^{I_1,X_1} \mid L^{I_2,X_2} = \begin{cases} L^{I_1|I_2,X_1\sqcup X_2} & \text{if } {}^\star T \sqsubseteq \mathtt{head}({}^\circ L), L = [{}^\circ L] \\ L^{I_1|I_2,X_1\sqcup X_2} & \text{if } {}^\star T = \mathtt{Shh}, L = [\mathtt{Shh}] \\ \top & \text{otherwise} \end{cases}
\end{array}
$$

$$
\begin{array}{l}
{}^\star T_1{}^{I_1,X_1} . {}^\star T_2{}^{I_2,X_2} = {}^\star T_1{}^{I_1,X_1} \sqcup {}^\star T_2{}^{I_2,X_2} \\
{}^\star T^{I_1,X_1} . L^{I_2,X_2} = \begin{cases} L^{I_1\sqcup I_2,X_1\sqcup X_2} & \text{if } {}^\star T \sqsubseteq \mathtt{head}({}^\circ L), L = [{}^\circ L] \\ L^{I_1\sqcup I_2,X_1\sqcup X_2} & \text{if } {}^\star T = \mathtt{Shh}, L = [\mathtt{Shh}] \\ \top & \text{otherwise} \end{cases}
\end{array}
$$

$$
\begin{array}{l}
{}^\circ E_1 \mid ({}^\circ E_2; F) = ({}^\circ E_1 \mid^\circ E_2); F \\
{}^\circ E_1 . ({}^\circ E_2; F) = ({}^\circ E_1 .^\circ E_2); F \\
({}^\circ E_1; F). E = \begin{cases} {}^\circ E_1; E & \text{if } E \sqsubseteq F \\ \top & \text{otherwise} \end{cases}
\end{array}
$$

Table 9. Sequential and parallel composition of EC types

The additional typing rules for the EC types are shown in Table 10. They include new rules only for the capabilities of opening, for messages and abstractions. For the other cases the rules of Table 6 apply. To adapt also the applicability of rule (amb) we need however to extend the relation $\rhd$. This is defined over pre-types S as the minimal reflexive and transitive relation which satisfies the rules of Table 8. The relation $\rhd$ reflects the idea that, if the type of exchanges of the local process is EC, then it may vary as the computation proceeds (as specified by Theorem 5.1).

Rules (ec-open) and (ec-co-open) are similar to the corresponding rules (open) and (co-open) for E ambients. Rule (ec-co-open) shows that the type of communication can vary *after* $\overline{\mathtt{open}}\, M$, provided that the current type of messages is EC (as dictated by $[\mathtt{Shh}]$). The typing rules for communication are designed to let the type of communication vary *after* a message is received.

1. Rule (ec-msg) says that, when messages carry the thread $\star$, a message can be typed only as a S process with a $\star$-message type; it has the thread and

[4] There is no typing rule for expressions (in both Tables 6 and 10) where the context is $Z.C$ with $Z = \mathtt{Proc}\,[L^{I,X}]$.

the mobility annotation carried by messages.

2. Rules (es-abs1) and (es-abs2) say that an abstraction can be typed as a S process only with a EC message type, which is obtained by eventually appending the type of the input to the list; it has no thread and is immobile. Rules (ec-abs1) and (ec-abs2) are defined similarly for EC types.

We now discuss the typing of processes (2) and (3) shown in the Introduction. We illustrate in particular how the complementary types of communication ($\star$-Messages and EC Messages) impose stronger conditions on the way the thread $\star$ can be passed, which guarantee safe evolving communication.

Assume that $\Gamma \vdash M_1 : W_1$, $\Gamma, x : W_1 \vdash M_2 : W_2$, $\Gamma, x : W_1, y : W_2 \vdash Q : \mathtt{Proc}\,[{}^\circ E]$, $\Gamma \vdash a : \mathtt{Amb}\,[L^{1,\ominus}]$, where $L = [{}^\circ T_1 \cdot {}^\circ T_2]$, ${}^\circ E = [{}^\circ T_2]^{1,\ominus}$ and ${}^\circ T_i =^{\star,\ominus} W_i$ for $i \in \{1,2\}$ with $W_1 \neq W_2$. For any type $Z \neq \top$ we have $\Gamma \vdash a[\,\langle M_1\rangle \mid (x : W_1)(\langle M_2\rangle \mid (y : W_2)Q)\,] : Z$. The most important observation is that we can give the following type to the local process of a,

