DYNAMIC AND LOCAL TYPING FOR MOBILE AMBIENTS

Mario Coppo[1*], Mariangiola Dezani-Ciancaglini[1†],
Elio Giovannetti[1‡], and Rosario Pugliese[2§]

[1]*Dip. di Informatica, Univ. di Torino, Corso Svizzera 185, 10149 Torino, Italy*
{coppo,dezani,elio}@di.unito.it

[2]*Dip. di Sistemi e Informatica, Univ. di Firenze, v. Lombroso 6/17, 50134 Firenze, Italy*
pugliese@dsi.unifi.it

Abstract An ambient calculus with both static and dynamic types is presented, where the latter ones represent mobility and access rights that may be dynamically consumed and acquired in a controlled way. Novel constructs and operations are provided to this end. Type-checking is purely local, except for a global hierarchy that establishes which locations have the authority to grant rights to which: there is no global environment (for closed terms) assigning types to names. Each ambient or process move is subject to a double authorization, one static and the other dynamic: static type-checking controls (communication and) "active" mobility rights, i.e., where a given ambient or process has the right to go; dynamic type-checking controls "passive" rights, i.e., which ambients a given ambient may be crossed by and which processes it may receive.

Keywords: Ambient calculi, type systems for security, local type checking, dynamic exchange of rights.

1. Introduction

The ever growing importance, in the last decades, of forms of distributed and mobile computing over wide physical or virtual domains has prompted the design of new theoretical models of computing: in particular, distributed process calculi and ambient calculi, for example [Hennessy and Riely, 2002; Cardelli and Gordon, 2000; Levi and Sangiorgi, 2003; Bugliesi et al., 2004].

[*] Partially supported by EU FET - Global Computing initiative, project DART IST-2001-33477.
[†] Partially supported by EU FET - Global Computing initiative, project DART IST-2001-33477.
[‡] Partially supported by EU FET - Global Computing initiative, project MIKADO IST-2001-32222.
[§] Partially supported by EU FET - Global Computing initiative, project MIKADO IST-2001-32222.

All such models rely on (often sophisticated) type systems for expressing and checking behavioural properties concerning mobility, resource access, security, etc. In most of them, a system or component is represented by a term t of a given calculus, a type T assigned to t, and an environment Γ. In the standard view, as is well-known, the term t abstractly describes the implementation, its type T may express some behavioural properties, and the environment Γ is a set of assumptions on the outside world. Typically, these are assumptions on types of non-local names; there is thus the notion of a global environment, which is the abstract description of situations where all the interacting parties are known in advance to each other, so that static checks performed before execution ensure the correctness of the whole system.

When dealing with distributed and mobile computing in wide-area "open" systems, however, one is often confronted with a scenario where interaction may take place between parties whose respective properties are unknown or only partially known to each other. If stopping the execution for re-checking is to be avoided, each component must dynamically carry with it sufficient behavioural information that can be checked at runtime by the other ones interacting with it. This may correspond to a formal system where, like in the one proposed in [Bugliesi and Castagna, 2002], the typing judgment $\Gamma \vdash_a t{:}T$ is relative to a locality a, and may be "packed" into a new kind of term $a[t]^{\Gamma}_{T}$ that carries at runtime the typing information.

In this paper we address the same problem (of theoretically modelling such kind of scenarios) via a new approach that, though similar in spirit to the one of [Bugliesi and Castagna, 2002] just recalled, is nevertheless different from it in several aspects.

We present a typed ambient calculus where global type assumptions on ambient names are eliminated, and the only global assumptions left are those on the input variables which, owing to their nature, have only a limited scope and do not span the whole system.

Behavioural type assumptions are instead local to each ambient. Thus an ambient type, which in most calculi [Cardelli et al., 2000; Bugliesi and Castagna, 2002; Bugliesi et al., 2004; Coppo et al., 2003; Bugliesi et al., 2003] specifies the behaviour of all ambients with the same name or group and requires cross-reference type controls, is here attached to a single ambient occurrence: ambients with the same name or group, occurring in different parts of a system, can have different types. The absence of static global type information requires the introduction of runtime types, with dynamic controls which test the compatibility between different local assumptions.

Our calculus is based on $\mathbf{M}^3$ [Coppo et al., 2003], a variant of the Calculus of Mobile Ambients (MA) where the open primitive for dissolving ambient boundaries has been removed and its role in enabling component interaction is played by primitives for general process mobility. The new operators were inspired by the go primitive of Dπ [Hennessy and Riely, 2002], but they are tailored to the ambient nested structure, which is richer than the flat structure of Dπ locations.

As a matter of fact, the open primitive of MA has been considered by many researchers as potentially dangerous, because it could be inadvertently or maliciously used to destroy an ambient's individuality (by dissolving its boundary). Several variants of MA have therefore been proposed, which either are equipped with additional constructs for controlling the execution of open, like co-capabilities of Safe Ambients [Levi and Sangiorgi, 2003], or replace it with other mechanisms for interaction between ambients, such as the communication between nested ambients that characterizes Boxed Ambients [Bugliesi et al., 2004]. Process mobility, however, seems to be a more suitable mechanism for modelling code exchange and remote execution.

The rest of the paper is organized as follows. In the next section we describe the main features of our system. In sect. 3 a meaningful example, which illustrates the use of the different constructs of the calculus, is developed at length. In sect. 4 we draw some short conclusions.

Due to space limitation most technical details, like the proof of subject reduction, have been omitted. We refer the interested reader to the full paper [Coppo et al., 2004] for a detailed account. In it we also introduce, along with further examples, a behavioural semantics (i.e., barbed congruence) and some equivalence laws based on it. The soundness of the laws is proved through standard techniques by relying on a higher-order labelled transition system and on a labelled bisimulation (which is proved sound w.r.t. to the behavioural semantics).

