A distributed abstract machine for Safe Ambients

Davide Sangiorgi[*]
INRIA Sophia-Antipolis, France

Andrea Valente
Università di Torino, Italy

Paola Giannini
Università del Piemonte Orientale, Italy

January 7, 2003

Abstract

An abstract machine for a distributed implementation of an ambient calculus is presented. The abstract machine is different from, and simpler than, previous implementations of ambient-like calculi, mainly because: the underlying calculus is typed Safe Ambients rather than the untyped Ambient calculus and therefore does not present certain forms of interferences among processes (the *grave interferences*); the logical structure of an ambient system and its physical distribution are separated. The proof of correctness and a description of a distributed implementation of the abstract machine in Java are given.

1 Introduction

The Ambient calculus [4] is a model for mobile distributed computing. An ambient 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, $n[\,\mathtt{in}\,m.\,P \mid Q\,] \mid m[\,R\,] \longrightarrow m[\,n[\,P \mid Q\,] \mid R\,]$; an ambient to exit another ambient, $m[\,n[\,\mathtt{out}\,m.\,P \mid Q\,] \mid R\,] \longrightarrow n[\,P \mid Q\,] \mid m[\,R\,]$; a process to dissolve an ambient boundary thus obtaining access to its content, $\mathtt{open}\,n.\,P \mid n[\,Q\,] \longrightarrow P \mid Q$.

Several studies of the basic theory of the Ambient calculus have recently appeared, concerning for instance behavioural equivalences, types, logics, static analysis techniques [5, 6, 1, 7, 12]. In comparison, little attention has been given to implementations. The only implementations of Ambients we are aware of are Cardelli's [2, 3], and Fournet, Lévy and Schmitt's [9]. The latter, formalised as a translation of Ambients into the distributed Join Calculus, is the only distributed implementation. Although ingenious, the algorithms that these implementations use for simulating the ambient reductions are fairly complex.

One of the difficulties of a distributed implementation of an ambient-like language is that each movement operation involves ambients on different hierarchical levels. For instance, the ambients affected by an `out` operation are the moving ambient, and its initial and its final parent; at the beginning they reside on three different levels. In [2, 3] locks are used to achieve a synchronisation among all ambients affected by a movement. In a distributed setting, however, this lock-based policy can be expensive. For instance, the serialisations introduced diminish the parallelism of the whole system. In [9] the synchronisations are simulated by means of protocols of asynchronous

[*]Sangiorgi's research has been partially funded by the european project FET-Global computing "PROFUNDIS".

messages. The problems of implementation have been a restraint to the development of programming languages based on Ambients and to experimentation of Ambients on concrete examples. In our opinion, implementation is one of the aspects of Ambients that most need investigations.

In this paper we study an abstract machine for a distributed implementation of an ambient-like calculus, called PAN, *Pervasive Ambient Network*. The calculus we implement is the typed Safe Ambients [11] (SA) rather than untyped Ambients. SA is a variant of the original Ambient calculus that eliminates certain forms of interference in ambients, the grave interferences. They are produced when an ambient tries to perform two different movement operations at the same time, as for instance $n[\,\mathtt{in}\, h.\, P \mid \mathtt{out}\, n.\, Q \mid R\,]$. The control of mobility is obtained in SA by a modification of the syntax and a type system. The type system allows two kinds of ambients: *single threaded* and the *immobile* ambients, and a limited interaction between the two. In [11] the absence of grave interferences is used to develop an algebraic theory and prove the correctness of some examples. One of the contributions of this paper is to show that the absence of grave interferences also brings benefits in the implementations.

The algorithms of our abstract machine are quite different from, and simpler than, those of [2, 3, 9], mainly for two reasons.

The first – the most important — is that the calculus that we actually take is typed SA rather than untyped Ambients. The absence of grave interferences, allows us to obtain important simplifications in the structure of the abstract machine, and the protocols needed to realize the movement operations.

The second reason is that in our abstract machine there is a separation between the logical structure of an ambient system and its physical distribution. Exploiting this, the interpretation of the movement associated to the various operations is reversed: the movement of the `open` capability is physical, that is, the location of some processes changes, whereas that of `in` and `out` is only logical, that is, some hierarchical dependencies among ambients may change, but not their physical location. Intuitively, `in` and `out` are acquisition of access rights, and `open` is exercise of them. We may say that the movement given by `in` and `out` is first-order (some names move, and some pointers are updated), whereas the movement given by `open` is higher order (some terms move).

Our algorithms are formulated as an abstract machine. This leads to a relatively simple proof of correctness with respect to the one of the Join Calculus implementation. Moreover, the machine is independent of a specific implementation language, and can thus be used as a basis for implementations on different languages. In the paper we present one such implementation, written in Java.

As a final consideration we would point out that with PAN it becomes evident that Ambient-based calculi really need two kind of operations, namely *physical* and *logical* ones. Creation of new subambients and *open* belong to the first category, being related to actual creation and destruction of unique elements in the tree of ambients; logical operations are the ones that only reshape the topology of the tree, ie `in` and `out`. Many of the advantages of PAN came from the acceptance of this underlying two-level model in the abstract machine; e.g., our located agent are *spawned* and given a *unique id*, while in the syntax of the original calculus there is a simple statement about the existence of an ambient inside another, moreover in-fligth messages in PAN are first-class entities, existing outside their sender and reciever during their life.

The paper is organized as follows. In Section 2 we introduce the syntax, semantics and relevant properties of the SA calculus. In Section 3 we introduce informally how the movement operations are realized in the abstract machine whose formal syntax and reduction relation are defined in Section 4. The proof of correctness for the fragment of SA including only single threaded ambients is given in Section 5, and in Section 6 the proof is extended to the whole SA, which includes also immobile ambients. In Section 7 we describe the Java implementation of the abstract machine.

Comparisons with the Join implementation are considered in Section 8, and in Section 9 we outline improvements and optimizations to the abstract machine and its implementation.

2 Safe Ambients: syntax and semantics

We briefly describe typed Safe Ambient (SA), from [11]. In the reduction rules of the original Ambient calculus, mentioned in Section 1, an ambient may enter, exit, or open another ambient. The second ambient undergoes the action; it has no control on *when* the action takes place. In SA this is rectified: *coactions* $\overline{\mathtt{in}}\, n, \overline{\mathtt{out}}\, n, \overline{\mathtt{open}}\, n$ are introduced with which any movement takes place only if both participants agree. The syntax of SA is presented in Table 1. Expressions that are not variables or names are the *capabilities*.

These are divided into *actions* ($\mathtt{in}\, n, \mathtt{out}\, n, \mathtt{open}\, n$) and *coactions* ($\overline{\mathtt{in}}\, n, \overline{\mathtt{out}}\, n, \overline{\mathtt{open}}\, n$). We often omit the trailing $\mathbf{0}$ in processes $M.\mathbf{0}$. Parallel composition has the least syntactic precedence, thus $m[\, M.P \mid Q\,]$ reads $m[\,(M.P) \mid Q\,]$. An ambient, or a parallel composition, or variable, is *unguarded* if it is not underneath a capability or an abstraction. In a recursion $\mathtt{rec}\, X.P$, the recursion variable X should be guarded in P. For simplicity of presentation we omit path expressions in the syntax.

Below are the reduction axioms: those for movement, that for exiting, that for opening, and the communication rule (communication is asynchronous, takes place inside ambients, and is anonymous—it does not use channel or process names):

$$n[\,\mathtt{in}\, m.P_1 \mid P_2\,] \mid m[\,\overline{\mathtt{in}}\, m.Q_1 \mid Q_2\,] \longrightarrow m[\,n[\,P_1 \mid P_2\,] \mid Q_1 \mid Q_2\,] \qquad \text{[R-IN]}$$

$$m[\,n[\,\mathtt{out}\, m.P_1 \mid P_2\,] \mid \overline{\mathtt{out}}\, m.Q_1 \mid Q_2\,] \longrightarrow n[\,P_1 \mid P_2\,] \mid m[\,Q_1 \mid Q_2\,] \qquad \text{[R-OUT]}$$

$$\mathtt{open}\, n.P \mid n[\,\overline{\mathtt{open}}\, n.Q_1 \mid Q_2\,] \longrightarrow P \mid Q_1 \mid Q_2 \qquad \text{[R-OPEN]}$$

$$\langle M \rangle \mid (x)P \longrightarrow P\{M/x\} \qquad \text{[R-MSG]}$$

The inference rules allow a reduction to occur underneath a restriction, a parallel composition, and inside an ambient. Moreover, the structural congruence relation ($\equiv$) can be applied before a reduction step. Structural congruence is used to bring the participants of a potential interaction into contiguous positions; its definition is standard, and includes rules for commuting the positions of parallel components, for stretching the scope of a restriction, for unfolding recursions.

$$\epsilon.P \longrightarrow P \qquad \text{[R-EPS]}$$

$$\mathtt{rec}\, X.P \longrightarrow P\{\mathtt{rec}\, X.P/X\} \qquad \text{[R-REC]}$$

$$\frac{P \longrightarrow P'}{(\boldsymbol{\nu} n)\, P \longrightarrow (\boldsymbol{\nu} n)\, P'} \qquad \text{[R-RES]}$$

$$\frac{P \longrightarrow P'}{P \mid Q \longrightarrow P' \mid Q} \qquad \text{[R-PAR]}$$

$$\frac{P \longrightarrow P'}{n[\,P\,] \longrightarrow n[\,P'\,]} \qquad \text{[R-AMB]}$$

$$\frac{P \equiv P' \qquad P' \longrightarrow P'' \qquad P'' \equiv P'''}{P \longrightarrow P'''} \qquad \text{[R-STRUCT]}$$

We write $\Longrightarrow$ for the reflexive and transitive closure of $\longrightarrow$. The use of coactions, in the syntax and operational rules, is the only difference between (untyped) SA and and the original Ambient calculus.

$n, m, \ldots$			*Names*	x, y, z			*Variables*
				X, Y, Z			*Recursion variables*
			Expressions				*Processes*
M, N	$:=$	x	*(variable)*	P, Q, R	$:=$	$\mathbf{0}$	*(nil)*
	$\mid$	n	*(name)*		$\mid$	$P \mid Q$	*(parallel)*
	$\mid$	$\mathtt{in}\, M$	*(enter)*		$\mid$	$(\boldsymbol{\nu} n)\, P$	*(restriction)*
	$\mid$	$\overline{\mathtt{in}}\, M$	*(allow enter)*		$\mid$	$M.\, P$	*(prefixing)*
	$\mid$	$\mathtt{out}\, M$	*(exit)*		$\mid$	$M[\, P\,]$	*(ambient)*
	$\mid$	$\overline{\mathtt{out}}\, M$	*(allow exit)*		$\mid$	$\langle M \rangle$	*(value message)*
	$\mid$	$\mathtt{open}\, M$	*(open)*		$\mid$	$(x)P$	*(abstraction)*
	$\mid$	$\overline{\mathtt{open}}\, M$	*(allow open)*		$\mid$	X	*(recursion variable)*
					$\mid$	$\mathtt{rec}\, X.\, P$	*(recursive process)*

Table 1: The syntax of Safe Ambients

Up to structural congruence, every ambient in a term can be rewritten into a normal form

$$\boldsymbol{\nu} \tilde{p}\, n[\, P_1 \mid \ldots \mid P_s \mid m_1[\, Q_1\,] \mid \ldots \mid m_r[\, Q_r\,]\,] \tag{1}$$

where the P_i's ($i = 1..s$) are neither ambients nor parallel compositions. In this case, $P_1, \ldots, P_s$ are the *local processes*, and $m_1[\, Q_1\,], \ldots, m_r[\, Q_r\,]$ are the *subambients* of the ambient n.

SA has two main kinds of types: *single-threaded* and *immobile*. We consider them separately. We begin with the single-threaded types, which we informally describe below. We consider immobility types in Section 6.

The capabilities of the local processes of an ambient control the activities of that ambient. In an untyped (or immobile) ambient such control is distributed over the local processes: any of them may exercise a capability. In a *single-threaded* (ST) ambient, by contrast, at any moment at most *one* local process has the control thread, and may therefore use a capability. An ST ambient n is willing to engage in at most one interaction at a time with external or internal ambients. Inside n, however, several activities may take place concurrently: for instance, a subambient may reduce, or two subambients may interact with each other. Thus, if an ambient n is ST, the following situation, where at least two local processes are ready to execute a capability, cannot occur: $n[\, \mathtt{in}\, m.\, P \mid \mathtt{out}\, h.\, Q \mid R\,]$. The control thread may move between processes local to an ST ambient by means of an open action. Consider, for instance, a reduction

$$n[\, \mathtt{open}\, m.\, P \mid m[\, \overline{\mathtt{open}}\, m.\, Q\,]\,] \longrightarrow n[\, P \mid Q\,] \tag{2}$$

where n and m are ST ambients. Initially $\mathtt{open}\, m.\, P$ has the control thread over n, and $\overline{\mathtt{open}}\, m.\, Q$ over m. At the end, m has disappeared; the control thread over n may or may not have moved from P to Q, depending on the type of m. If the movement occurs, Q can immediately exercise a capability, whereas P cannot; to use further capabilities within n, P will have to get the thread back.

For simplicity, we assume here a strong notion of ST, whereby a value message $\langle M \rangle$ never carries the thread. In [11] a weaker notion is used, where also messages may carry the thread. In this case, the control thread over an ambient may move, other than by an open operation, as a result of the consumption of a value by an abstraction. The results in our paper can be adapted to this weaker notion.

In the remainder, all processes are assumed to be well-typed, and closed, i.e., without free variables.