$$\mathtt{Proc}\,[{}^\circ {T_1}^{1,\ominus}] \mid \mathtt{Proc}\,[[{}^\circ T_1 \cdot {}^\circ T_2]^{0,\ominus}] = \mathtt{Proc}\,[[{}^\circ T_1 \cdot {}^\circ T_2]^{1,\ominus}]$$

where $\mathtt{Proc}\,[{}^\circ {T_1}^{1,\ominus}]$ is the type of the message $\langle M_1\rangle$ and $\mathtt{Proc}\,[[{}^\circ T_1 \cdot {}^\circ T_2]^{0,\ominus}]$ is the type of the abstraction $(x : W_1)(\langle M_2\rangle \mid (y : W_2)Q)$. This shows that the thread over a moves from message $\langle M_1\rangle$ to the continuation of the abstraction $(x : W_1)(\langle M_2\rangle \mid (y : W_2)Q)$.

By contrast the process (5) $a[\,\langle M_1\rangle \mid (z : W_1)P \mid (x : W_1)(\langle M_2\rangle \mid (y : W_2)Q)\,]$ cannot be typed when $W_1 \neq W_2$. There is no type for $(z : W_1)P$ which can be combined in parallel with $\mathtt{Proc}\,[L^{1,\ominus}]$ because: (i) abstractions can be given *only* EC message types (see the rules of Table 10); two EC message types cannot be combined in parallel (see Table 9). In other words $\star$ can be passed to *at most* one abstraction.

Assume that $\Gamma \vdash M_1 : W_1$, $\Gamma, x : W_1 \vdash M_2 : W_2$, $\Gamma \vdash c : \mathtt{Amb}\,[E]$, and $\Gamma, y : W_2 \vdash Q : \mathtt{Proc}\,[E]$, $\Gamma \vdash a : \mathtt{Amb}\,[[{}^\circ T]^{1,\uparrow}; E]$, where ${}^\circ T =^{\star,\ominus} W_1$, and $E = T^{I,X}$, with $I \sqsubseteq 1$ and $T =^{I,X} W_2$. For any type $Z \neq \top$ we have (also when $W_1 \neq W_2$) $\Gamma \vdash a[\,\mathtt{in}\,c.\,\langle M_1\rangle \mid (x : W_1)\overline{\mathtt{open}}\,a.\,\langle M_2\rangle\,] \mid c[\,\overline{\mathtt{in}}\,c.\,\overline{\mathtt{open}}\,a.\,(y : W_2)Q\,] : Z$.

By contrast, the process (4)

$$a[\,(z : W_1)P \mid \mathtt{in}\,c.\,\overline{\mathtt{open}}\,a.\,\langle M_2\rangle\,] \mid c[\,\overline{\mathtt{in}}\,c.\,\mathtt{open}\,a.\,(y : W_2)Q\,]$$

cannot be typed when $W_1 \neq W_2$. Process $\mathtt{in}\,c.\,\overline{\mathtt{open}}\,a.\,\langle M_2\rangle$ has a EC process type $\mathtt{Proc}\,[[\mathtt{Shh}]^{1,\uparrow}; E]$ showing that it has the thread and that no other parallel process can receive the thread carried by messages. As above, it is not possible to type the abstraction $(z : W_1)P$.

5 Properties of the type system

We present the fundamental properties of types [5]. Theorem 5.1 states subject reduction. Notice that, when the type of communication is EC and a reduction `com` is performed, the type can evolve as modelled by the relation $\rhd$.