2. The typed language and its reduction semantics

The syntax of the pre-terms of the language (where type constraints are ignored) is given in Fig. 1, the one of types in Fig. 2. The precise syntax of the language results from the typing rules given in Definition 3. Processes are built through the usual constructs of sequential action prefixing, parallel composition, and ambient construction. In the following a process of the form $m{:}g(G)[\mathsf{c},\mathsf{e}\|P]$, which corresponds, in our richer calculus, to the term $m[P]$ of standard ambient calculi, will be simply called an ambient, distinct from the mere ambient name m. Communication is only local (and synchronous), as in the original MA [Cardelli and Gordon, 2000]. Actions include the usual in and out primitives for ambient mobility, and the two new primitives down and up for moving processes between ambients; already taken into consideration in [Coppo et al., 2003], they replace the to primitive of $\mathbf{M}^3$.

The down action is analogous to the in action. Its (simplified) reduction rule (see [Coppo et al., 2003]) is: $\mathsf{down}\ m.P \mid Q \mid m[R] \ \rightarrow\ Q \mid m[P \mid R]$; i.e., the process $\mathsf{down}\ m.P$ enters an ambient named m where it continues as P. On the contrary, the up action is only partially analogous to out, since its argument is the name of the destination ambient and not that of the source ambient, like in the case of out. The corresponding (simplified) reduction rule is $m[n[\mathsf{up}\ m.P \mid R] \mid Q] \ \rightarrow\ m[P \mid n[R] \mid Q]$, also given in [Coppo et al., 2003]. The explicit mention of the target ambient allows more effective controls on the incoming process. In the present setting the four mobility primitives, though keeping their basic behaviours just recalled, come with richer syntactic forms that correspond to more sophisticated reduction rules, as we will see later in the section.

Types describe communication and mobility properties. With reference to an ambient, we distinguish between its active and passive mobility: by the former we intend which ambients the given ambient may cross or send processes to; by the latter, the ambients by which it may be crossed or sent processes. Of course, by directly specifying the active mobility of each ambient one indirectly specifies the passive mobility of all the concerned ambients, and vice versa. One of the main features of our system is

$\mathscr{A}$ denotes the set of *ambient names* and $\mathscr{G}$ denotes the set of *group names*.

α	$::=$	**ambients**	
		$m, n, \ldots$	ambient names
		$x, y, \ldots$	variables

γ	$::=$	**groups**	
		$g, h, \ldots$	group names
		$x, y, \ldots$	variables

χ	$::=$	**capabilities**	
		$\mathsf{in}\ \alpha{:}g$	moves the containing ambient into ambient α of group g
		$\mathsf{out}\ \alpha{:}g$	moves the containing ambient out of ambient α of group g
		$\mathsf{up}\ \alpha{:}g\ \mathsf{with}\ G$	moves the continuation process out from its ambient up to enclosing ambient α of group g requiring rights G
		$\mathsf{down}\ \alpha{:}g\ \mathsf{with}\ G$	moves the continuation process from its ambient down to enclosed ambient α of group g requiring rights G
		$\mathsf{add}^{\mathsf{c}}\ \gamma^{\varphi}\ \mathsf{in}\ \alpha{:}g$	adds the crossing right γ with multiplicity φ to ambient α of group g
		$\mathsf{add}^{\mathsf{e}}\ \langle\gamma, G\rangle^{\varphi}\ \mathsf{in}\ \alpha{:}g$	adds the entering right $\langle\gamma, G\rangle$ with multiplicity φ to ambient α of group g
		$\chi{\cdot}\chi'$	path
		$x, y, \ldots$	variables

$M,\ N,\ L$	$::=$	**messages**	
		α	ambients
		γ	groups
		χ	capabilities

ρ	$::=$	**guards**	
		χ	capabilities
		$\langle\overrightarrow{M}\rangle$	synchronous output
		$(\overrightarrow{x{:}\,W})$	typed input

$P,\ Q,\ R$	$::=$	**processes**	
		$\mathsf{0}$	null
		$\rho\,.\,P$	prefixing
		$P \mid Q$	parallel composition
		$\alpha{:}g(G)[\mathsf{c}, \mathsf{e} \Vert P]$	ambient
		$!\,\rho\,.\,P$	guarded replication
		$(\nu n)P$	name restriction

where c are multisets of groups, e are multisets of pairs $\langle g, G\rangle$; $\overrightarrow{W}$ and G are defined in Figure 2.

Figure 1. Syntax

		$\mathscr{C}, \mathscr{E}, \ldots$	sets of groups;
G	$::=$	$\mathsf{mc}(\mathscr{C}, \mathscr{E}, T)$	*mobcom* type: mobility rights and communication type
Pro	$::=$	$g(G)$	process type: processes that can stay in ambients of group g with rights G
Cap	$::=$	$g'(G') \rightsquigarrow g(G)$	capabilities that can be consumed by processes of type $g'(G')$ and leave processes of type $g(G)$ as continuations
W	$::=$		message type
		Cap	capability type
		group	group
		amb	ambient type
T	$::=$		communication type
		shh	no communication
		$\overrightarrow{W}$	communication of messages of type $\overrightarrow{W}$
Σ	$::=$	$\varnothing$	context
		$\Sigma, x : W$	

Figure 2. Types

that static types directly specify active mobility while dynamic types directly specify passive mobility, and the compatibility between them is tested by runtime checks.

The type system is based on *ambient groups*: a group is a name that represents (i.e., labels) a set of ambient name occurrences. Different ambient names may belong to the same group, but at the same time different occurrences of the same ambient name may be labelled with different group names, i.e., different ambients with the same name may belong to different groups. The mobility properties directly specified for each ambient are always expressed with reference to groups and not to individual ambients, so as to avoid a dependence of types from values.

In order to enable an ambient to check at runtime (i.e., during reduction) that the active types of the incoming processes are compatible with its own type, the primitives for moving processes between ambients carry with them the communication and mobility types of their continuations.

While local typing allows the control of the active mobility behaviour, the absence of global type information makes impossible a static control of the passive behaviour. This check has therefore also to be performed dynamically; for that reason, at runtime each ambient carries a specification of which (groups of) ambients can cross it and which can send it processes, and how many times.