Let $P \equiv P_1 \mid \ldots \mid P_s$, where the P_i's $(i = 1..s)$ are not parallel compositions, be an SA term. The *unguarded prefixes/capabilities* of P are the $\mathtt{cap} \in \{\mathtt{in}, \overline{\mathtt{in}}, \mathtt{out}, \overline{\mathtt{out}}, \mathtt{open}, \overline{\mathtt{open}}\}$ such that for some i, $P_i = (\boldsymbol{\nu}\tilde{h})\ \mathtt{cap}.P_i'$. A term P' is an *unboxed subcomponent* of P if P' is a subterm P such that P' is not subterm of an ambient $m[\,Q\,]$. The main property that ST ambients have is expressed from the following statement (a simple variant of a theorem in [11]).

Proposition 2.1 *If P is a well-typed SA term and the subterm $n[\,Q\,]$ is a single threaded ambient, then no unboxed subcomponent of Q has more than one unguarded capability.*

As proved in [11], well-typedness is preserved under reduction (Subject Reduction), so the previous property is true under reduction.

3 The abstract machine, informally

We describe the data structures and the algorithms of the abstract machine, called PAN. PAN separates between the logical and the physical distribution of the ambients. The logical distribution is given by the tree structure of the ambient syntax. (an ambient can contain other ambients). The physical distribution is given by the association of a location to each ambient.

In PAN, an ambient named n is represented as a *located ambient* $h{:}\,n[\,P\,]_k$, where h is the location, or site, at which the ambient runs, k is the location of the parent of the ambient, and P collects the processes local to the ambient. While the same name may be assigned to several ambients, a location univocally identifies an ambient; it can be thought of as its physical address.

A tree of ambients is rendered, in PAN, by the parallel composition of the (unguarded) ambients in the tree. In this sense, the physical and the logical topology are separated: the space of physical locations is flat, and each location hosts at most one ambient, each ambient resides at a distinct physical location (this gives us the physical distribution), but each ambient knows the location at which its parent resides. (this gives us the logical topology). For instance, an SA term $n[\,P_1 \mid P_2 \mid m_1[\,Q_1\,] \mid m_2[\,Q_2\,]\,]$, where P_1 and P_2 are the local processes of n, and Q_i $(i = 1, 2)$ is a local process of m_i (i.e., m_i has no subambients), becomes in PAN:

$$h{:}\,n[\,P_1 \mid P_2\,]_{\mathtt{root}} \parallel k_1{:}\,m_1[\,Q_1\,]_h \parallel k_2{:}\,m_2[\,Q_2\,]_h$$

where h, k_1, k_2 are different location names, $\mathtt{root}$ is a special name indicating the outermost location, and $\parallel$ is parallel composition of located ambients. (The above configuration is actually obtained after two creation steps, in which the root ambient spawns off the two ambients located at k_1 and k_2.) Since ambients may run at different physical sites, they communicate with each other by means of *asynchronous* messages.

All the actions ($\mathtt{in}$, $\mathtt{out}$, and $\mathtt{open}$) can modify the logical distribution. Only $\mathtt{open}$, however, can modify the physical distribution. The algorithms that PAN adopts to model reduction in SA are based on 3 steps: first, a *request* message is sent upward, from a child ambient that wants to move (logically or physically) to its parent; second, a *match* is detected by the parent itself; third, a *completion* message is sent back to the child, for its relocation. The only exception is the algorithm for $\mathtt{open}$, where a further message is needed to migrate the child's local processes to the parent. These steps are sketched in Figures 1-3, where a, b, c represent three ambients, a straight line represents a pointer from an ambient to its parent, and a curved line represents the sending of a message. Thus in Figure 1, at the beginning a and b are sibling ambients and c is their parent. This figure illustrates an R-IN reduction in which a becomes a child of b. In the first phase, a demands to enter b (precisely, if n is the name of b, then a demands of entering an ambient with name n), and b accepts an ambient in. For this, a and b send requests $\mathtt{in}$ and $\overline{\mathtt{in}}$ to their parent c (the actual messages may also contain the name and location of the sender; these are not shown in the figures). In the second phase, c sees that two matching requests have been sent and authorises the movement. Finally, in the third phase, c sends completion messages to a and b. The message sent to a also contains the location of b, which a will use to update its parent field. An ambient that has sent a request to its parent but has not yet received an acknowledgement back, goes into a *wait* state, in which it will not send further requests. In the figures, this situation is represented by a circle that encloses the ambient. An ambient in a *wait* state, however, can still receive and answer requests from its children and can perform local communications.

Figure 2 sketches an R-OUT reduction. In the first phase, ambient a demands its parent b to exit. When b authorises the movement (phase 2), it sends a an acknowledgement containing the location of the parent of b, namely c, and upon receiving this message (phase 3) a updates its parent field. Note that the grandparent ambient c is not affected by the dialog between a and b. Figure 3 sketches an R-OPEN reduction. Ambient a accepts to be opened, and thus notifies its parent c. If a matching capability exists, that is, one of the processes local to c demands to open a, then c authorises a to migrate its local processes into c. Ambient a then becomes a forwarder ($a \triangleright c$ in the figure) whose job is just to forward any messages sent to a on to c. Such a forwarder is necessary, in general, because a may have subambients, which would run at different locations and which would send their requests of movement to a. (With appropriate optimisations, forwarders can be further simplified, or even altogether removed; see the discussion in Section 9.)

The other reduction rule of SA, the R-MSG rule, is an interaction between two processes local to the same ambient. In PAN, this is simulated by essentially the same rule: no messages need to be exchanged between locations.

Using R-OPEN, rather than R-IN or R-OUT, for the physical movements may appear counterintuitive. One should however bear in mind that, in an ambient-like formalism, entering and exiting ambients is not very useful without opening some ambients. The processes local to an ambient a can interact with the processes local to ambient b only if one of the ambients moves inside the other *and* is opened. For instance, suppose we want to model a traveller that starts on a base site, goes to distant server, interacts with its services, and then comes back reporting a result. For simplicity, we suppose that interaction just consists in loading a value v that is also the final result. The distant server is defined thus:

$$\mathit{SERVER} \stackrel{\text{def}}{=} s[\, \overline{\mathtt{in}}\, s.\, \mathtt{open}\, go.\, \overline{\mathtt{out}}\, s.\, P \mid \langle v \rangle \,]$$

The base site and the traveller T are:

$$\begin{aligned} \mathit{BASE} &\stackrel{\text{def}}{=} n[\, T \mid \overline{\mathtt{out}}\, n.\, \overline{\mathtt{in}}\, n.\, \mathtt{open}\, \mathit{return}.\, (x)\; R \,] \\ T &\stackrel{\text{def}}{=} go[\, \mathtt{out}\, n.\, \mathtt{in}\, s.\, \overline{\mathtt{open}}\, go.\, (x)\; \mathit{return}[\, \mathtt{out}\, s.\, \mathtt{in}\, n.\, \overline{\mathtt{open}}\, \mathit{return}.\, \langle x \rangle \,]\,] \end{aligned}$$

We have:

$$\begin{aligned} \mathit{SERVER} \mid \mathit{BASE} &\Longrightarrow s[\, \overline{\mathtt{out}}\, s.\, P \mid \langle v \rangle \mid (x)\; \mathit{return}[\, \mathtt{out}\, s, \mathtt{in}\, n.\, \overline{\mathtt{open}}\, \mathit{return}.\, \langle x \rangle \,]\,] \mid \\ &\qquad n[\, \mathtt{in}\, n.\, \mathtt{open}\, \mathit{return}.\, (x)\; R \,] \\ &\Longrightarrow s[\, P \,] \mid n[\, (x)\; R \mid \langle v \rangle \,] \\ &\longrightarrow s[\, P \,] \mid n[\, R\{v/x\} \,] \end{aligned} \tag{3}$$

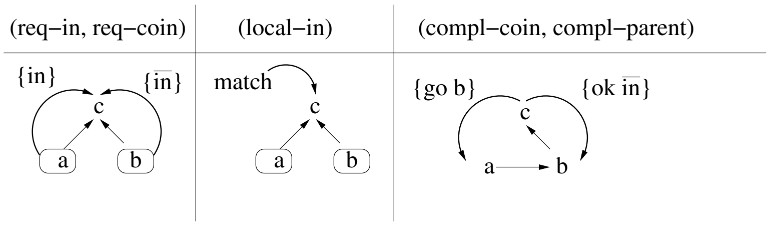

Figure 1: The simulation of the reduction R-IN in PAN

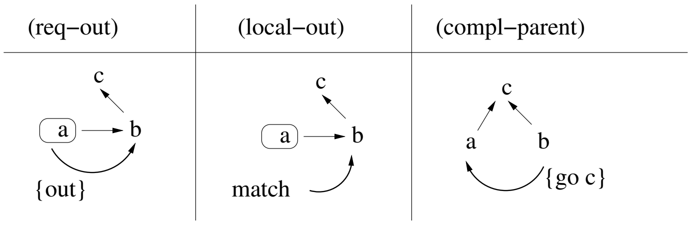

Figure 2: The simulation of the reduction R-OUT in PAN

In the PAN execution, the initial configuration is (omitting some local processes of s and n)

$$h\colon s[\ldots]_{\mathtt{root}} \parallel k\colon n[T \mid \ldots]_{\mathtt{root}}$$

and the first step is to spawn off a new location hosting the traveller, obtaining:

$$h\colon s[\ldots]_{\mathtt{root}} \parallel k\colon n[\ldots]_{\mathtt{root}} \parallel \ell\colon go[\ldots]_{k}$$

In the reductions in (3), two R-OPEN are executed. When this is modeled in PAN, in the first R-OPEN the (derivative of the) traveller physically moves into the distant server so to load v; in the second, the (derivative of the) traveller comes back to deliver its result to R.

4 The abstract machine, formally

In this section we introduce the syntax and reduction semantics of the abstract machine.

Syntax. The syntax of PAN is shown in Table 4. A term of PAN, a *net*, is the parallel composition of *agents* and *messages*, with some names possibly restricted. An agent can be a *located ambient* or a *forwarder*. Located ambients are the basic unit of PAN, and represent ambients of SA with

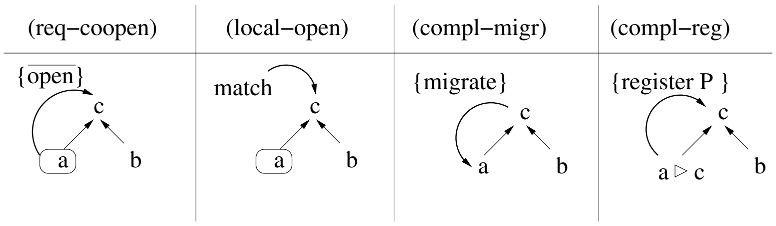

Figure 3: The simulation of the reduction R-OPEN in PAN

their local processes. A located ambient becomes a forwarder when opened. Messages are of two forms: *requests* and *completions*. The former are the messages that an ambient sends to its parent to request a movement operation; the latter are the messages that the parent sends back to the children to complete the operation. An `open` needs however a third completion message (a `register` message) in which the processes local to a child are migrated to its parent. The syntax of the processes inside located ambients is similar to that of processes in SA. The only additions are: the prefix $\texttt{wait}.P$, which appears in an ambient when this has sent a request to its parent but has not received an answer yet; and the requests, which represent messages received from the children and not yet served. We use A to range over nets.

Remark 4.1 *Note that the typing rules for single-threadness (ST) extend to* `wait`*, thus* $\texttt{wait}.P \mid M.Q$ *is disallowed in a ST ambient. In particular, in an ST ambient, there can be at most one* `wait` *prefix.*

Semantics. The reduction relation of PAN, $\longmapsto$, from nets to nets, is defined by the rules below. The axioms are divided into five groups, for, respectively: reductions local to an ambient; creation of new ambients and new restrictions; forwarding; consumption of request messages; emission of request messages; consumption of completion messages. There is finally a group of inference rules. The rules for local reductions, and the associated inference rule PAR-PROC, have a special format. We write

$$P \xrightarrow[h:n]{k} Q \gg \widetilde{Msg}$$

to mean a process P, local to an ambient n that is located at h, and whose parent is located at k, becomes Q and, as a side effect, the messages in $\widetilde{Msg}$ are generated. We use $\widetilde{Msg}$ to indicate a possibly empty parallel composition of messages. When n or h or k are unimportant, we replace them with $-$, as in $P \xrightarrow[-:n]{k} Q \gg \widetilde{Msg}$. When all the three of them are unimportant, we simply write $P \xrightarrow[-:-]{-} P' \gg \widetilde{Msg}$.

The local reductions correspond to the reduction rule of the SA calculus. The rule LOCAL-COM is simply local message exchange, no completion is generated. In the rule LOCAL-IN both (action and coaction) request messages are consumed and we generate two completions: one for the ambient requesting the action that has to change its parents, with the indication of the location of the new parent (k), and the other for the ambient requesting the coaction saying that the process can consume the coaction. In rule LOCAL-OUT, since no request message is generated for the coaction, it is generated only one completion message for the ambient requesting to be moved (again the completion contains the location of the new parent). The reduction, however, must occur in the ambient containing the coaction, and the coaction is consumed allowing the process P to go on. For the LOCAL-OPEN again the matching happen in the ambient containing the coaction. However, the process P, is stopped, and has to wait for the register completion to continue. The register completion will contain the local processes of the ambient that is opened.

The creation rule, NEW-LOCAMB assigns to the ambient m subambient of n a new location k. The logical structure is realized by assigning the location of n (h) as father of the location k. The rule NEW-RES enlarges the scope of a restriction. The rule FW-MSG, if there is a forwarder from h to k, redirect the messages sent to h to k, so that in the rule CONSUME-REQ such messages may enter the right ambient (and be processed by the local reduction).