[5] The proofs can be found in [11].

$$\frac{\Gamma \vdash M \;:\texttt{Amb}\,[{}^{\circ}E; F]}{\Gamma \vdash \overline{\texttt{open}}\,M \;:\texttt{Proc}\,[[\texttt{Shh}]^{1,\ominus}; F].\diamond}\ \text{ec-co-open}$$

$$\frac{\Gamma \vdash M \;:\texttt{Amb}\,[{}^{\circ}E; F]}{\Gamma \vdash \texttt{open}\,M \;:\texttt{Proc}\,[\texttt{Shh}^{1,\ominus}]\,.\,(\diamond \mid \texttt{Proc}\,[F])}\ \text{ec-open}$$

$$\frac{\Gamma, x : W_1 \vdash P \;:\texttt{Proc}\,[[{}^{\circ}L]^{1,X}; F] \qquad \texttt{head}({}^{\circ}L) \neq^{\star,X} W_1}{\Gamma \vdash (x : W_1)P \;:\texttt{Proc}\,[[({}^{\star,X}W_1)\cdot{}^{\circ} L]^{0,\ominus}; F]}\ \text{ec-abs1}$$

$$\frac{\Gamma, x : W_1 \vdash P \;:\texttt{Proc}\,[[{}^{\circ}L]^{1,X}] \qquad \texttt{head}({}^{\circ}L) \neq^{\star,X} W_1}{\Gamma \vdash (x : W_1)P \;:\texttt{Proc}\,[[({}^{\star,X}W_1)\cdot{}^{\circ} L]^{0,\ominus}]}\ \text{es-abs1}$$

$$\frac{\Gamma, x : W_1 \vdash P \;:\texttt{Proc}\,[[{}^{\circ}L]^{1,X}; F] \qquad \texttt{head}({}^{\circ}L) =^{\star,X} W_1}{\Gamma \vdash (x : W_1)P \;:\texttt{Proc}\,[[{}^{\circ}L]^{0,\ominus}; F]}\ \text{ec-abs2}$$

$$\frac{\Gamma, x : W_1 \vdash P \;:\texttt{Proc}\,[[{}^{\circ}L]^{1,X}] \qquad \texttt{head}({}^{\circ}L) =^{\star,X} W_1}{\Gamma \vdash (x : W_1)P \;:\texttt{Proc}\,[[{}^{\circ}L]^{0,\ominus}]}\ \text{es-abs2}$$

$$\frac{\Gamma \vdash M \;: W}{\Gamma \vdash \langle M\rangle \;:\texttt{Proc}\,[({}^{\star,X}W)^{1,X}]}\ \text{ec-msg}$$

Table 10. Additional Typing Rules for EC types.

Theorem 5.1 (Subject Reduction). *Let* $\Gamma \vdash P : \mathtt{Proc}[S]$.

1. *Suppose that* $S = T^{I,X}$ *or* $S = T^{I,X}; Y, J$. *If* $P \longrightarrow_\sigma P'$ *then also* $\Gamma \vdash P' : \mathtt{Proc}[S]$.
2. *Suppose that* $S =^\circ E$ *or* $S =^\circ E; J, Y$ *or* $S =^\circ E; F$. *If* $P \longrightarrow_\sigma P'$ *with* $\sigma \neq \mathtt{com}$, *then also* $\Gamma \vdash P' : \mathtt{Proc}[S]$. *If* $P \longrightarrow_{\mathtt{com}} P'$, *then* $\Gamma \vdash P' : \mathtt{Proc}[S']$ *where* $S \rhd S'$.

The types satisfy the same properties of the ST and immobility types in [12]. Theorem 5.2 shows that: (i) in a ST ambient, at any time, there is at most one capability ready to fire; (ii) in an immobile ambient, at any time, there is no capability of the form $\mathtt{in}\,a$ or $\mathtt{out}\,a$ ready to fire. Property (i) guarantees that a single-threaded ambient is willing to engage, at any time, in at most one interaction with internal or external ambients. This implies the absence of all grave interferences which is very useful, for instance for proving the correctness of programs algebraically [12].

We say that an occurrence of an expression M in a process is *unguarded* if it does appear under a prefix, an abstraction or an ambient; likewise for unguarded messages and abstractions. Also we use $\mathcal{T}$ to range over types of messages.

Theorem 5.2 (Single-Threaded and Immobile Ambients). *Suppose that* $\Gamma \vdash P : \mathtt{Proc}[S]$ *where* $S = \mathcal{T}^{I,X}$, $S = \mathcal{T}^{I,X}; J, Y$ *or* $S = \mathcal{T}^{I,X}; F$. *If* $I = 1$ *then* P *has at least one unguarded capability. Also, if* $X = \ominus$ *then* P *has no unguarded capabilities of the form* $\mathtt{in}\,a$ *and* $\mathtt{out}\,a$.