Thus every mobility action becomes subject to a double authorization, one static and the other dynamic. The fact that “passive” permits are dynamically checked allows them to also be dynamically granted; to this end, we have introduced two new primitives through which a process may enrich the rights of another one thus enabling it to carry out a given task.

Static types and their packing for runtime use

The static type system is centered on the notion of process type, which consist *both* of a group name g *and* of a mobility and communication type (or *mobcom* type for short) G; following the notation of [Cardelli et al., 1999], we write it $g(G)$.

The mobcom type G is of the form $\mathsf{mc}(\mathscr{C}, \mathscr{E}, T)$, where $\mathscr{C}$ is the set of (groups of) ambients into which the process may drive (through an in or out action) its enclosing ambient, $\mathscr{E}$ the set of (groups of) ambients to which it may send (through a down or up action) a continuation process, and T is the process communication type. We use the notation $\mathscr{C}(G)$, $\mathscr{E}(G)$, $T(G)$ to respectively indicate the components $\mathscr{C}$, $\mathscr{E}$, T of G.

Like in most ambient calculi, all the (parallel) processes within an ambient must have the same process type $g(G)$, which is thus a sort of inner type of that ambient and, as we will see, is bound locally to it. Ambient names, on the other hand, only have the atomic type amb, and are therefore omitted in the environments. As a consequence, name restriction may be simply written as $(\nu m)P$. Similarly, group names have the atomic type group. Group names can be exchanged in communications but group variables can only be used in a limited way, as will be remarked later.

Ambient and process mobility actions must specify not only an ambient name, as in MA, but also a group name. For instance, the syntax of the usual in and out primitives becomes $\mathsf{in}\alpha{:}g$, $\mathsf{out}\alpha{:}g$ (with α ambient name or variable). A process may contain one such action (i.e., it may be well typed) only if its type allows it to drive its enclosing ambient across the boundary of ambients labelled with the group name g. Values exchanged in communications may be ambient names (of type amb), group names (of type group), or capabilities.

A capability type consists, as in [Coppo et al., 2003], of a pair of process types, written here with the notation $g'(G') \rightsquigarrow g(G)$. Capability types take into account the fact that, owing to the down or up actions, a process can move from one ambient to another, changing its type accordingly. A capability χ of type $g'(G') \rightsquigarrow g(G)$ drives a continuation process P from an ambient of (inner) type $g'(G')$ to an ambient of type $g(G)$. Obviously P must have the type $g(G)$ of the destination ambient while the resulting process $\chi.P$ has the type $g'(G')$ of the source ambient. This is formalized in the rule ($\textsc{Prefix}$) of Fig. 3. The type of a sequence of in- or out-capabilities has the form $g(G) \rightsquigarrow g(G)$, because in executing in or out actions a process remains in the same ambient.

The meaning of types, informally described in the body of this section, is formally defined by the set of the typing rules shown in Fig. 3.

Dynamic types

Type controls no longer statically performed must of course be done dynamically. To this end, the inner type $g(G)$ of an ambient named m is bound to it with the notation $m{:}g(G)[\ldots]$. An ambient is also characterized by two components c and e that record by which ambient or process groups it may be entered and how many times. The complete notation is $\alpha{:}g(G)[\mathsf{c}, \mathsf{e} \| P]$, with α variable or ambient name.

$$\frac{n \in \mathscr{A}}{\mathcal{O};\Sigma \vdash n : \mathsf{amb}}\ (\textsc{Amb Const}) \qquad \frac{g \in \mathscr{G}}{\mathcal{O};\Sigma \vdash g : \mathsf{group}}\ (\textsc{Grp Const})$$

$$\frac{x : W \in \Sigma}{\mathcal{O};\Sigma \vdash x : W}\ (\textsc{Env}) \qquad \frac{}{\mathcal{O};\Sigma \vdash \mathsf{0} : ((g)G)}\ (\textsc{Null})$$

$$\frac{\mathcal{O};\Sigma \vdash \alpha : \mathsf{amb} \quad g \in \mathscr{C}(G)}{\mathcal{O};\Sigma \vdash \mathsf{in}\ \alpha{:}g : g'(G) \rightsquigarrow g'(G)}\ (\textsc{In}) \qquad \frac{\mathcal{O};\Sigma \vdash \alpha : \mathsf{amb} \quad g \in \mathscr{C}(G)}{\mathcal{O};\Sigma \vdash \mathsf{out}\ \alpha{:}g : g'(G) \rightsquigarrow g'(G)}\ (\textsc{Out})$$

$$\frac{\mathcal{O};\Sigma \vdash \alpha : \mathsf{amb} \quad g \in \mathscr{E}(G')}{\mathcal{O};\Sigma \vdash \mathsf{down}\ \alpha{:}g\ \mathsf{with}\ G : g'(G') \rightsquigarrow g(G)}\ (\textsc{Down})$$

$$\frac{\mathcal{O};\Sigma \vdash \alpha : \mathsf{amb} \quad g \in \mathscr{E}(G')}{\mathcal{O};\Sigma \vdash \mathsf{up}\ \alpha{:}g\ \mathsf{with}\ G : g'(G') \rightsquigarrow g(G)}\ (\textsc{Up})$$

$$\frac{\langle\!\langle g, g' \rangle\!\rangle \in \mathcal{O} \quad \mathcal{O};\Sigma \vdash \alpha : \mathsf{amb} \quad \mathcal{O};\Sigma \vdash \gamma : \mathsf{group}}{\mathcal{O};\Sigma \vdash \mathsf{add}^{\mathsf{c}}\ \gamma^{\varphi}\ \mathsf{in}\ \alpha{:}g : g'(G) \rightsquigarrow g'(G)}\ (\textsc{Add-c})$$

$$\frac{\langle\!\langle g, g' \rangle\!\rangle \in \mathcal{O} \quad \mathcal{O};\Sigma \vdash \alpha : \mathsf{amb} \quad \mathcal{O};\Sigma \vdash \gamma : \mathsf{group}}{\mathcal{O};\Sigma \vdash \mathsf{add}^{\mathsf{e}}\ \langle \gamma, G \rangle^{\varphi}\ \mathsf{in}\ \alpha{:}g : g'(G') \rightsquigarrow g'(G')}\ (\textsc{Add-e})$$