The rules for emission of requests, generate the request messages corresponding to all the actions and coactions except for `open`, and $\overline{\texttt{out}}$. The request message is sent to the location of the father of the ambient and contains the location of the ambient requesting the action. This allows the father

$$a, b, .. \in \mathit{Names} \qquad h, k, .. \in \mathit{Locations} \qquad p, q, .. \in \mathit{Names} \cup \mathit{Locations}$$

$$
\begin{array}{rcll}
 & & & \textit{Nets} \\
A & := & \mathbf{0} & \text{(empty)} \\
 & | & \mathit{Agent} & \text{(agent)} \\
 & | & h\{\mathit{MsgBody}\} & \text{(message)} \\
 & | & A_1 \parallel A_2 & \text{(composition)} \\
 & | & (\boldsymbol{\nu} p\,)A & \text{(restriction)}
\end{array}
\qquad
\begin{array}{rcll}
 & & & \textit{Agents} \\
\mathit{Agent} & := & h \rhd k & \text{(forwarder)} \\
 & | & h\colon n[\,P\,]_k & \text{(located ambient)} \\
 & & & \textit{Message body} \\
\mathit{MsgBody} & := & \mathit{Request} & \text{(request)} \\
 & | & \mathit{Completion} & \text{(completion)}
\end{array}
$$

$$
\begin{array}{rcll}
\mathit{Request} & := & \mathtt{in}\, n, h & \text{(the agent at } h \text{ wants to enter } n\text{)} \\
 & | & \overline{\mathtt{in}}\, n, h & \text{(the agent at } h\text{, named } n\text{, accepts someone in)} \\
 & | & \mathtt{out}\; n, h & \text{(the agent at } h \text{ wants to go out of } n\text{)} \\
 & | & \overline{\mathtt{open}}\, n, h & \text{(the agent at } h\text{, named } n\text{, accepts to be opened)}
\end{array}
$$

$$
\begin{array}{rcll}
\mathit{Completion} & := & \mathtt{go}\, h & \text{(change the parent to be } h\text{)} \\
 & | & \mathtt{OK}\overline{\mathtt{in}} & \text{(request } \overline{\mathtt{in}} \text{ accepted)} \\
 & | & \mathtt{migrate} & \text{(request } \overline{\mathtt{open}} \text{ accepted)} \\
 & | & \mathtt{register}\; P & \text{(add } P \text{ to the local processes)}
\end{array}
$$

Process-related syntax:

$$
\begin{array}{rcl}
P & := & \mathbf{0} \\
 & | & P_1 \mid P_2 \\
 & | & (\boldsymbol{\nu} n\,)P \\
 & | & M.\,P \\
 & | & M[\,P\,] \\
 & | & \langle M \rangle \\
 & | & (x)\; P \\
 & | & X \\
 & | & \mathtt{rec}\; X.\,P \\
 & | & \mathtt{wait}.\,P \\
 & | & \{\mathit{Request}\}
\end{array}
\qquad\qquad
\begin{array}{rcl}
M & := & x \\
 & | & n \\
 & | & \mathtt{in}\, M \\
 & | & \overline{\mathtt{in}}\, M \\
 & | & \mathtt{out}\; M \\
 & | & \overline{\mathtt{out}}\; M \\
 & | & \mathtt{open}\, M \\
 & | & \overline{\mathtt{open}}\, M
\end{array}
$$

Figure 4: The syntax of PAN

which does not know the location of its children, to do the matching and send the completion to the right location. Notice that the emission of requests for the coactions ($\overline{\texttt{in}}\, n$, $\overline{\texttt{open}}\, n$) is done only if the capability is an unguarded prefix of the environment n. For the $\overline{\texttt{out}}\, n$ coaction, for which no request message is generated, the same property is insured by the fact that the local reduction LOCAL-OUT must happen in the ambient n.

The rules for consumption of completions realize the actual movement operations, consuming the completion message. The COMPL-PARENT rule modifies the father of the ambient located at h to be h'. This completion was sent as the result of and `in` or `out` request. The process Q, which was the one prefixed by the corresponding capability, is allowed to continue. For the $\overline{\texttt{in}}$ completion we just allow the process Q, which was the one prefixed by the $\overline{\texttt{in}}$ capability, to go on. The process was stopped to avoid the possibility that it may flood the father with requests, before satisfying the current one. The COMPL-MIGR rule dissolves the ambient n located at h leaving a forwarder to the location of its father, to whom it sends (through the `register` completion) the local processes of n. The processes will be added to the local processes of the father ambient via the COMPL-REG rule. In the implementation, the completion messages are first read, and then consumed, whereas here these two phases are merged. This abstraction is completely harmless – changing the machine to reflect it quite straightforward.

The inference rules allow to execute local reductions inside located ambients, restrictions, and parallel compositions. First the side condition of rule PAR-PROC insures that local reductions and emissions of requests may be done only in located ambients not containing subambients. So subambients of an ambients are activated as soon as possible, before any local reduction takes place (here we exploit the fact that recursions are guarded, otherwise there could be an infinite number of ambients to create). The rule NEW-LOCAMB, therefore, has the precedence on all the other rules. Rule PROC-AGENT transforms a message (request or completion) into an agent. For instance, if $P \xrightarrow[h:n]{k} Q \gg \widetilde{Msg}$, then, using PROC-AGENT and PAR-AGENT, we have, for any net A:

$$A \parallel h{:}n[\,P\,]_k \longmapsto A \parallel h{:}n[\,Q\,]_k \parallel \widetilde{Msg}$$

The inference rule STRUCT-CONG make use of the structural congruence relation $\equiv$, whose definition is similar to that for SA:

Definition 4.2 (structural congruence) Structural congruence *is the smallest congruence* $\equiv$ *such that:*

1. *parallel composition and* $\mathbf{0}$ *form an abelian monoid (on processes as well as on agents);*
2. $\boldsymbol{\nu}p\,(A_1) \parallel A_2 \equiv \boldsymbol{\nu}p\,(A_1 \parallel A_2)$ *if p not free in A_2, and* $\boldsymbol{\nu}n\,(P_1) \mid P_2 \equiv \boldsymbol{\nu}p\,(P_1 \mid P_2)$ *if n not free in P_2;*
3. $\boldsymbol{\nu}p\,\mathbf{0} \equiv \mathbf{0}$, *and* $\boldsymbol{\nu}p\,\boldsymbol{\nu}q\,P \equiv \boldsymbol{\nu}q\,\boldsymbol{\nu}p\,P$*;*
4. $\texttt{rec}\,X.\,P \equiv P\{^{\texttt{rec}\,X.\,P}/_X\}$.

Remark 4.3 *In our syntax and semantics, we have omitted the productions ϵ and $M_1.\,M_2$ for the capabilities, but extending SA and PAN with these primitives is straightforward. Simply add the following rule to the structural congruence*

$$(M_1.\,M_2).\,P \equiv M_1.\,M_2.\,P$$

Local reductions

$$\langle M \rangle \mid (x).\, P \xrightarrow[-:-]{-} P\{{}^{M}\!/\!_{x}\} \gg \mathbf{0} \qquad \text{[LOCAL-COM]}$$

$$\{\texttt{in}\, n, h\} \mid \{\overline{\texttt{in}}\, n, k\} \xrightarrow[-:-]{-} \mathbf{0} \gg h\{\texttt{go}\, k\} \parallel k\{\texttt{OK}\overline{\texttt{in}}\} \qquad \text{[LOCAL-IN]}$$

$$\{\texttt{out}\, n, h\} \mid \overline{\texttt{out}}\, n.\, P \xrightarrow[-:n]{k} P \gg h\{\texttt{go}\, k\} \qquad \text{[LOCAL-OUT]}$$

$$\texttt{open}\, n.\, P \mid \{\overline{\texttt{open}}\, n, h\} \xrightarrow[-:-]{-} \texttt{wait}.\, P \gg h\{\texttt{migrate}\} \qquad \text{[LOCAL-OPEN]}$$

Creation

$$h{:}\, n[\, m[\, P\,] \mid Q\,]_{h'} \longmapsto h{:}\, n[\, Q\,]_{h'} \parallel \boldsymbol{\nu} k\, (k{:}\, m[\, P\,]_{h}) \qquad \text{[NEW-LOCAMB]}$$

$$h{:}\, n[\, \boldsymbol{\nu} m\, P\,]_{k} \longmapsto \boldsymbol{\nu} m\, (h{:}\, n[\, P\,]_{k}) \qquad \text{[NEW-RES]}$$

Forwarder

$$h \rhd k \parallel h\{\mathit{MsgBody}\} \longmapsto h \rhd k \parallel k\{\mathit{MsgBody}\} \qquad \text{[FW-MSG]}$$

Consumption of request messages

$$h{:}\, n[\, P\,]_{h'} \parallel h\{\mathit{Request}\} \longmapsto h{:}\, n[\, P \mid \{\mathit{Request}\}\,]_{h'} \qquad \text{[CONSUME-REQ]}$$

Emission of request messages (should be $h \neq \texttt{root}$)

$$\texttt{in}\, m.\, P \xrightarrow[h:-]{k} \texttt{wait}.\, P \gg k\{\texttt{in}\, m, h\} \qquad \text{[REQ-IN]}$$

$$\overline{\texttt{in}}\, n.\, P \xrightarrow[h:n]{k} \texttt{wait}.\, P \gg k\{\overline{\texttt{in}}\, n, h\} \qquad \text{[REQ-COIN]}$$

$$\texttt{out}\, m.\, P \xrightarrow[h:-]{k} \texttt{wait}.\, P \gg k\{\texttt{out}\, m, h\} \qquad \text{[REQ-OUT]}$$

$$\overline{\texttt{open}}\, n.\, P \xrightarrow[h:n]{k} \texttt{wait}.\, P \gg k\{\overline{\texttt{open}}\, n, h\} \qquad \text{[REQ-COOPEN]}$$

Consumption of completion messages

$$h{:}\, n[\, P \mid \texttt{wait}.\, Q\,]_{k} \parallel h\{\texttt{go}\, h'\} \longmapsto h{:}\, n[\, P \mid Q\,]_{h'} \qquad \text{[COMPL-PARENT]}$$

$$h{:}\, n[\, P \mid \texttt{wait}.\, Q\,]_{k} \parallel h\{\texttt{OK}\overline{\texttt{in}}\} \longmapsto h{:}\, n[\, P \mid Q\,]_{k} \qquad \text{[COMPL-COIN]}$$

$$h{:}\, n[\, P \mid \texttt{wait}.\, Q\,]_{k} \parallel h\{\texttt{migrate}\} \longmapsto h \rhd k \parallel k\{\texttt{register}\, P \mid Q\} \qquad \text{[COMPL-MIGR]}$$

$$h{:}\, n[\, P \mid \texttt{wait}.\, Q\,]_{k} \parallel h\{\texttt{register}\, R\} \longmapsto h{:}\, n[\, P \mid Q \mid R\,]_{k} \qquad \text{[COMPL-REG]}$$

Inference rules

$$\frac{P \xrightarrow[h:n]{k} P' \gg \widetilde{Msg} \qquad Q \text{ does not have unguarded ambients}}{P \mid Q \xrightarrow[h:n]{k} P' \mid Q \gg \widetilde{Msg}} \qquad \text{[PAR-PROC]}$$

$$\frac{P \xrightarrow[h:n]{k} P' \gg \widetilde{Msg}}{h{:}\, n[\, P\,]_{k} \longmapsto h{:}\, n[\, P'\,]_{k} \parallel \widetilde{Msg}} \qquad \text{[PROC-AGENT]}$$

$$\frac{A \longmapsto A'}{A \parallel B \longmapsto A' \parallel B} \qquad \text{[PAR-AGENT]}$$

$$\frac{A \longmapsto A'}{\boldsymbol{\nu} p\, A \longmapsto \boldsymbol{\nu} p\, A'} \qquad \text{[RES-AGENT]}$$

$$\frac{A \equiv A' \qquad A' \longmapsto A'' \qquad A'' \equiv A'''}{A \longmapsto A'''} \qquad \text{[STRUCT-CONG]}$$

Figure 5: Reduction Rules of PAN

5 Correctness of the abstract machine for single threaded ambients

In this section we present the proof of adeguacy of the translation of SA terms into PAN when the only ambients in a term are single threaded. In particular, we prove that if P is a well typed SA terms, P exhibit the same observable behavior as the PAN term which is its translation.

We first introduce some definitions. The *name* of a located ambient $h{:}n[\,P\,]_k$ is n, the *home location* is h, the *parent location* (of h) is k. The *home location* of a forwarder $h \triangleright k$ is h, and the *parent location* is k. The *target location* of a message $h\{\mathit{MsgBody}\}$ is h. The *source location* of a request $\{M, k\}$ (that can be part of a message or of a local process) is k. There is no source location in a completion message. A special location is `root`, indicating the outermost (logical) location. By convention, the ambient located at `root` has name `rootname` and parent `rootparent`. We can extract a relation among locations from a net A, whereby two locations are related if they are the home and the parent location of the same agent. We call this the *dependency relation on the locations of A*.

As for SA terms, also a located ambient has a normal form like equation (1). (Note that the subambients of a located ambient are not located.) A located ambient is in *wait state* if the local processes of the ambient have an unguarded `wait` prefix.

Since PAN separates between the logical and physical distribution of ambients, we need to make sure that the two are consistent. For instance, the graph of the dependencies among locations in the physical distribution, which represents the logical structure, should be a tree. We also need conditions that ensure that the `wait` prefix is used as described informally in previous sections. We therefore introduce the notion of well-formedness.