6 Conclusions

The types of Section 3 extend the approach of [10] to full SA following the lines of [12] for communication. The complete extension including also basic (multi-threaded) ambients can be found in [11]. The idea of studying types for evolving communication is not new [1, 2]. The type system of [1, 2] assigns to processes behaviours representing explicitly the relationship (sequential or parallel) between inputs and outputs. This approach is more general and powerful than ours. The price to pay is that it is more complex: type-checking for instance is exponential. In fact, the sub-typing relation is defined relative to the underlying trace semantics and requires to consider for the parallel composition of behaviours all traces which can be formed by arbitrary interleavings. Our types follow a different approach; they are defined as a natural extension of the ST types and capture the evolution both of thread and mobility rights and of communication without using any causality information. As a consequence, type-checking is polynomial. The intuition is that two EC communication types cannot be combined in parallel (see Table 9) so that there is no need to merge two EC communication types (lists of messages). Moreover, the SA processes corresponding to the MA examples of orderly-communication shown in [2] can be typed with our types. They include the processes (2) and (3) discussed in the Introduction and a protocol for routing packets that is similar. Therefore, we believe that our types provide a good compromise between accuracy and complexity.

Acknowledgments This work has been partially supported by the project *Mefisto*.

References

1. Torben Amtoft and Assaf J. Kfoury and Santiago M. Pericas-Geertsen. What are Polymorphically-Typed Ambients? In *Proc. ESOP'01*, volume 2028 of *Lecture Notes in Computer Science*, pages 206–220. Springer Verlag, 2001.
2. Torben Amtoft and Assaf J. Kfoury and Santiago M. Pericas-Geertsen. Orderly Communication in the Ambient Calculus. To appear in *Computer Languages*.
3. M. Bugliesi and G. Castagna. Secure safe ambients. In *Proc. POPL '01*, pages 222-235. ACM Press, 2001.
4. L. Cardelli, G. Ghelli, and A.D. Gordon. Mobility types for mobile ambients. In *Proc. ICALP'99*, volume 1644 of *Lecture Notes in Computer Science*, pages 230–239. Springer Verlag, 1999.
5. L. Cardelli and A.D. Gordon. Mobile ambients. In *Proc. FoSSaCS '98*, volume 1378 of *Lecture Notes in Computer Science*, pages 140–155. Springer Verlag, 1998.
6. L. Cardelli and A.D. Gordon. Types for mobile ambients. In *Proc. POPL'99*, pages 79–92. ACM Press, 1999.
7. L. Cardelli, G. Ghelli, and A.D. Gordon. Types for ambient calculus. To apper in *Information and Computation*.
8. P. Degano, F. Levi and C. Bodei. Safe Ambients: Control Flow Analysis and Security. In *Proc. ASIAN '00*, volume 1961 of *Lecture Notes in Computer Science*, pages 199–214. Springer Verlag, 2000.
9. M. Dezani-Ciancaglini and I. Salvo. Security Types for Mobile Safe Ambients. In *Proc. ASIAN '00*, volume 1961 of *Lecture Notes in Computer Science*, pages 215–236. Springer Verlag, 2000.
10. X. Guan, Y. Yang, and J. You. Typing Evolving Ambients. *Information Processing Letters*, 80(5), 265-270, 2001.
11. F. Levi. Types for Evolving Communication in Safe Ambients (Draft). Available at http://www.di.unipi.it/~levifran/papers.html.
12. F. Levi and D. Sangiorgi. Controlling Interference in Ambients. In *Proc. POPL '00*, pages 352–364. ACM Press, 2000.
13. M. Merro and M. Hennessy. Bisimulation congruences in Safe Ambients. In *Proc. POPL '02*. ACM Press, 2002.
14. R. Milner, J. Parrow, and D. Walker. A calculus of mobile processes, (Parts I and II). *Information and Computation*, 100:1–77, 1992.
15. P. Zimmer Subtyping and typing algorithms for mobile ambients. In *Proc. FOSSACS '00*, volume 1784 of *Lecture Notes in Computer Science*, pages 375–390. Springer Verlag, 2000.