$$\frac{\mathcal{O};\Sigma \vdash \chi : g'(G') \rightsquigarrow g''(G'') \quad \mathcal{O};\Sigma \vdash \chi' : g''(G'') \rightsquigarrow g(G)}{\mathcal{O};\Sigma \vdash \chi . \chi' : g'(G') \rightsquigarrow g(G)}\ (\textsc{Path})$$

$$\frac{\mathcal{O};\Sigma \vdash \chi : g'(G') \rightsquigarrow g(G) \quad \mathcal{O};\Sigma \vdash P : g(G)}{\mathcal{O};\Sigma \vdash \chi . P : g'(G')}\ (\textsc{Prefix})$$

$$\frac{\mathcal{O};\Sigma, \overrightarrow{x : W} \vdash P : g(G) \quad G \equiv \mathsf{mc}(\mathscr{C}, \mathscr{E}, \overrightarrow{W})}{\mathcal{O};\Sigma \vdash (\overrightarrow{x : W})\,.\,P : g(G)}\ (\textsc{Input})$$

$$\frac{\mathcal{O};\Sigma \vdash P : g(G) \quad \mathcal{O};\Sigma \vdash \overrightarrow{M} : \overrightarrow{W} \quad G \equiv \mathsf{mc}(\mathscr{C}, \mathscr{E}, \overrightarrow{W})}{\mathcal{O};\Sigma \vdash \langle \overrightarrow{M} \rangle\,.\,P : g(G)}\ (\textsc{Output})$$

$$\frac{\mathcal{O};\Sigma \vdash \alpha : \mathsf{amb} \quad \mathcal{O};\Sigma \vdash P : g(G) \quad \langle g'', G'' \rangle \in \mathsf{e} \implies G'' \leq G}{\mathcal{O};\Sigma \vdash \alpha{:}g(G)[\mathsf{c}, \mathsf{e} \| P] : g'(G')}\ (\textsc{Amb})$$

$$\frac{\mathcal{O};\Sigma \vdash P : g(G) \quad \mathcal{O};\Sigma \vdash Q : g(G)}{\mathcal{O};\Sigma \vdash P \,|\, Q : g(G)}\ (\textsc{Par}) \qquad \frac{\mathcal{O};\Sigma \vdash \rho\,.\,P : g(G)}{\mathcal{O};\Sigma \vdash !\,\rho\,.\,P : g(G)}\ (\textsc{Repl})$$

$$\frac{\mathcal{O};\Sigma \vdash P : g(G)}{\mathcal{O};\Sigma \vdash (\nu n)P : g(G)}\ (\textsc{Amb Res})$$

Figure 3. Typing rules

More precisely, in a given ambient the component c is the multiset of groups of ambients that are allowed to cross its external boundary, while e is the multiset of groups of ambients that are allowed to send processes to it. In e each element is actually a pair $\langle g, G\rangle$ consisting of a group and a mobcom type; its meaning is that a process coming from an ambient of group g is given the entrance permit if it respects the behavioural constraints specified by the type G. In an ambient $m{:}g_m(G_m)[\mathsf{c}, \mathsf{e} \| P]$ all the pairs $\langle g, G\rangle$ occurring in e must therefore be such that G is a subtype of G_m.

Each execution of an in or out action consumes one element of c and each execution of a down or up action consumes one element of e, with the exception of *starred* elements, which represent permanent permits, i.e., elements with infinite multiplicity. The control of the mobility constraints represented by c and e is performed dynamically during process reduction. A reduction rule cannot fire if the corresponding side conditions on c and e are not satisfied.

In the following we give the formal definition of a multiset with possibly infinite multiplicities, and we define the operations of addition and removal of elements.

DEFINITION 1

- *A multiset over a set of elements $\mathcal{S}$ is a function from $\mathcal{S}$ to the set of multiplicities $\omega \cup \{*\}$ (ranged over by f); $\omega \cup \{*\}$ is the set of natural numbers ω extended with the extra element $*$ denoting the infinity;*
- *If f is a multi-set on $\mathcal{S}$, $s, r \in \mathcal{S}$, $\varphi \in \omega \cup \{*\}$ we define:*
  - $s \in \mathsf{f}$ *iff* $\mathsf{f}(s) \neq 0$;
  - $(\mathsf{f} \cup s^{\varphi})(r) = \begin{cases} \mathsf{f}(s) + \varphi & \textit{if } r = s, \\ \mathsf{f}(r) & \textit{otherwise}. \end{cases}$
  - $(\mathsf{f} \downarrow s)(r) = \begin{cases} \mathsf{f}(s) - 1 & \textit{if } r = s, \\ \mathsf{f}(r) & \textit{otherwise}. \end{cases}$

  where $* \pm \varphi = *$.

A partial order on mobcom types is naturally defined via set inclusion, and so is the notion of glb of mobcom types. Communication subtyping is characterized only by the fact that shh is smaller than any other communication type.

DEFINITION 2

- $T \leq T'$ *if either* $T = \mathsf{shh}$ *or* $T = T'$.
- $\mathsf{mc}(\mathscr{C}, \mathscr{E}, T) \leq \mathsf{mc}(\mathscr{C}', \mathscr{E}', T')$ *if* $\mathscr{C} \subseteq \mathscr{C}'$ *and* $\mathscr{E} \subseteq \mathscr{E}'$ *and* $T \leq T'$.
- $T \sqcap T' = \begin{cases} \mathsf{shh} & \textit{if } T = \mathsf{shh} \textit{ or } T' = \mathsf{shh} \\ T & \textit{if } T = T' \\ \textit{undefined} & \textit{otherwise} \end{cases}$
- $\mathsf{mc}(\mathscr{C}, \mathscr{E}, T) \sqcap \mathsf{mc}(\mathscr{C}', \mathscr{E}', T') = \mathsf{mc}(\mathscr{C} \cap \mathscr{C}', \mathscr{E} \cap \mathscr{E}', T \sqcap T')$
  if $T \sqcap T'$ is defined and is undefined otherwise.