In the definition below, the dependency relation on the locations of A form is required to be a tree. Hence we can define a partial order $\leq$ on locations, where $h \leq k$ indicates that either $k = h$, or that k is between h and the root of the tree in the ordering.

Definition 5.1 (well-formedness) *A net A is* well-formed *if* `root` *and* `rootparent` *are the only free locations of A, and the following conditions are true for A (by alpha-conversion, we assume that all restricted locations and names are different from each other and from free locations and names):*

1. *The dependency relation on the locations of A forms a tree.*
2. *If there is a message $h\{\mathtt{go}\ k\}$, then $k \not\leq h$.*
3. *There is a special located ambient, called the root, located at* `root`*, with name* `rootname`*, and with parent* `rootparent`*. The name* `rootname` *and the location* `rootparent` *cannot appear anywhere else. The location* `root` *cannot be*
   (a) *the source of a request message,*
   (b) *the target of a completion message that is not a register.*
4. *For every location h, there is at most one agent located at h.*
5. *For every forwarder $h \triangleright k$, the home location h does not appear*
   (a) *as source location of a request message or a request process.*
   (b) *as target of a completion message*
6. *The prefix* `wait` *can only appear as unguarded prefix of a local process of a located ambient.*

7. *A located ambient with home location h is in wait state iff h appears exactly once in at most one of the following ways:*
   (a) *as the source of a request (in a message or in a local process);*
   (b) *as the target of a completion message;*
   (c) *as the ambient parent of the target of a migrate message.*
8. *A located ambient with home location h such that h is the target of a completion does not contain subambients.*
9. *No ambient (located or not) has an unboxed subterm with more than one unguarded prefix.*
10. *If there is a message $h\{\mathtt{register}\, R\}$ and its ambient target (up to $\equiv$) is $h{:}\, n[P_1 \mid \mathtt{wait}.\, P_2\,]_k$, then $P_1 \mid P_2 \mid R$ has at most one unguarded prefix, and it is not a* `wait`.
11. *If there is a message $h\{\mathtt{migrate}\}$ and the ambient at h (up to $\equiv$) is $h{:}\, n[Q_1 \mid \mathtt{wait}.\, Q_2\,]_{h'}$, and the ambient parent of h (up to $\equiv$) is $k{:}\, m[P_1 \mid \mathtt{wait}.\, P_2\,]_{k'}$ then $Q_1 \mid Q_2 \mid P_1 \mid P_2$ has at most one unguarded prefix.*
12. (a) *In every request message $k\{M, h\}$, location h is a child of k, and*
    (b) *if inside an ambient located at k there is a request $\{M, h\}$ then h is a child of k.*

Clause 4 implies that if there is a forwarder $h \rhd k$, then there is no ambient $h{:}\, n[\cdots]_k$ (located at h). Clauses 7, 9, 10, and 11 are for single-threaded agents only. We will see what to use for immobile ambients in Section 6.

Since in a well formed net the top location, `root`, is required not be a forwarder, we can define: the *target ambient* of a message $h\{\mathit{MsgBody}\}$ in a well-formed net A, as the least location k with $h \le k$ and such that the agent at k is a located ambient (not the home location of a forwarder). Moreover we define the relations *parent ambient* and *child ambient*. Given a well formed net A, the *parent ambient* of an ambient located at h is k if k is a located ambient, $h < k$, and for no located ambient h', such that $h' \neq h$ and $h' \neq k$, $h < h'$ and $h' < k$. Similarly, h is a *child ambient* of an agent k (not necessarily an ambient), if h is a located ambient, $h < k$, and for no located ambient h', such that $h' \neq h$ and $h' \neq k$, it holds that $h < h'$ and $h' < k$. Note that since the dependency relation forms a tree with `root` as the root, for all located ambients different from the ambient located at `root`, the parent ambient exists and it is unique, whereas there can be any number of children ambient (also zero).

Lemma 5.2 *If A is well-formed and $A \equiv A'$ then also A' is well formed.*

Proof: Straightforward. □

Definition 5.3 *A reduction $A \longmapsto A'$ is an* administrative reduction*, written $A \longmapsto_{\mathrm{adm}} A'$, if there is a derivation of $A \longmapsto A'$ in that does not use local reductions (axioms* LOCAL-COM*,* LOCAL-IN*,* LOCAL-OUT*, or* LOCAL-OPEN*).*

A reduction $A \longmapsto A'$ is a proper reduction*, written $A \longmapsto_{\mathrm{pr}} A'$, if there is a derivation of $A \longmapsto A'$ that uses one of the local reductions.*

As usual, $\Longmapsto_{\mathrm{adm}}$ and $\Longmapsto_{\mathrm{pr}}$ are the reflexive and transitive closures of $\longmapsto_{\mathrm{adm}}$ and $\longmapsto_{\mathrm{pr}}$, respectively.

We can give the following characterization of administrative and proper reductions.

- $A \longmapsto_{\text{adm}} A'$ iff $A \equiv \boldsymbol{\nu}\overline{h}\, A_1 \parallel A_2$, and $A' \equiv \boldsymbol{\nu}\overline{h}\, A_1' \parallel A_2$ and either:

  1. A_1 is the antecedent of one of the rules: [NEW-...], [FW-MSG], [CONSUME-REQ], [COMPL-...], and A_1' is the consequent, or
  2. $A_1 = h{:}\, n[\, \mathtt{cap}\, m.\, Q \mid Q'\,]_k$, where $\mathtt{cap} \in \{\mathtt{in}\,, \mathtt{out}\,\}$, Q' does not have unguarded ambients, and $A_1' = h{:}\, n[\, \mathtt{wait.}\, Q \mid Q'\,]_k \parallel k\{\mathtt{cap}\, m, h\}$, or
  3. $A_1 = h{:}\, n[\, \mathtt{cap}\, n.\, Q \mid Q'\,]_k$, where $\mathtt{cap} \in \{\overline{\mathtt{in}}\,, \overline{\mathtt{open}}\,\}$, Q' does not have unguarded ambients, and $A_1' = h{:}\, n[\, \mathtt{wait.}\, Q \mid Q'\,]_k \parallel k\{\mathtt{cap}\, n, h\}$

- $A \longmapsto_{\text{pr}} A'$ iff $A \equiv \boldsymbol{\nu}\overline{h}\, A_1 \parallel A_2$, and $A' \equiv \boldsymbol{\nu}\overline{h}\, A_1' \parallel A_2$ and either

  1. $A_1 = h{:}\, n[\, \langle M \rangle \mid (x).\, Q \mid Q'\,]_k$, and $A_1' = h{:}\, n[\, Q\{M/x\} \mid Q'\,]_k$, or
  2. $A_1 = h{:}\, n[\, \{\mathtt{in}\, m, h'\} \mid \{\overline{\mathtt{in}}\, m, k'\} \mid Q'\,]_k$, and $A_1' = h{:}\, n[\, Q'\,]_k \parallel h'\{\mathtt{go}\, k'\} \parallel k'\{\mathtt{OK}\overline{\mathtt{in}}\}$, or
  3. $A_1 = h{:}\, n[\, \{\mathtt{out}\, n, h'\} \mid \overline{\mathtt{out}}\ n.\, Q \mid Q'\,]_k$, and $A_1' = h{:}\, n[\, Q \mid Q'\,]_k \parallel h'\{\mathtt{go}\, k\}$, or
  4. $A_1 = h{:}\, n[\, \{\overline{\mathtt{open}}\, m, h'\} \mid \mathtt{open}\ m.\, Q \mid Q'\,]_k$, and $A_1' = h{:}\, n[\, \mathtt{wait.}\, Q \mid Q'\,]_k \parallel h'\{\mathtt{migrate}\}$

  where Q' does not have unguarded ambients.

The following lemma proves that administrative reductions on well formed nets are weakly confluent.

Lemma 5.4 *Suppose A is well-formed. If $A \longmapsto_{\text{adm}} A'$ and $A \longmapsto_{\text{adm}} A''$ then either $A' \equiv A''$, or there is A''' such that $A' \Longmapsto_{\text{adm}} A'''$ and $A'' \Longmapsto_{\text{adm}} A'''$.*

Proof: If the reductions of $A \longmapsto_{\text{adm}} A'$ and $A \longmapsto_{\text{adm}} A''$ involve non overlapping agents, the result is obvious. Therefore, let us consider the possible reductions involving overlapping agents. Let $A \equiv \boldsymbol{\nu}\overline{h}\, A_1 \parallel A_2$.

1. $A_1 = h{:}\, n[\, \mathtt{cap}\, m.\, Q \mid Q'\,]_k$, where $\mathtt{cap} \in \{\mathtt{in}\,, \mathtt{out}\,, \overline{\mathtt{in}}, \overline{\mathtt{open}}\,\}$, and $A \longmapsto_{\text{adm}} A'$ applying a rule [REQ-...] then

   - since clause 9 of Definition 5.1 holds for A, Q' cannot have unguarded prefix so neither a rule [COMPL-...], nor another rule [REQ-...] involving A_1 can be applied;
   - rule [NEW-LOCAMB] cannot be applied since Q' does not have unguarded ambients.

   The only conflict may arise when $A_2 = h\{\mathit{Request}\} \parallel A_3$, and rule [CONSUME-REQ] is applied to A_1 and $h\{\mathit{Request}\}$. In this case we would have

   - $A' \equiv \boldsymbol{\nu}\overline{h}\, A_1' \parallel A_2$ where $A_1' = h{:}\, n[\, \mathtt{wait.}\, Q \mid Q'\,]_k \parallel k\{\mathtt{cap}\, m, h\}$, and
   - $A'' \equiv A_1'' \parallel A_3$ where $A_1'' = h{:}\, n[\, \mathtt{cap}\, m.\, Q \mid Q' \mid \{\mathit{Request}\}\,]_k$

   Let $A''' = A_1' = h{:}\, n[\, \mathtt{wait.}\, Q \mid Q' \mid \{\mathit{Request}\}\,]_k \parallel k\{\mathtt{cap}\, m, h\} \parallel A_3$, $A' \longmapsto_{\text{adm}} A'''$ via [CONSUME-REQ] and $A'' \longmapsto_{\text{adm}} A'''$ via the corresponding [REQ-...].

2. $A_1 = h{:}\, n[\, m[\, P\,] \mid Q\,]_k$, and $A \longmapsto_{\text{adm}} A'$ applying a rule [NEW-LOCAMB]. In this case no rule [COMPL-...] involving A_1 can be applied, since clause 8 of Definition 5.1 holds for A. As in the previous case, the only conflict may arise with rule [CONSUME-REQ], and A''' can be defined in a similar way.

3. $\underline{A_1 = h \rhd k \parallel h\{\mathit{Request}\}}$, and $A \longmapsto_{\mathrm{adm}} A'$ applying a rule [FW-MSG]. From clause 4 of Definition 5.1, no ambient can be located at h so no other rule involving $h\{\mathit{Request}\}$ can be applied.

4. $\underline{A_1 = h{:}\,n[\,P \mid \mathtt{wait}.\,Q\,]_k \parallel h\{\mathit{Completion}\}}$, and $A \longmapsto_{\mathrm{adm}} A'$ applying any rule [COMPL-...]. First notice that, since clause 7 of Definition 5.1 holds for A, no other rule [COMPL-...] can be applied. However, if $A_2 = h\{\mathit{Request}\} \parallel A_3$ also rule [CONSUME-REQ] can be applied. When $\mathit{Completion} \neq \mathtt{migrate}$, applying first [CONSUME-REQ] and then the appropriate [COMPL-...] produces the same result as applying first [COMPL-...] and then [CONSUME-REQ].

   For $\mathit{Completion} = \mathtt{migrate}$, the situation is more complex. We have

   $$A_1 = h{:}\,n[\,P \mid \mathtt{wait}.\,Q\,]_k \parallel h\{\mathtt{migrate}\}.$$

   $A \longmapsto_{\mathrm{adm}} A'$ applying rule [COMPL-MIGR] where

   $$A' = h \rhd k \parallel k\{\mathtt{register}\,P \mid Q\} \parallel h\{\mathit{Request}\} \parallel A_3$$

   $A \longmapsto_{\mathrm{adm}} A''$ applying rule [CONSUME-REQ], where

   $$A'' = h{:}\,n[\,P \mid \mathtt{wait}.\,Q \mid \{\mathit{Request}\}\,]_k \parallel h\{\mathtt{migrate}\} \parallel A_3.$$

   From clause 7c of Definition 5.1, we have that $A_3 = k{:}\,m[\,P' \mid \mathtt{wait}.\,Q'\,]_{k'} \parallel A_4$. Therefore,

   $$\begin{array}{ll} A' \longmapsto_{\mathrm{adm}} h \rhd k \parallel k\{\mathtt{register}\,P \mid Q\} \parallel k\{\mathit{Request}\} \parallel A_3 & \text{with rule[FW-MSG]} \\ \quad \longmapsto_{\mathrm{adm}} h \rhd k \parallel k\{\mathit{Request}\} \parallel k{:}\,m[\,P' \mid Q' \mid P \mid Q\,]_{k'} \parallel A_4 & \text{with rule[COMPL-REG]} \\ \quad \longmapsto_{\mathrm{adm}} A''' & \text{with rule [CONSUME-REQ]} \end{array}$$

   where $A''' = h \rhd k \parallel k{:}\,m[\,P' \mid Q' \mid P \mid Q \mid \{\mathit{Request}\}\,]_{k'} \parallel A_4$, and

   $$\begin{array}{ll} A'' \longmapsto_{\mathrm{adm}} h \rhd k \parallel k\{\mathtt{register}\,P \mid Q \mid \{\mathit{Request}\}\} \parallel A_3 & \text{with rule[COMPL-MIGR]} \\ \quad \longmapsto_{\mathrm{adm}} A''' & \text{with rule[COMPL-REG]} \end{array}$$

   So $A' \Longmapsto_{\mathrm{adm}} A'''$ and $A'' \Longmapsto_{\mathrm{adm}} A'''$.

This concludes the proof. □

We now prove that well formedness is preserved by administrative reductions.

Lemma 5.5 *If A is well-formed, and $A \longmapsto_{\mathrm{adm}} A'$, then also A' is well-formed.*

Proof: By case analysis on the reduction $A \longmapsto_{\mathrm{adm}} A'$. For each case one has to consider all the clauses of well-formedness (Definition 5.1).
Let $A \equiv \boldsymbol{\nu}\overline{h}\,A_1 \parallel A_2$, and $A' \equiv \boldsymbol{\nu}\overline{h}\,A_1' \parallel A_2$. For the rule [FW-MSG] the clauses of well formedness hold for A' since they hold for A. For rule [CONSUME-REQ] notice only that clause 12b holds for A' since clause 12a holds for A, and for rule [NEW-...] since k is new so the dependency relation in A' still forms a tree. All the other clauses hold since they hold for A.
Consider the [REQ-...] rules:

- $A_1 = h{:}\,n[\,\mathtt{cap}\,m.\,Q \mid Q'\,]_k$, where $\mathtt{cap} \in \{\mathtt{in}, \mathtt{out}, \overline{\mathtt{in}}, \overline{\mathtt{open}}\}$, and Q' does not contain un unguarded ambient, and

- $A_1' = h{:}\,n[\,\texttt{wait}.\,Q \mid Q'\,]_k \parallel k\{\texttt{cap}\,m, h\}$.

Note that in A, h cannot be neither $\texttt{root}$ nor the source of a forwarder, therefore in A' clauses 3 and 5 hold. Since in A the process Q' cannot have any unguarded prefix, then clauses 8 and 9 hold in A'. Clause 7 holds for A' since it holds for A and A_1'. Finally since h is a child of k, clause 12a holds for A'. The other clauses hold since they hold for A.
If the rule applied is [COMPL-PARENT], since clause 2 holds for A, we have that the dependency relation in A' still forms a tree. Since clause 9 holds for A, P cannot have unguarded prefixes, and so clause 9 holds also for A'. Moreover, from clause 6 for A, such prefix cannot be $\texttt{wait}$. So also the resulting ambient in not in $\texttt{wait}$, and clauses 7, 9, and 12 hold for A'. The other clauses hold since they hold for A. Similarly for rule [COMPL-COIN].
For the rule [COMPL-MIG]:

- $A_1 = h{:}\,n[\,P \mid \texttt{wait}.\,Q\,]_k \parallel h\{\texttt{migrate}\}$, and
- $A_1' = h \rhd k \parallel k\{\texttt{register}\,P \mid Q\}$

All the clauses except for 5, 7, and 10 hold for A' because they hold A. Consider clause 5. Since clause 7 holds for A, the ambient h cannot be neither the source of a request, nor the target of a completion, so clause 5 holds for A'. Since clauses 7b and 7c hold for A, we know that in A_2 there should be an agent in $\texttt{wait}$ located at k. So clause 7 is verified for A'. Moroever, clause 10 for A' is insured from the fact that clauses 11 and 6 hold for A.
For rule [COMPL-REG] we only have to check clauses 7 and 9. In A' the ambient h is not in $\texttt{wait}$ because 10 holds for A', so clause 7 holds for A'. Clause 9 is derived from the fact that clause 10 holds for A. $\square$

Administrative reductions are strongly normalizing.

Lemma 5.6 *There is a function μ from well-formed terms to natural numbers such that for A, A', if $A \longmapsto_{\text{adm}} A'$, then $\mu(A) > \mu(A')$.*

Proof: To define $\mu(A)$ we define also $\mu(P)$ for a process P.
By structural induction on A and P.

- $\mu(\mathbf{0}) = \mu(h \rhd k) = 0$,
- $\mu(h{:}\,n[\,P\,]_k) = \mu(P)$,
- $\mu(h\{\mathit{MsgBody}\}) = d + 1$ if $\mathit{MsgBody} \neq \texttt{register}\,P$ where d is the distance of h from $\texttt{root}$,
- $\mu(h\{\texttt{register}\,P\}) = d + 1 + \mu(P)$ where d is the distance of h from $\texttt{root}$,
- $\mu(A_1 \parallel A_2) = \mu(A_1) + \mu(A_2)$,
- $\mu((\boldsymbol{\nu} p\,)A) = \mu(A)$,
- $\mu(P_1 \parallel P_2) = \mu(P_1) + \mu(P_2)$,
- $\mu((\boldsymbol{\nu} n\,)P) = \mu(P) + 1$,
- $\mu(\texttt{cap}\,v.\,P) = \mu(P) + \omega$, where $v = x$ or $v = n$, and $\texttt{cap} \in \{\overline{\texttt{in}}, \overline{\texttt{open}}, \texttt{in}, \texttt{out}\}$
- $\mu(M[\,P\,]) = \mu(\texttt{rec}\,X.\,P) = \mu((x)\;P) = \mu(\texttt{wait}.\,P) = \mu(P) + 1$,

- $\mu(\langle M \rangle) = \mu(X) = \mu(\{\mathit{Request}\}) = 1$.

First note that for all A, $\mu(A) \geq 0$. We have to show that $A \longmapsto_{\mathrm{adm}} A'$ implies $\mu(A) > \mu(A')$. Let us just consider the [REQ-…] rules. In this case $A \longmapsto_{\mathrm{adm}} A'$ iff $A \equiv \boldsymbol{\nu}\overline{h}\, A_1 \parallel A_2$, $A' \equiv \boldsymbol{\nu}\overline{h}\, A'_1 \parallel A_2$, and

- $A_1 = h{:}\, n[\, \mathtt{cap}\, m.\, Q \mid Q' \,]_k$,
- $A'_1 = h{:}\, n[\, \mathtt{wait}.\, Q \mid Q' \,]_k \parallel k\{\mathtt{cap}\, m, h\}$.

Therefore $\mu(A) = \mu(A_2) + \mu(Q') + \mu(Q) + \omega$, and $\mu(A') = \mu(A_2) + \mu(Q') + \mu(Q) + 1 + d + 1$ where d is the distance of k from $\mathtt{root}$. So $\mu(A) > \mu(A')$. The other rules are similar. □

We can now conclude that administrative reductions are Church-Rosser.

Theorem 5.7 *Suppose A is well-formed. If $A \Longmapsto_{\mathrm{adm}} A_1$ and $A \Longmapsto_{\mathrm{adm}} A_2$ then either $A_1 \equiv A_2$, or there is A_3 such that $A_1 \Longmapsto_{\mathrm{adm}} A_3$ and $A_2 \Longmapsto_{\mathrm{adm}} A_3$.*

Proof: Standard consequence of Lemma 5.4, using term-rewriting techniques. (Note that since we have just weak confluence, we need also termination). □

Definition 5.8 *A net A is in* normal form *if it cannot perform an administrative reduction.*

Corollary 5.9 *If A is well-formed, then there is a normal form $A^\star$ such that $A \Longmapsto_{\mathrm{adm}} A^\star$. Moreover, the normal form $A^\star$ is unique up to $\equiv$, in the sense that if there is another process A' which satisfies the above clause, then $A' \equiv A^\star$.*

Proof: From Theorem 5.7 and Lemma 5.6. □

Normal forms are characterized by the following well-formed nets.

Lemma 5.10 *Let A be a well-formed net in normal form. Then $A \equiv \boldsymbol{\nu}\overline{n}\ A_1 \parallel A_2 \parallel \cdots \parallel A_p$ where for all i, $1 \leq i \leq p$,*

- *A_i is an agent (either a forwarder or a located ambient)*
- *if A_i is a located ambient, then $A_i = h{:}\, n[\, P_1 \mid \cdots \mid P_q \,]_k$, where all j, $1 \leq j \leq q$,*
  - *$P_j = \langle M \rangle$, or $P_j = (x)P$, or*
  - *$P_j = X$, or $\mathtt{rec}\, X.\, P$, or*
  - *$\mathtt{wait}.\, P$, or $\{\mathit{Request}\}$.*

Definition 5.11 *Let $[\![.]\!]$ be the translation of terms of SA into terms of PAN, so defined:*

$$[\![P]\!] \stackrel{\mathit{def}}{=} \mathtt{root}{:}\, \mathtt{rootname}[\, P \,]_{\mathtt{rootparent}}$$

We write $[\![P]\!]^\star$ for the normal form of $[\![P]\!]$.

Lemma 5.12 *Suppose that P is well-typed. Then $[\![P]\!]$ is well-formed.*

Proof: All the clauses are trivially verified, since there is only one located ambient, and there are neither requests nor $\mathtt{wait}$ ambients. We use Proposition 2.1 to insure clause 9 of the definition of well-formedness. □

Given a well-formed net A we define an SA process P that is encoded by A. (A process may be encoded by more than one net.) The mapping is defined in two steps, first we remove from nets of PAN all messages (both agents and processes) and `wait` prefixes, obtaining a net that is the compositions of agents, such that inside located ambients there are SA processes. Then we remove locations and forwarders obtaining an SA term.

Definition 5.13 *Define $\mathcal{W}(.)$ from well-formed nets of PAN to well-formed nets of PAN as follows. Given A if A does not contain* `wait` *prefixes then $\mathcal{W}(A) = A$. Otherwise if $A \equiv \boldsymbol{\nu}\overline{h}\, A_1 \parallel A_2$, with $A_1 = h\!:n[\, P_1 \mid \mathtt{wait}.\, P_2\,]_k$, and k' is the ambient parent of h in A:*

1. *if $A_2 \equiv k''\{M, h\} \parallel A_3$ where k' is the ambient target of the message $k''\{M, h\}$, then $\mathcal{W}(A) = \boldsymbol{\nu}\overline{h}\, h\!:n[\, P_1 \mid M.\, P_2\,]_k \parallel \mathcal{W}(A_3)$;*
2. *if $A_2 \equiv k'\!:m[\,\{M, h\} \mid Q\,]_{h'} \parallel A_3$, then $\mathcal{W}(A) = \boldsymbol{\nu}\overline{h}\, h\!:n[\, P_1 \mid M.\, P_2\,]_k \parallel k'\!:m[\, Q\,]_{h'} \parallel \mathcal{W}(A_3)$;*
3. *if $A_2 \equiv h\{\mathtt{go}\ h'\} \parallel A_3$, then $\mathcal{W}(A) = \boldsymbol{\nu}\overline{h}\, h\!:n[\, P_1 \mid P_2\,]_{h'} \parallel \mathcal{W}(A_3)$;*
4. *if $A_2 \equiv h\{\mathtt{OK}\overline{\mathtt{in}}\} \parallel A_3$, then $\mathcal{W}(A) = \boldsymbol{\nu}\overline{h}\, h\!:n[\, P_1 \mid P_2\,]_k \parallel \mathcal{W}(A_3)$;*
5. *if $A_2 \equiv h\{\mathtt{migrate}\} \parallel A_3$, then since A is well-formed $A_3 \equiv k'\!:m[\,\mathtt{wait}.\, Q_1 \mid Q_2\,]_{h'} \parallel A_3'$ then $\mathcal{W}(A) = \boldsymbol{\nu}\overline{h}\, h \triangleright k \parallel k'\!:m[\, P_1 \mid P_2 \mid Q_1 \mid Q_2\,]_{h'} \parallel \mathcal{W}(A_3')$;*
6. *if $A_2 \equiv h\{\mathtt{register}\ R\} \parallel A_3$, then $\mathcal{W}(A) = \boldsymbol{\nu}\overline{h}\, h\!:n[\, P_1 \mid P_2 \mid R\,]_k \parallel \mathcal{W}(A_3)$.*

Since clauses 7, 6, and 12 of Definition 5.1 hold for A, $\mathcal{W}(A)$ does not contain any `wait` prefix, message agents, or request processes. Note that, if A is in normal form, then only possible clause applicable is 2.

It is not difficult to prove the following lemma.

Lemma 5.14 *If A is well-formed then $\mathcal{W}(A)$ is well-formed.*

Proof: For all the clauses except the first two the result derives from Lemma 5.5. For the first two clauses note that since clause 9 holds for A then P_1 cannot start with a prefix, so clause 9 holds also for the resulting agent. $\square$

Next, since $\mathcal{W}(A)$ is well formed the dependency relation between locations forms a tree, we can remove locations and forwarders reconstructing the nested structure of SA terms. Given a location k in a well-formed net $A \equiv \boldsymbol{\nu}\overline{h}\, A'$ we define $A'\!\restriction_k \stackrel{\text{def}}{=} A''$ where A'' is the composition of the located agents or forwarders whose location h is such that $h \leq k$ (hence $A\!\restriction_k = \mathbf{0}$ if no such agent exists).

Definition 5.15 *We define $\mathcal{L}(.)$ from well-formed terms of PAN that do not contain message agents to SA terms, as follows. Given $A \equiv \boldsymbol{\nu}\overline{h}\, h\!:n[\, P\,]_k \parallel A' \parallel_{1 \leq i \leq p} A_i$ where k is the root of the dependency relation, and $\{A_i \mid 1 \leq i \leq p\}$ is the set of the ambients that are children of the agent h,*

$$\mathcal{L}(A) \stackrel{\mathit{def}}{=} n[\, P \mid_{1 \leq i \leq p} \mathcal{L}((A_i \parallel A')\!\restriction_{h_i})\,]$$

where h_i is the home location of A_i.