The elements of c and e are similar to the co-capabilities of Safe Ambients [Levi and Sangiorgi, 2003], with starred elements corresponding to banged co-capabilities.

Dynamic modification of mobility rights

The components c and e of an ambient process may allow or forbid movements at runtime; they can therefore be changed dynamically without breaking the subject reduction. As a matter of fact, this is achieved:

- by automatically removing a (consumable) permit when a movement action is performed;
- by adding to c or to e an element with a multiplicity, by means of one of the two newly introduced permit-adding primitives $\mathsf{add}^{\mathsf{c}}$ and $\mathsf{add}^{\mathsf{e}}$.

Action $\mathsf{add}^{\mathsf{c}}\ g^{\varphi}\ \mathsf{in}\ m{:}g_m$ dynamically adds the group g with multiplicity φ to the c component of a local ambient named m of group g_m (see the reduction rule (R-add$^{\mathsf{c}}$)); as usual, by "local ambient" we intend one that is found in the same enclosing ambient. Action $\mathsf{add}^{\mathsf{e}}\ \langle g, G_1\rangle^{\varphi}\ \mathsf{in}\ m{:}g_m$ dynamically adds the group/type pair $\langle g, G_1\rangle$ with multiplicity φ to the e component of a local ambient m (rule (R-add$^{\mathsf{e}}$)). In a term $m{:}g_m(G_m)[\mathsf{c}, \mathsf{e} \| P]$ all mobcom types occurring in e are subtypes of G_m. This property is preserved by the reduction rule (R-add$^{\mathsf{e}}$) since the $\mathsf{add}^{\mathsf{e}}\ \langle g, G_1\rangle^{\varphi}\ \mathsf{in}\ m{:}g_m$ action can be performed only if $G_1 \sqcap G_m$ is defined.

A process may perform a permit-adding operation on an ambient only if its group is higher than the target's group in a global administrative hierarchy, represented by a partial order relation $\mathcal{O}$ over group names. Such hierarchy is the only global environment of our calculus; it might be thought of as some general (not necessarily centralized) coordination and administration structure of the network. The typing rules (ADD-C), (ADD-E) assure that this hierarchy is always respected.

Mobility actions and dynamic type-checking

Process mobility actions must specify, in addition to the ambient and group name of the destination, the mobcom type G of the continuation process (i.e., of the process that will run within the target ambient). The complete syntax of the down action is $\mathsf{down}\, m{:}g_m\, \mathsf{with}\, G$, that of up is similar. Of course, the type G needs to be compatible (via subtyping) with the mobcom type G_m of the destination ambient. More precisely, if a process $\mathsf{down}\ m{:}g_m\ \mathsf{with}\ G'\,.\,P$ is of group g (i.e., is typed with a type $g(G)$) the e component of the target ambient m must contain the group g paired with a type G'' such that $G' \leq G''$. The typing rule (AMB), along with the reduction rule (R-add$^{\mathsf{e}}$), ensures that $G'' \leq G_m$ and so $G' \leq G'' \leq G_m$. Hence, the migrating process P is guaranteed not to require more rights than those specified by the inner type $g_m(G_m)$ of the destination ambient, which was statically checked.

Reduction rules are thus dependent on the typing assumptions and the reduction relation is labelled with the process type $g(G)$, even though the group name g is only involved in the (R-down) rule and the type G never plays any role in reduction. In fact reduction rules are only defined for well typed terms, i.e., for processes that are typed with some type $g(G)$. The complete set of rules is given in Fig. 4.

A basic property of the system is that typing is preserved by $\leq$ on mobcom types.

LEMMA 3 *If $\mathcal{O}; \Sigma \vdash P : g(G)$ and $G \leq G'$ then $\mathcal{O}; \Sigma \vdash P : g(G')$*

Basic reduction rules:

(R-in) $n{:}g_n(G_n)[\mathsf{c}_n, \mathsf{e}_n \| \,\mathsf{in}\; m{:}g_m \,.\, P \mid Q\,] \mid m{:}g_m(G_m)[\mathsf{c}_m, \mathsf{e}_m \| R]$
$\quad \to_{g,G}\; m{:}g_m(G_m)[\mathsf{c}_m \downarrow g_n, \mathsf{e}_m \| \, n{:}g_n(G_n)[\mathsf{c}_n, \mathsf{e}_n \| P \mid Q\,] \mid R\,]$
if $g_n \in \mathsf{c}_m$

(R-out) $m{:}g_m(G_m)[\mathsf{c}_m, \mathsf{e}_m \| \, n{:}g_n(G_n)[\mathsf{c}_n, \mathsf{e}_n \| \,\mathsf{out}\; m{:}g_m \,.\, P \mid Q\,] \mid R\,]$
$\quad \to_{g,G}\; n{:}g_n(G_n)[\mathsf{c}_n, \mathsf{e}_n \| \, P \mid Q\,] \mid m{:}g_m(G_m)[\mathsf{c}_m \downarrow g_n, \mathsf{e}_m \| R]$
if $g_n \in \mathsf{c}_m$

(R-down) $\mathsf{down}\; m{:}g_m\; \mathsf{with}\; G' \,.\, P \mid m{:}g_m(G_m)[\mathsf{c}_m, \mathsf{e}_m \| Q]$
$\quad \to_{g,G}\; m{:}g_m(G_m)[\mathsf{c}_m, \mathsf{e}_m \downarrow \langle g, G'' \rangle \| P \mid Q]$
if $\langle g, G'' \rangle \in \mathsf{e}_m \;\&\; G' \leq G''$

(R-up) $m{:}g_m(G_m)[\mathsf{c}_m, \mathsf{e}_m \| n{:}g_n(G_n)[\mathsf{c}_n, \mathsf{e}_n \| \mathsf{up}\; m{:}g_m\; \mathsf{with}\; G' \,.\, P \mid R] \mid Q]$
$\quad \to_{g,G}\; m{:}g_m(G_m)[\mathsf{c}_m, \mathsf{e}_m \downarrow \langle g_n, G'' \rangle \| n{:}g_n(G_n)[\mathsf{c}_n, \mathsf{e}_n \| R] \mid Q \mid P]$
if $\langle g_n, G'' \rangle \in \mathsf{e}_m \;\&\; G' \leq G''$