Note that, since the dependency relation forms a tree, in the previous definition for $i \neq j$, $(A_i \parallel A')\!\restriction_{h_i}$ and $(A_j \parallel A')\!\restriction_{h_j}$ do not share any agent.

Definition 5.16 *Define $\{\!\!\{.\}\!\!\}$ from well-formed terms of PAN to SA terms as follows:*

$$\{\!\!\{A\}\!\!\} = \mathcal{L}(\mathcal{W}(A)).$$

Let us now consider forwarders. Note that the previous definitions are given in terms of the target and parent relations, so they depend on forwarders between agents.

Definition 5.17 *We write $A >_{\mathrm{fw}} A'$ if $A \equiv \boldsymbol{\nu} h\,(h \rhd k \mid B)$, for some h, k and $A' \equiv B\{k/h\}$. And we write $>^{\star}_{\mathrm{fw}}$ for the reflexive and transitive closure of $>_{\mathrm{fw}}$.*

Substitution of forwarders preserves the target and parent relation between agents.

Lemma 5.18 *If A is well-formed, and $A >_{\mathrm{fw}} A'$, then A' is well-formed, and*

- *if k is the ambient target of a message $h\{MsgBody\}$ in A, then k is also the ambient target of the corresponding message in A' (in A', h may be substituted with some other location);*
- *if k is the ambient parent of $h{:}\,n[\,P\,]_{h'}$ in A then k is the ambient parent of the corresponding agent in A' (in this case, in A', h' may be substituted with some other location).*

Administrative reductions and forwarders substitutions do not change the process that is encoded by A.

Lemma 5.19 *Let A be a well-formed net*

1. *$A \longmapsto_{\mathrm{adm}} A'$ then $\{\!\!\{A\}\!\!\} = \{\!\!\{A'\}\!\!\}$,*
2. *$A >^{\star}_{\mathrm{fw}} A'$ then $\{\!\!\{A\}\!\!\} = \{\!\!\{A'\}\!\!\}$.*

Proof: 1. If $A \longmapsto_{\mathrm{adm}} A'$, let $A \equiv \boldsymbol{\nu}\overline{h}\, A_1 \parallel A_2$, where $A' \equiv \boldsymbol{\nu}\overline{h}\, A_1' \parallel A_2$.

If $A_1 = h{:}\,n[\,m[\,P\,] \mid Q\,]_{h'}$ is the antecedent of the rule [NEW-LOCAMB], let $A_1' = h{:}\,n[\,Q\,]_{h'} \parallel k{:}\,m[\,P\,]_h$, and $A' \equiv \boldsymbol{\nu}\overline{h}\,\boldsymbol{\nu} k\, A_1' \parallel A_2$. Since A is well formed neither P nor Q have a `wait` prefix. So $\mathcal{W}(A_1) = A_1$, and $\mathcal{L}(A_1 \parallel A_2)$ contains $n[\,m[\,P\,] \mid Q \mid Q_1 \mid \cdots \mid Q_p\,]$ where Q_i are the translations of the nets $A_2\lceil_{h_i}$, where h_i are the children of h in A_2. For A_1' we have that $\mathcal{W}(A_1') = A_1'$, and so also $\mathcal{L}(A_1' \parallel A_2)$ contains $n[\,m[\,P\,] \mid Q \mid Q_1 \mid \cdots \mid Q_p\,]$ where Q_i is as before.

When A_1 is the antecedent of the rule [FW-MSG], if h' is the location of the target of the message in A_1, then h' is also the target of the message in A_1'. For the consumption rules we use the clauses of the definition of $\mathcal{W}(.)$ to show that $\mathcal{W}(A_1 \parallel A_2) = \mathcal{W}(A_1' \parallel A_2)$.

If $A_1 = h{:}\,n[\,\mathtt{cap}\; m.Q \mid Q'\,]_k$, where $\mathtt{cap} \in \{\overline{\mathtt{in}}, \overline{\mathtt{open}}, \mathtt{in}, \mathtt{out}\}$, Q' does not have unguarded ambients, and $A_1' = h{:}\,n[\,\mathtt{wait}.Q \mid Q'\,]_k \parallel k\{\mathtt{cap}\; m, h\}$, use the first two clauses of the definition of $\mathcal{W}(.)$ to show that $\mathcal{W}(A_1 \parallel A_2) = \mathcal{W}(A_1' \parallel A_2)$.

2. From Lemma 5.18 since the translation $\{\!\!\{.\}\!\!\}$ depends only on the ambient target and parent relations we derive that if $A >_{\mathrm{fw}} A'$, then $\{\!\!\{A\}\!\!\} = \{\!\!\{A'\}\!\!\}$. The result can be obtained by induction on the number of substitution such that $A >^{\star}_{\mathrm{fw}} A'$.

$\square$

Lemma 5.20 *Let P be a well typed SA process.*

1. *If $[\![P]\!] \Longmapsto_{\mathrm{adm}} A$ then $\{\!\!\{A\}\!\!\} = \mathtt{root}[\,P\,]$.*
2. *If $\mathtt{root}[\,P\,] \equiv \{\!\!\{A\}\!\!\}$ for some well formed A, and A contains neither forwarders nor completions, then $[\![P]\!] \Longmapsto_{\mathrm{adm}} A$.*

Proof: 1. From the definition of $[\![P]\!]$ is is easy to see that $\{\!\{[\![P]\!]\}\!\} = \mathtt{root}[\,P\,]$. From Lemma 5.19.1 we derive the result.

2. By case analysis on the definition of $\{\!\{.\}\!\}$. First, since there are no forwarders the dependency relation coincides with the ambient parent/child relation. So the mapping $\mathcal{L}(.)$ is trivial. Moreover, since A does not contain completions the only clauses of $\mathcal{W}()$ that may be applied to A are the first two.

$\square$

Definition 5.21 *Let $\rightsquigarrow$ to be such that $A \rightsquigarrow A'$ if there are A_1, A_2 where A_1 is in normal form, such that $A \Longmapsto_{\mathrm{adm}} A_1$, $A_1 \longmapsto_{\mathrm{pr}} A_2$, and $A_2 \Longmapsto_{\mathrm{adm}} A'$. (That is, the sequence of all possible administrative reductions followed by a proper reduction, followed by any number of administrative reductions.)*

Proper reductions of nets encoding well-typed processes produce well-formed nets.

Lemma 5.22 *Suppose that P is well-typed and that $[\![P]\!] \rightsquigarrow A''$. Then A'' is well-formed.*

Proof: Let $[\![P]\!]^\star = A$. From Lemmas 5.12, and 5.5 we derive that A is well formed. Let $A \longmapsto_{\mathrm{pr}} A'$, we will show that A' is well formed. Therefore, since $A' \Longmapsto_{\mathrm{adm}} A''$, we can conclude, again from Lemma 5.5 that A'' is well formed.
Let $A \equiv \boldsymbol{\nu}\overline{h}\, A_1 \parallel A_2$, $A' \equiv \boldsymbol{\nu}\overline{h}\, A_1' \parallel A_2$, and $A_1 \longmapsto_{\mathrm{pr}} A_1'$. Consider the four possible derivations corresponding to the application of the [LOCAL-...] rules.
[LOCAL-COM]. Let $A_1 = h{:}n[\,\langle M\rangle \mid (x).\,Q \mid Q'\,]_k$, and $A_1' = h{:}n[\,Q\{^M\!/\!_x\} \mid Q'\,]_k$. All the clauses of Definition 5.1 except 8 and 9 are easily verified for A'. Let us consider 8. Even though Q' may not contain an unguarded ambient, it could be $Q' = \mathtt{wait}.\,Q_1 \mid Q_2$, and Q containing an unguarded ambient. However, since A is a normal form (and is well formed) we know that there are no completions with target h. So clause 8 holds for A'. To verify 9 note that, since A is in normal form Q' is neither an unguarded ambient, nor has a subcomponent with a prefix different from `wait`. If Q' does not have a subcomponent with a `wait` prefix, then 9 holds for A'. In case $Q' = \mathtt{wait}.\,Q_1 \mid \cdots$ since clause 7, and 12 of well-formedness hold for A, there is a corresponding request (either agent or process). So from the definition of $\mathcal{W}(\cdot)$ in $\{\!\{A\}\!\}$ the ambient n is $n[\,\langle M\rangle \mid (x).\,Q \mid \mathtt{cap}.\,Q_1 \mid \cdots]$ for some $\mathtt{cap} \in \{\overline{\mathtt{in}}, \overline{\mathtt{open}}, \mathtt{in}, \mathtt{out}\}$. Since P is well-typed, and types are preserved by reduction, from Proposition 2.1 we have that $Q\{^M\!/\!_x\}$ cannot have an unguarded prefix. Therefore clause 9 holds for A'.
[LOCAL-IN]. Let $A_1 = h{:}n[\,\{\mathtt{in}\, m, h'\} \mid \{\overline{\mathtt{in}}\, m, k'\} \mid Q'\,]_k$, and $A_1' = h{:}n[\,Q'\,]_k \parallel h'\{\mathtt{go}\ k'\} \parallel k'\{\mathtt{OK}\overline{\mathtt{in}}\}$. The only nontrivial clauses to verify are clauses 2, 3, and 7. To verify 2, note that, from clause 12 for A, we derive that k is father of h' and k', and so $k' \not\leq h'$ (also $h' \not\leq k'$). That 3b holds for A' derives from the fact that 3a holds for A. Finally, since A is well-formed (clause 7) the ambients located at h' and k' are in `wait`, and $h' \neq k'$. Therefore clause 7 holds also for A' with the completions generated by the reduction.
[LOCAL-OUT]. Let $A_1 = h{:}n[\,\{\mathtt{out}\, n, h'\} \mid \overline{\mathtt{out}}\ n.\,Q \mid Q'\,]_k$, and $A_1' = h{:}n[\,Q \mid Q'\,]_k \parallel h'\{\mathtt{go}\ k\}$. As before, from the fact that clause 12 holds for A, we derive that clause 2 holds for A', and also since 3a holds for A, then 3b holds for A. Since A is well-formed (clause 7) the ambients located at h' is in `wait`. Therefore clause 7 holds also for A' with the completion generated by the reduction. The clause 8 holds for A' with a proof similar to the one for message exchanging, [LOCAL-COM]. Finally to verify clause 9 for A', note that, since clause 9 holds for A_1, then Q' cannot have an unguarded prefix the clause holds also for A_1'. All the other clauses hold for A' since they hold for A.

[LOCAL-OPEN]. Let $A_1 = h{:}n[\,\{\overline{\texttt{open}}\,m, h'\} \mid \texttt{open}\ m.\,Q \mid Q'\,]_k$, and $A_1' = h{:}n[\,\texttt{wait}.\,Q \mid Q'\,]_k \parallel h'\{\texttt{migrate}\}$. We verify only clauses 7 and 11; all the others are either trivial or have a proof similar to one of the previous cases. Since A is well-formed, $A_2 = h'{:}m[\,\texttt{wait}.\,P \mid P'\,]_{k'} \parallel A_2'$ and h is the father of h'. Consider clause 7. The ambient in wait state located at h' was in A the source of a request (that does not occur in A') and is in A' is the target of a migrate completion. The new ambient in wait state (located at h) is the ambient parent of the target of a migrate message, so clause 7 holds for A'. Finally we want to verify clause 11 for A'. Note that, from the well-formedness of A we have that Q' cannot contain a subcomponent with a $\texttt{wait}$ prefix. From Lemma 5.20, in P we have that the ambient n is $n[\,m[\,\overline{\texttt{open}}\ m.\,R \mid R' \mid \cdots\,] \mid \texttt{open}\ m.\,Q \mid Q' \mid \cdots\,]$, so $P \longrightarrow P'$ where P' contains $[\,Q \mid Q' \mid R \mid R' \mid \cdots\,]$ and P' (from subject reduction) is well-typed (with m is single threaded). Therefore, from Proposition 2.1, clause 11 holds for A'. $\square$

Theorem 5.23 *1. If P is a well-typed SA process, and $P \longrightarrow P'$, then $[\![P]\!] \rightsquigarrow\ A >^{\star}_{\mathrm{fw}} [\![P']\!]^{\star}$.*

2. If A is well-formed, $\{\!\{A\}\!\}$ is well-typed, and $A \rightsquigarrow A'$, then $\{\!\{A\}\!\} \longrightarrow \{\!\{A'\}\!\}$.

Proof: 1. If $P \longrightarrow P'$ there are m_1,...,m_s, Q_0, ...,Q_s, Q, and Q' such that

- $P \equiv \boldsymbol{\nu}\overline{n}\ m_1[\,m_2[\,\cdots m_s[\,Q \mid Q_s\,]\cdots\,] \mid Q_1\,] \mid Q_0$,
- $P' \equiv \boldsymbol{\nu}\overline{n}\ m_1[\,m_2[\,\cdots m_s[\,Q' \mid Q_s\,]\cdots\,] \mid Q_1\,] \mid Q_0$,

Q is the left-hand-side of one of the rules (R-Com), (R-In), (R-Out), or (R-Open), and Q' is the right-hand-side of the corresponding rule.

Let $A' = [\![P]\!]^{\star}$, from Lemma 5.20.1 $\{\!\{A'\}\!\} = \texttt{root}[\,P\,]$, and since A' is in normal form, and Lemma 5.10,

$$A' \equiv \boldsymbol{\nu}\overline{n}\ h_1{:}m_1[\,Q_1'\,]_{\texttt{root}} \parallel h_2{:}m_2[\,Q_2'\,]_{h_1} \parallel \cdots \parallel h_s{:}m_s[\,Q_s' \mid Q''\,]_{h_{s-1}} \parallel Q_0'$$

where there are neither forwarders nor messages in Q_0'.

If the reduction axiom applied is [R-MSG], then $Q = \langle M\rangle \mid (x)P$ and $Q' = P\{^{M}\!/\!_{x}\}$. Since no administrative reduction of PAN can be applied to Q, we also have that in A, $Q'' = Q$. The local reduction [LOCAL-COM] can be applied to Q'' producing Q' and emitting no message. So $A' \longmapsto_{\mathrm{pr}} A''$ where

$$A'' \equiv \boldsymbol{\nu}\overline{n}\ h_1{:}m_1[\,Q_1'\,]_{\texttt{root}} \parallel h_2{:}m_2[\,Q_2'\,]_{k_1} \parallel \cdots \parallel h_s{:}m_s[\,Q_s' \mid Q'\,]_{k_{s-1}} \parallel Q_0'$$

From Lemma 5.22 we know that A'' is well formed, so $\{\!\{A''\}\!\}$ is defined. Moreover, from $\{\!\{A'\}\!\} = \texttt{root}[\,P\,]$, we derive that $\{\!\{A''\}\!\} = \texttt{root}[\,P'\,]$, and A'' does not contain neither forwarders (since A' does not contain forwarders) nor completions (since no completion is generated by the reduction [LOCAL-COM]). So from Lemma 5.20.2 we have that $[\![P']\!] \Longmapsto_{\mathrm{adm}} A''$. Let $A = (A'')^{\star}$. Since A is in normal form, and from Corollary 5.9 normal forms are unique, $A \equiv [\![P']\!]^{\star}$. This proves that $[\![P]\!] \rightsquigarrow [\![P']\!]^{\star}$.

If the reduction axiom applied is [R-OPEN], then $Q = \texttt{open}\,n.\,R \mid n[\,\overline{\texttt{open}}\,n.\,R_1 \mid R_2\,]$ and $Q' = R \mid R_1 \mid R_2$. So

- $P \equiv \boldsymbol{\nu}\overline{n}\ m_1[\,m_2[\,\cdots m_s[\,\texttt{open}\,n.\,R \mid n[\,\overline{\texttt{open}}\,n.\,R_1 \mid R_2\,] \mid Q_s\,]\cdots\,] \mid Q_1\,] \mid Q_0$,
- $P' \equiv \boldsymbol{\nu}\overline{n}\ m_1[\,m_2[\,\cdots m_s[\,R \mid R_1 \mid R_2 \mid Q_s\,]\cdots\,] \mid Q_1\,] \mid Q_0$,

Since A' is in normal form,

$$\begin{array}{rcl} A' & \equiv & \boldsymbol{\nu}\overline{n}\ h_1{:}\,m_1[\,Q'_1\,]_{\mathtt{root}} \parallel h_2{:}\,m_2[\,Q'_2\,]_{h_1} \parallel \cdots \parallel \\ & & h_s{:}\,m_s[\,Q'_s \mid \mathtt{open}\,n.\,R \mid \{\overline{\mathtt{open}}\ n, h\}\,]_{h_{s-1}} \parallel h{:}\,n[\,\mathtt{wait}.\,R_1 \mid R'_2\,]_{h_s} \parallel Q''_0 \end{array}$$

where $Q'_0 = h{:}\,n[\,\mathtt{wait}.\,R_1 \mid R'_2\,]_{h_s} \parallel Q''_0$. So $A' \longmapsto_{\mathrm{pr}} A''$ where

$$\begin{array}{rcl} A'' & = & \boldsymbol{\nu}\overline{n}\ h_1{:}\,m_1[\,Q'_1\,]_{\mathtt{root}} \parallel h_2{:}\,m_2[\,Q'_2\,]_{h_1} \parallel \cdots \parallel \\ & & h_s{:}\,m_s[\,Q'_s \mid \mathtt{wait}.\,R]_{h_{s-1}} \parallel h\{\mathtt{migrate}\} \parallel h{:}\,n[\,\mathtt{wait}.\,R_1 \mid R'_2\,]_{h_s} \parallel Q''_0 \end{array}$$

Let A''' be

$$\boldsymbol{\nu}\overline{n}\ h_1{:}\,m_1[\,Q'_1\,]_{\mathtt{root}} \parallel h_2{:}\,m_2[\,Q'_2\,]_{h_1} \parallel \cdots \parallel h \rhd h_s \parallel h_s{:}\,m_s[\,Q'_s \mid R \mid R_1 \mid R'_2\,]_{h_{s-1}} \parallel Q''_0$$

From Lemma 5.22 we know that A'' is well formed, so also A''' is well formed. From definition of $\mathcal{W}(.)$, $\mathcal{W}(A''') = \mathcal{W}(A'')$ and A''' does not contain completions. Moreover, from $\{\!\!\{A'\}\!\!\} = \mathtt{root}[\,P\,]$, we derive that $\{\!\!\{A'''\}\!\!\} = \mathtt{root}[\,P'\,]$. Let $A = (A''')^\star$, the only forwarder contained in A is $h \rhd h_s$. Let $B = A\{{}^{h}\!/\!{}_{h_s}\}$, B does not contain forwarders (and completions), it is well formed, and from Lemma 5.19, $\{\!\!\{B\}\!\!\} = \mathtt{root}[\,P\,]$. So from Lemma 5.20.2 we have that $[\![P']\!] \Longmapsto_{\mathrm{adm}} B$. Since A is in normal form, also B is in normal form, and from Corollary 5.9 normal forms are unique. Therefore, $B \equiv [\![P']\!]^\star$. So $[\![P]\!] \rightsquigarrow\ A >_{\mathrm{fw}} B \equiv [\![P']\!]^\star$.

If the reduction axioms applied are [R-IN] or [R-OUT] the proof is similar to the previous one (except that non forwarders are generated), so $[\![P]\!] \rightsquigarrow [\![P']\!]^\star$.

2. Let $A \rightsquigarrow A'$, there are B, and B' such that B is in normal form, $A \Longmapsto_{\mathrm{adm}} B$, $B \longmapsto_{\mathrm{pr}} B'$, and $B' \Longmapsto_{\mathrm{adm}} A'$. Let $B \equiv \boldsymbol{\nu}\overline{h}\,A_1 \parallel A_2$, $B' \equiv \boldsymbol{\nu}\overline{h}\,A'_1 \parallel A_2$, and $A_1 \longmapsto_{\mathrm{pr}} A'_1$. Consider the four possible derivations corresponding to the application of the [LOCAL-...] rules.

   <u>[LOCAL-COM].</u> Let $A_1 = h{:}\,n[\,\langle M\rangle \mid (x).\,Q \mid Q'\,]_k$, and $A'_1 = h{:}\,n[\,Q\{{}^{M}\!/\!{}_{x}\} \mid Q'\,]_k$. With a proof similar to the corresponding one of Lemma 5.22 we can show that B' is well formed (using the fact that $\{\!\!\{B\}\!\!\} = \{\!\!\{A\}\!\!\}$ is well typed). From the definition of $\{\!\!\{\cdot\}\!\!\}$,

$$\{\!\!\{B\}\!\!\} \equiv \boldsymbol{\nu}\overline{n}\ m_1[\,m_2[\cdots m_s[\,n[\,\langle M\rangle \mid (x).\,Q \mid Q''\,] \mid Q_s\,]\cdots] \mid Q_1\,] \mid Q_0$$

   Applying rule [R-MSG], $\{\!\!\{B\}\!\!\} \longrightarrow P'$ such that

$$P' = \boldsymbol{\nu}\overline{n}\ m_1[\,m_2[\cdots m_s[\,n[\,Q\{{}^{M}\!/\!{}_{x}\} \mid Q''\,] \mid Q_s\,]\cdots] \mid Q_1\,] \mid Q_0$$

   From the definition of $\{\!\!\{\cdot\}\!\!\}$ we can see that $\{\!\!\{B'\}\!\!\} = P'$. This concludes the proof.

   <u>[LOCAL-OPEN].</u> Let $A_1 = h{:}\,n[\,\{\overline{\mathtt{open}}\,m, h'\} \mid \mathtt{open}\ m.\,Q \mid Q'\,]_k$, and $A'_1 = h{:}\,n[\,\mathtt{wait}.\,Q \mid Q'\,]_k \parallel h'\{\mathtt{migrate}\}$. From the definition of $\{\!\!\{\cdot\}\!\!\}$,

$$\{\!\!\{B\}\!\!\} \equiv \boldsymbol{\nu}\overline{n}\ m_1[\,m_2[\cdots m_s[\,n[\,\mathtt{open}\,m.\,R \mid m[\,\overline{\mathtt{open}}\,m.\,R_1 \mid R_2\,] \mid Q''\,] \mid Q_s\,]\cdots] \mid Q_1\,] \mid Q_0$$

   With a proof similar to the corresponding one of Lemma 5.22 we can show that B' is well formed (using the fact that $\{\!\!\{B\}\!\!\} = \{\!\!\{A\}\!\!\}$ is well typed). Applying rule [R-OPEN], $\{\!\!\{B\}\!\!\} \longrightarrow P'$ such that

$$P' = \boldsymbol{\nu}\overline{n}\ m_1[\,m_2[\cdots m_s[\,n[\,R \mid R_1 \mid R_2 \mid Q''\,] \mid Q_s\,]\cdots] \mid Q_1\,] \mid Q_0$$

   From the definition of $\{\!\!\{\cdot\}\!\!\}$ we can see that $\{\!\!\{B'\}\!\!\} = P'$.

   The cases [LOCAL-IN] and [LOCAL-OUT] are similar.

□

Definition 5.24 (observability) *We write* $A \downarrow_n$ *if* $A \equiv \boldsymbol{\nu}\widetilde{p}\,(\texttt{root:rootname}[\,\{M,h\}\mid P\,]_{\texttt{rootparent}} \parallel A')$ *where* $M \in \{\overline{\texttt{in}}\,n, \overline{\texttt{open}}\,n\}$ *and* $n \notin \widetilde{p}$. *We write* $A \Downarrow_n$ *if* $A \Longmapsto \downarrow_n$.

Observability in SA is defined similarly as for PAN: $P \Downarrow_n$ *if* $P \Longrightarrow P'$, *for some* P' *such that* $P' \equiv \boldsymbol{\nu}\widetilde{n}\,(n[\,M.Q_1 \mid Q_2\,] \mid Q_3)$ *where* $M \in \{\overline{\texttt{in}}\,n, \overline{\texttt{open}}\,n\}$ *and* $n \notin \widetilde{n}$.

Corollary 5.25 (adequacy) *Let* $P \in SA$. *It holds that* $P \Downarrow_n$ *iff* $[\![P]\!] \Downarrow_n$, *for all* n.

Proof: Follows from Theorem 5.23. □

6 Correctness of the abstract machine for immobile ambients

In this section we extend the proof of correctness to all well-typed SA terms.

The other type of ambients in SA are the *immobile* ambients. A well-typed SA program may contain both single-threaded *and* immobile ambients. These are ambients that:

1. cannot jump into or out of other ambients;

2. cannot be opened.

Thus the only capabilities that an immobile ambient can exercise are $\overline{\texttt{in}}$, $\overline{\texttt{out}}$, and $\texttt{open}$. The main property that immobile ambients have is expressed from the following statement.

Proposition 6.1 *If* P *is well-typed SA term and the subterm* $n[\,Q\,]$ *is an immobile ambient, then no unboxed subcomponent of* Q *has the unguarded capabilities* $\texttt{in}$, $\texttt{out}$, $\overline{\texttt{open}}$.