(R-add$^{\mathsf{c}}$) $\mathsf{add}^{\mathsf{c}}\; g'^{\varphi}\; \mathsf{in}\; m{:}g_m \,.\, P \mid m{:}g_m(G_m)[\mathsf{c}_m, \mathsf{e}_m \| R]$
$\quad \to_{g,G}\; P \mid m{:}g_m(G_m)[\mathsf{c}_m \cup g'^{\varphi}, \mathsf{e}_m \| R]$

(R-add$^{\mathsf{e}}$) $\mathsf{add}^{\mathsf{e}}\; \langle g', G' \rangle^{\varphi}\; \mathsf{in}\; m{:}g_m \,.\, P \mid m{:}g_m(G_m)[\mathsf{c}_m, \mathsf{e}_m \| R]$
$\quad \to_{g,G}\; P \mid m{:}g_m(G_m)[\mathsf{c}_m, \mathsf{e}_m \cup \langle g', G' \sqcap G_m \rangle^{\varphi} \| R]$
if $G' \sqcap G_m$ is defined

(R-comm) $(\overrightarrow{x{:}\,W}) \,.\, P \mid \langle \overrightarrow{M} \rangle \,.\, Q \;\to_{g,G}\; P\{\overrightarrow{x} := \overrightarrow{M}\} \mid Q$

Structural reduction rules:

(R-par) $P \;\to_{g,G}\; Q \quad \Rightarrow \quad P \mid R \;\to_{g,G}\; Q \mid R$

(R-amb) $P \;\to_{g,G}\; Q \quad \Rightarrow \quad n{:}g(G)[\mathsf{c}, \mathsf{e} \| P] \;\to_{g',G'}\; n{:}g(G)[\mathsf{c}, \mathsf{e} \| Q]$

(R-$\equiv$) $P' \equiv P',\; P \;\to_{g,G}\; Q,\; Q \equiv Q' \quad \Rightarrow \quad P' \;\to_{g,G}\; Q'$

(R-ν) $P \;\to_{g,G}\; Q \quad \Rightarrow \quad (\nu n)P \;\to_{g,G}\; (\nu n)Q$

Structural Congruence: $(\,|\,, 0)$ is a commutative monoid.

$$(\nu n)(P \mid Q) \;\equiv\; (\nu n)P \mid Q \quad (n \notin \mathrm{fn}(Q)) \qquad (\nu n)(\nu m)P \;\equiv\; (\nu m)(\nu n)P$$

$$n{:}g(G)[\mathsf{c}, \mathsf{e} \| (\nu m)P] \;\equiv\; (\nu m)n{:}g(G)[\mathsf{c}, \mathsf{e} \| P] \quad (n \neq m) \qquad !\,P \;\equiv\; P \mid \,!\,P$$

Figure 4. Reduction

Using Lemma 3 a property of subject reduction, which ensures that static typing is preserved by computation, can be proved with standard techniques.

THEOREM 4 (SUBJECT REDUCTION) *If* $\mathcal{O}; \Sigma \vdash P : g(G)$ *and* $P \rightarrow_{g,G} Q$ *then* $\mathcal{O}; \Sigma \vdash Q : g(G)$.

Finally, observe that group variables may only occur as first arguments of $\mathsf{add}^{\mathsf{c}}$ and $\mathsf{add}^{\mathsf{e}}$, so that they never occur in G or within the c and e components, since otherwise their role in allowing a safe name restriction would be defeated.

3. An example

Our main example is the modelling of a public transportation system, the *train* introduced by [Cardelli, 1999] as a nice pictorial illustration of the issues related to the control of mobility.

We want to represent a railway network connecting a set of different places (e.g., cities) in the world. Trains move between stations, travellers may get into and off trains only at stations and cannot drive them (no hijacking possible). The number of passengers in a train at any given instant cannot exceed the number of seats; a passenger takes a seat on boarding and releases it on getting off. Each train has a fixed route.

For the sake of simplicity, we assume that:

- There is a top-level untrusted ambient *world*, which includes stations, travellers, and some other unspecified process R (e.g., other means of transport); it has group and mobcom type $g_w(G_w)$, but no assumption can be made on G_w. Also, c_{world}, e_{world} and R are unknown.
- In our intended representation different stations should be found within different cities or localities, and moving from one city to another would only be possible by train. The presence of cities would however increase the size of the example in a trivial manner, without providing more insights; we therefore place stations directly within *world*, although in this way travellers appear to use a train to end up in the same ambient *world* where they started from.
- There are only two stations *stA* and *stB*, and one train TRAIN commuting between them. Initially, the train is within *stA*.

Stations and trains are represented by ambient processes; travellers are represented by simple processes; the number of free seats in a train is represented by the multiplicity of the right to get into the train.

Stations are ambients $stA{:}g_{st}(G_{st})[\ldots]$ and $stB{:}g_{st}(G_{st})[\ldots]$, of group g_{st} and mobcom type G_{st}. They are immobile, and can have travellers both going down into the trains or up into the world; they can be crossed by trains, and can receive travellers both from train and from the outside world. Correspondingly:

$$G_{st} = \mathsf{mc}(\varnothing, \{g_{tr}, g_w\}, \mathsf{shh}) \qquad \mathsf{c}_{st} = \{g_{tr}^*\} \qquad \mathsf{e}_{st} = \{\langle g_{tr}, G_{to}\rangle^*, \langle g_w, G_{from}\rangle^*\}$$

where $G_{from} = \mathsf{mc}(\varnothing, \{g_{tr}\}, \mathsf{shh})$ and $G_{to} = \mathsf{mc}(\varnothing, \{g_w\}, \mathsf{shh})$. Note that $G_{from} \leq G_{st}$ and $G_{to} \leq G_{st}$, i.e., both G_{from} and G_{to}, which represent two accepted behaviours for processes entering the station, are compatible with G_{st}, as is required by the typing rule (Amb).

The train is an ambient of group g_{tr}, which can cross stations and world, send traveller processes into stations, and receive a maximum number n of passengers from stations, provided they behave as good passengers (and not, for example, as drivers):

$$\begin{aligned}&\textsc{train} \triangleq \mathit{tr}{:}g_{tr}(G_{tr})[\mathsf{c}_{tr}, \mathsf{e}_{tr} \| \,!\, \mathsf{out}\ \mathit{stA}{:}g_{st} \,.\, \mathsf{in}\ \mathit{stB}{:}g_{st} \,.\, \mathsf{out}\ \mathit{stB}{:}g_{st} \,.\, \mathsf{in}\ \mathit{stA}{:}g_{st}]\\ &\text{where } \ G_{tr} = \mathsf{mc}(\{g_{st}\}, \{g_{st}\}, \mathsf{shh}) \quad \mathsf{c}_{tr} = \varnothing \quad \mathsf{e}_{tr} = \{\langle g_{st}, G_{psng}\rangle^{\mathsf{n}}\}\\ &\qquad\qquad\qquad\text{with } \ G_{psng} = \mathsf{mc}(\varnothing, \{g_{st}\}, \mathsf{shh})\end{aligned}$$

A traveller is represented by a parametric process $\textsc{traveller}(\mathit{src}, \mathit{dst})$ which from some unspecified place in the world enters the station *src* to become a passenger of a train that takes it to the station *dst*:

$$\begin{aligned}&\textsc{traveller}(\mathit{src}, \mathit{dst}) \triangleq \mathsf{down}\ \mathit{src}{:}g_{st}\ \mathsf{with}\ G_{from} \,.\, \textsc{psng}\\ &\text{where } \ \textsc{psng} \triangleq \mathsf{down}\ \mathit{tr}{:}g_{tr}\ \mathsf{with}\ G_{psng} \,.\, \mathsf{up}\ \mathit{dst}{:}g_{st}\ \mathsf{with}\ G_{to}\\ &\qquad\qquad\quad .\, \mathsf{add}^{\mathsf{e}}\ \langle g_{st}, G_{psng}\rangle\ \mathsf{in}\ \mathit{tr} : g_{tr} \,.\, \mathsf{up}\ \mathit{world}{:}g_w\ \mathsf{with}\ G_w \,.\, \mathrm{P}\end{aligned}$$

The mobcom types G_{from} and G_{to} specify the behaviours of a passenger respectively in the departure station, going to board a train, and in the arrival station, going to exit the station into the outside world or city.

The initial configuration is:

$$\begin{aligned}&(\nu\ \mathit{stA}, \mathit{stB})\mathit{world} : g_w(G_w)[\mathsf{c}_w, \mathsf{e}_w \| R \mid \textsc{trvlrs}(\mathit{stA}, \mathit{stB})\\ &\qquad \mid\ \mathit{stA} : g_{st}(G_{st})[\mathsf{c}_{st}, \mathsf{e}_{st} \|\ \textsc{train}] \ \mid\ \mathit{stB} : g_{st}(G_{st})[\mathsf{c}_{st}, \mathsf{e}_{st} \| 0\,]\\ &\qquad \mid \textsc{trvlrs}(\mathit{stB}, \mathit{stA})\,]\end{aligned}$$

where $\textsc{trvlrs}(\mathit{src}, \mathit{dst})$ is a parallel composition of processes $\textsc{traveller}$.

Our specification satisfies many properties of interest; some of them immediately follow from the definitions. For instance, from the definition of e_{tr} it follows that no traveller can get into the train *tr* when this is outside a station: any action to such purpose from a process in *world* will be dynamically blocked. Also, by the definition of e_{tr} and of the $\textsc{psng}$ process[1] it follows that at most n $\textsc{psng}$ processes can be within the train *tr* at the same time.

A *bad* passenger willing to get off the train when this is not in a station, though it maybe statically well typed, is dynamically not allowed to do so. Suppose the bad passenger be represented by the process

$$\textsc{badpsng} \triangleq \mathsf{down}\ \mathit{tr}{:}g_{tr}\ \mathsf{with}\ G_{bad} \,.\, \mathsf{up}\ \mathit{world}{:}g_w\ \mathsf{with}\ G_w \,.\, \textsc{bp}$$

By assuming $\mathcal{O}; \Sigma \vdash \textsc{bp} : g_w(G_w)$ one may derive the typing

$$\mathcal{O}; \Sigma \vdash \mathsf{up}\ \mathit{world}{:}g_w\ \mathsf{with}\ G_w \,.\, \textsc{bp} : g_{tr}(G_{bad}) \ \text{ with } G_{bad} \triangleq \mathsf{mc}(\varnothing, \{g_w\}, \mathsf{shh})$$

Observe that the process type $g_{tr}(G_{bad})$ characterizes a process that might stay within the train and go from it directly into the world. From the above we may infer the typing $\mathcal{O}; \Sigma \vdash \textsc{badpsng} : g_{st}(G_{st})$, since for that it is enough that G_{st} allows the process to get into the train, i.e., $g_{tr} \in \mathscr{E}(G_{st})$.

[1]The present specification does not prevent a passenger to add more than one pair to e_{tr}.

The process BADPSNG is therefore statically allowed to stay within a station, as for example in the well-typed term $stA : g_{st}(G_{st})[\mathsf{c}_{st}, \mathsf{e}_{st} \| \text{BADPSNG} \mid \text{TRAIN}]$. Nevertheless, when trying at runtime to get into the train, the process is blocked because $\mathsf{e}_{tr} = \{\langle g_{st}, G_{psng}\rangle^{\mathsf{n}}\}$ (with $G_{psng} \leq G_{tr}$). As a matter of fact, for the action $\mathsf{down}\ tr{:}g_{tr}$ with G_{bad} to fire, it is required that $G_{bad} \leq G_{psng}$, which is not the case since $G_{bad} = \mathsf{mc}(\varnothing, \{g_w\}, \mathsf{shh})$ while $G_{psng} = \mathsf{mc}(\varnothing, \{g_{st}\}, \mathsf{shh})$: G_{bad} allows going into the world while G_{psng} does not.

This should have been somehow expected, because in our calculus the dynamic checks performed by an ambient are assigned the task of controlling that mobile processes willing to get in do respect some fixed policies expressed through types and, if this is not the case, of preventing them from getting in. Notice that all the previous properties are guaranteed by only exploiting in the operational semantics information local to the involved processes/ambients.

A similar scenario has already been modelled in [Cardelli, 1999; Ferrari et al., 2002]. In both cases, the mobility control is implemented by informing the passenger when the train has reached the station at which he wants to get off. More specifically, in [Cardelli, 1999] a new primitive for *ambient renaming* is exploited. Intuitively, the train ambient takes a suitable name to implicitly inform the passengers when it has arrived at a certain station, while it takes a name unknown to passengers when it is moving (in this way passengers cannot get in or off the train). In [Ferrari et al., 2002] a suitable ambient called *announcement* is generated by the train when it arrives at a station. This ambient informs the passengers of the arrival at a certain station.

4. Conclusions

The calculus we presented is a first attempt to model the interplay of static and dynamic type-checking when handling the security requirements of global computing applications. In particular, the packing of a mobility and communication type within a mobile process and its subsequent check at destination may be considered as an abstract modelling of the proof-carrying code approach.

Due to the absence of static ambient types (apart from the atomic type amb), static typing rules may be easily translated into a simple type inference algorithm that, given a term in whose body all the mobcom types are left unspecified, reconstructs the minimal such type allowing the term to be well typed. The algorithm will merely build a type by recording the capabilities occurring in the term. The groups assigned to ambient occurrences, on the other hand, as well as the dynamic components c and e, define the policy and the mobility constraints established by the designer of the application, and cannot be sensibly inferred.

A still unsatisfactory aspect of our model is that the authority (specified by the partial order $\mathcal{O}$) granting dynamic rights to ambients is a too coarse-grain notion: either an ambient is authorized to grant a right with any (even infinite) multiplicity, or the ambient may grant none. It would be useful that this authority could have different degrees, related to maximal multiplicities of granted rights. As noticed by one referee, another useful extension would be the introduction of a primitive for group restriction as in [Cardelli et al., 2000; Coppo et al., 2003]. This could provide protection from external untrusted agents, but the interaction with the partial order $\mathcal{O}$ representing the administrative hierarchy requires a careful handling. A modification of the calculus in this sense, along with a possible increasing of the expressivity of types, is currently under investigation.

Acknowledgements We gratefully acknowledge the anonymous referees for careful reading and useful suggestions.

References

Bugliesi, Michele and Castagna, Giuseppe (2002). Behavioral typing for Safe Ambients. *Computer Languages*, 28(1):61 – 99.

Bugliesi, Michele, Castagna, Giuseppe, and Crafa, Silvia (2004). Access control for mobile agents: The calculus of boxed ambients. *ACM Transactions on Programming Languages and Systems*, 26(1):57–124.

Bugliesi, Michele, Crafa, Silvia, Merro, Massimo, and Sassone, Vladimiro (2003). Communication and Mobility Control in Boxed Ambients. To appear in *Information and Computation*. Extended and revised version of M. Bugliesi, S. Crafa, M. Merro, and V. Sassone. Communication Interference in Mobile Boxed Ambients. In FSTTCS'02, volume 2556 of LNCS, pages 71-84. Springer-Verlag, 2002.

Cardelli, Luca (1999). Abstractions for mobile computation. In Vitek, Jan and Jensen, Christian, editors, *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*, volume 1603 of *LNCS*, pages 51–94. Springer-Verlag.

Cardelli, Luca, Ghelli, Giorgio, and Gordon, Andrew D. (1999). Mobility types for mobile ambients. In Wiedermann, Jiri, van Emde Boas, Peter, and Nielsen, Mogens, editors, *ICALP'99*, volume 1644 of *LNCS*, pages 230–239. Springer-Verlag.

Cardelli, Luca, Ghelli, Giorgio, and Gordon, Andrew D. (2000). Ambient groups and mobility types. In van Leeuwen, Jan, Watanabe, Osamu, Hagiya, Masami, and Peter D. Mosses, Takayasu Ito, editors, *International Conference IFIP TCS 2000*, volume 1872 of *LNCS*, pages 333–347. Springer-Verlag. Extended version to appear in Information and Computation, special issue on TCS'2000.

Cardelli, Luca and Gordon, Andrew D. (2000). Mobile ambients. *Theoretical Computer Science*, 240(1):177–213. Special Issue on Coordination, Daniel Le Métayer Editor.

Coppo, Mario, Dezani-Ciancaglini, Mariangiola, Giovannetti, Elio, and Pugliese, Rosario (2004). Dynamic and local typing for mobile ambients. Research report, Dipartimento di Sistemi e Informatica, Università di Firenze. Available at `http://www.dsi.unifi.it/~pugliese /DOWNLOAD/dltma-full.pdf`.

Coppo, Mario, Dezani-Ciancaglini, Mariangiola, Giovannetti, Elio, and Salvo, Ivano (2003). M3: Mobility types for mobile processes in mobile ambients. In Harland, James, editor, *CATS 2003*, volume 78 of *ENTCS*. Elsevier.

Ferrari, Gianluigi, Moggi, Eugenio, and Pugliese, Rosario (2002). Guardians for ambient-based monitoring. In Sassone, Vladimiro, editor, *F-WAN*, volume 66 of *ENTCS*. Elsevier.

Hennessy, Mattew and Riely, James (2002). Resource Access Control in Systems of Mobile Agents. *Information and Computation*, 173:82–120.

Levi, Francesca and Sangiorgi, Davide (2003). Controlling interference in Ambients. *Transactions on Programming Languages and Systems*, 25(1):1–69.