Since immobile ambients can exercise several capabilities at the same time, in the abstract machine, an immobile ambient can contain several `wait` prefixes. To be able to distinguish among them, and make sure that a completion message wakes up the right process, we add an index i (where i is a natural number) to `wait` prefixes and messages. Thus in the syntax for nets we add the production

$$h\{\mathit{MsgBody}\}_i$$

and in the process-related syntax we add the productions

$$\texttt{wait}_i.\,P \;\Big|\; \{\mathit{Request}\}_i$$

The additions to the reduction rules are simple. We report them below. We assume that the set of names is partitioned into two (infinite) subsets, one for the names of the single-threaded ambients, the other for the names of the immobile ambients. In rule IMM-PAR-PROC, we write $\texttt{wait}(R)$, where R is a process, for the set

$$\{i \;:\; \texttt{wait}_i \text{ appears in } R\ \}$$

The side condition in the rules ensures that all `wait` prefixes inside an ambient have different indices. Consider the rules for immobile ambients. Firstly, for the local reductions we have that, when matching a $\overline{\texttt{in}}$ request from an immobile ambient with an `in` request from a ST ambient, rule [IMM-LOCAL-IN],we generate an indexed completion for the immobile ambient (which corresponds to an indexed `wait`) and a regular completion for the moving ST ambient. The rule [IMM-LOCAL-OPEN] reflects the fact that, in a well-types SA term, a single threaded ambient may be opened by an immobile ambient. (However, it must be the case that the local processes of the immobile ambient do not have any unguarded prefix.) Hence, if the ambient sending the `open` request is immobile we generate an indexed request. When combining the rules for immobile and ST ambients,

Local reductions

$$\{\mathtt{in}\, n, h\} \mid \{\overline{\mathtt{in}}\, n, k\}_i \xrightarrow[-:-]{-} \mathbf{0} \gg h\{\mathtt{go}\, k\} \mid k\{\mathtt{OK}\overline{\mathtt{in}}\}_i \qquad [\textsc{Imm-local-in}]$$

$$\mathtt{open}\, n.\, P \mid \{\overline{\mathtt{open}}\, n, h\} \xrightarrow[-:m]{-} \mathtt{wait}_i.\, P \gg h\{\mathtt{migrate}\}_i \quad \text{where } i \text{ is new and } m \text{ immobile}$$
$$[\textsc{Imm-local-open}]$$

Forwarder

$$h \rhd k \parallel h\{\mathit{MsgBody}\}_i \longmapsto h \rhd k \parallel k\{\mathit{MsgBody}\}_i \qquad [\textsc{fw-msg-Imm}]$$

Consumption of request messages

$$h{:}\, n[\, P\,]_{h'} \parallel h\{\mathit{Request}\}_i \longmapsto h{:}\, n[\, P \mid \{\mathit{Request}\}_i\,]_{h'} \qquad [\textsc{Imm-consume-req}]$$

Emission of request messages (should be $h \neq \mathtt{root}$, and n immobile)

$$\overline{\mathtt{in}}\, n.\, P \xrightarrow[h:n]{k} \mathtt{wait}_i.\, P \gg k\{\overline{\mathtt{in}}\, n, h\}_i \quad \text{where } i \text{ is new} \qquad [\textsc{Imm-req-coin}]$$

Consumption of completion messages

$$h{:}\, n[\, P \mid \mathtt{wait}_i.\, Q\,]_k \parallel h\{\mathtt{OK}\overline{\mathtt{in}}\}_i \longmapsto h{:}\, n[\, P \mid Q\,]_k \qquad [\textsc{Imm-compl-coin}]$$

$$h{:}\, n[\, P \mid \mathtt{wait}.\, Q\,]_k \parallel h\{\mathtt{migrate}\}_i \longmapsto h \rhd k \parallel k\{\mathtt{register}\, P \mid Q\}_i \qquad [\textsc{Imm-compl-migr}]$$

$$h{:}\, n[\, P \mid \mathtt{wait}_i.\, Q\,]_k \parallel h\{\mathtt{register}\, R\}_i \longmapsto h{:}\, n[\, P \mid Q \mid R\,]_k \qquad [\textsc{Imm-compl-reg}]$$

Inference rules

$$\frac{P \xrightarrow[h:n]{k} P' \gg \widetilde{Msg} \qquad \begin{array}{l} Q \text{ does not have unguarded ambients} \\ n \text{ immobile} \\ \mathtt{wait}(P) \cap \mathtt{wait}(Q) = \emptyset \end{array}}{P \mid Q \xrightarrow[h:n]{k} P' \mid Q \gg \widetilde{Msg}} \qquad [\textsc{Imm-par-proc}]$$

to avoid ambiguity, we need to add to the rule [LOCAL-OPEN] of Figure 5 a side condition, and the rule becomes

$$\texttt{open}\, n.\, P \mid \{\overline{\texttt{open}}\, n, h\} \xrightarrow[-:m]{-} \texttt{wait}.\, P \gg h\{\texttt{migrate}\} \quad \text{where } m \text{ is single threaded}$$

Note also that, since the local processes of immobile ambients can only have the unguarded prefixes $\overline{\texttt{in}}$, $\overline{\texttt{out}}$, and $\texttt{open}$, the only requests generated by immobile ambients can be $\overline{\texttt{in}}$ requests (rule [IMM-REQ-COIN]). Again, when combining the rules for immobile and ambients, to avoid ambiguity, we need to add to the rule [REQ-COIN] of Figure 5 the side condition that n is single threaded.

With the new rules, the addition of immobile ambients requires few modifications to the correctness proof of Section 5.

The definition of well-formedness is modified as follows.

Definition 6.2 *Clauses 1– 5 unchanged.*

6. *The prefixes* $\texttt{wait}$, *and* $\texttt{wait}_i$ *(for some i) can only appear as the unguarded prefix of the local process of a located ambient.*

7. (a) *A located ambient with home location h is in wait state iff h appears exactly once in at most one of the following ways:*
      i. *as the source of a request (in a message or in a local process);*
      ii. *as the target of a completion message;*
      iii. *as the ambient parent of the target of* $h\{\texttt{migrate}\}$ *or* $h\{\texttt{migrate}\}_i$.
   (b) *A located ambient with home location h is in wait$_i$ state iff h appears exactly once in at most one of the following ways:*
      i. *as the source of a* $\overline{\texttt{in}}$ *request (in a message or in a local process) with index i;*
      ii. *as the target of* $h\{\texttt{register}\, R\}_i$ *or* $h\{\texttt{OK}\overline{\texttt{in}}\}_i$;
      iii. *as the ambient parent of the target of* $h\{\texttt{migrate}\}_i$.

8. *An located ambient with home location h such that h is the target of a completion (indexed or not) does not contain subambients.*

9. (a) *No single threaded ambient (located or not) has an unboxed subterm with more than one unguarded prefix. No local process of a (located) single threaded ambient has an unguarded prefix* $\texttt{wait}_i$ *for some i.*
   (b) *No immobile ambient (located or not) has an unboxed subterm with the unguarded prefixes* $\texttt{in}, \texttt{out}, \overline{\texttt{open}}$ *or* $\texttt{wait}$. *For all i, no local process of a (located) immobile ambient has more than one unguarded prefix* $\texttt{wait}_i$ *(that is, all the indices of* $\texttt{wait}$ *prefixes must be distinct).*

10. (a) *If there is a message* $h\{\texttt{register}\, R\}$ *and its ambient target (up to $\equiv$) is* $h{:}\, n[\, P_1 \mid \texttt{wait}.\, P_2\,]_k$, *then* $P_1 \mid P_2 \mid R$ *has at most one unguarded prefix, and it is not a* $\texttt{wait}$
    (b) *If there is a message* $h\{\texttt{register}\, R\}_i$, *then R does not have the unguarded prefixes* $\texttt{in}, \texttt{out}, \overline{\texttt{open}}$, *and* $\texttt{wait}$.

11. (a) *If there is a message* $h\{\texttt{migrate}\}$ *and the ambient at h (up to $\equiv$) is* $h{:}\, n[\, Q_1 \mid \texttt{wait}.\, Q_2\,]_{h'}$, *and the ambient parent of h (up to $\equiv$) is* $k{:}\, m[\, P_1 \mid \texttt{wait}.\, P_2\,]_{k'}$ *then* $Q_1 \mid Q_2 \mid P_1 \mid P_2$ *has at most one unguarded prefix.*

(b) *If there is a message* $h\{\mathtt{migrate}\}_i$, *and the ambient at* h *(up to* $\equiv$*) is* $h{:}\, n[\, Q_1 \mid \mathtt{wait}.\, Q_2\,]_{h'}$ *then* $Q_1 \mid Q_2$ *does not have the unguarded prefixes* $\mathtt{in}\,, \mathtt{out}\,, \overline{\mathtt{open}}$, *and* $\mathtt{wait}$.

12. (a) *In every request message* $k\{M, h\}$, *or* $k\{M, h\}_i$ *the location* h *is a child of* k, *and*

(b) *if inside an ambient located at* k *there is a request* $\{M, h\}$ *or* $\{M, h\}_i$ *then* h *is a child of* k.

Administrative and proper reductions are characterized by changing the previous clauses, as follows.

- For $A \longmapsto_{\mathrm{adm}} A'$:
  1. A_1 may be also the antecedent of [FW-MSG-IMM], [IMM-CONSUME-REQ], and [IMM-COMPL-...];
  2. as before;
  3. $A_1 = h{:}\, n[\, \mathtt{cap}\, n.\, Q \mid Q'\,]_k$, then Q' does not have unguarded ambients
     (a) if n is single threaded, $\mathtt{cap} \in \{\overline{\mathtt{in}}\,, \overline{\mathtt{open}}\,\}$, then $A_1' = h{:}\, n[\, \mathtt{wait}.\, Q \mid Q'\,]_k \parallel k\{\mathtt{cap} n, h\}$;
     (b) if n is immobile, $\mathtt{cap} = \overline{\mathtt{in}}\,$, then $A_1' = h{:}\, n[\, \mathtt{wait}_i.\, Q \mid Q'\,]_k \parallel k\{\mathtt{cap}\, n, h\}_i$

- For $A \longmapsto_{\mathrm{pr}} A'$:
  1. as before;
  2. (a) if $A_1 = h{:}\, n[\, \{\mathtt{in}\, m, h'\} \mid \{\overline{\mathtt{in}}\, m, k'\} \mid Q'\,]_k$, then $A_1' = h{:}\, n[\, Q'\,]_k \parallel h'\{\mathtt{go}\ k'\} \parallel k'\{\mathtt{OK}\overline{\mathtt{in}}\}$;
     (b) if $A_1 = h{:}\, n[\, \{\mathtt{in}\, m, h'\} \mid \{\overline{\mathtt{in}}\, m, k'\}_i \mid Q'\,]_k$, then $A_1' = h{:}\, n[\, Q'\,]_k \parallel h'\{\mathtt{go}\ k'\}_i \parallel k'\{\mathtt{OK}\overline{\mathtt{in}}\}_i$;
  3. as before (note that n could be single threaded or immobile);
  4. if $A_1 = h{:}\, n[\, \{\overline{\mathtt{open}}\, m, h'\} \mid \mathtt{open}\ m.\, Q \mid Q'\,]_k$,
     (a) if n is single treaded, then $A_1' = h{:}\, n[\, \mathtt{wait}.\, Q \mid Q'\,]_k \parallel h'\{\mathtt{migrate}\}$;
     (b) if n is immobile, then $A_1' = h{:}\, n[\, \mathtt{wait}_i.\, Q \mid Q'\,]_k \parallel h'\{\mathtt{migrate}\}_i$.

     In all the clauses Q' does not have unguarded ambients.

For the new administrative reduction the proof of the Church Rosser property requires minor modification (mostly additions) to the lemmas.

Lemma 6.3 *Suppose* A *is well-formed. If* $A \longmapsto_{\mathrm{adm}} A'$ *and* $A \longmapsto_{\mathrm{adm}} A''$ *then either* $A' \equiv A''$, *or there is* A''' *such that* $A' \Longmapsto_{\mathrm{adm}} A'''$ *and* $A'' \Longmapsto_{\mathrm{adm}} A'''$.

Proof: Let $A \equiv \boldsymbol{\nu} \overline{h}\, A_1 \parallel A_2$.

1. $A_1 = h{:}\, n[\, \mathtt{cap}\, m.\, Q \mid Q'\,]_k$, where $\mathtt{cap} \in \{\mathtt{in}\,, \mathtt{out}\,, \overline{\mathtt{in}}, \overline{\mathtt{open}}\,\}$, and $A \longmapsto_{\mathrm{adm}} A'$ applying a rule [REQ-…].

   The proof of Lemma 5.4 holds for n single threaded. If n is immobile then there may be other capabilities. So $Q' = \mathtt{cap}\, m.\, P \mid P'$, $A'' \equiv A_1'' \parallel A_3$ and A_1'' obtained from A_1 by applying a [REQ-…] or [IMM-COMPL-COIN] or [IMM-COMPL-REG] (if P' contains the reuired completion) involving Q'. It is easy to see that, since Q' is contained in A_1' and $\mathtt{cap}\, m.\, Q$ in A_1'', we can apply the corresponding rule obtaining an A''' such that $A' \longmapsto_{\mathrm{adm}} A'''$ and $A'' \longmapsto_{\mathrm{adm}} A'''$. The conflict with the [CONSUME-REQ] rule can be handled in a similar way.

2. and 3. The proof is correct for also for immobile ambients.

4. $\underline{A_1 = h{:}\, n[\, P \mid \texttt{wait}_i.\, Q\,]_k \parallel h\{\mathit{Completion}\}_i}$, and $A \longmapsto_{\mathrm{adm}} A'$ applying any rule [IMM-COMPL-$\underline{\ldots}$]. If $P = \texttt{wait}_j.\, Q' \mid P'$ and $A_2 = h\{\mathit{Completion}\}_j \parallel A_2'$ where $i \neq j$, then $A \longmapsto_{\mathrm{adm}} A''$ by applying a rule [IMM-COMPL-...] to the $\texttt{wait}_j$. Again an A''' can be obtained from both: A'' by applying the rule [IMM-COMPL-...] to the completion indexed by i, and A' by applying the rule [IMM-COMPL-...] to the completion indexed by j.

   For $\mathit{Completion} = \texttt{migrate}$, when the conflict arises from the application of the rule [CONSUME-REQ] the proof is as for the single threaded case.

□

Lemma 6.4 *If A is well-formed, and $A \longmapsto_{\mathrm{adm}} A'$, then also A' is well-formed.*

Proof: For this proof we have just to consider the new rules introduced for immobile ambients. Let $A \equiv \boldsymbol{\nu}\overline{h}\, A_1 \parallel A_2$, and $A' \equiv \boldsymbol{\nu}\overline{h}\, A_1' \parallel A_2$ by applying [FW-MSG-IMM] or [IMM-CONSUME-REQ] the proof is as for the proof of the corresponding rules from single threaded ambients in Lemma 5.5.
Consider the rule [IMM-REQ-COIN].

- $A_1 = h{:}\, n[\, \overline{\texttt{in}}\ m.\, Q \mid Q'\,]_k$, where Q' does not contain unguarded ambients, and
- $A_1' = h{:}\, n[\, \texttt{wait}_i.\, Q \mid Q'\,]_k \parallel k\{\overline{\texttt{in}}\ m, h\}_i$.

Since h cannot be neither $\texttt{root}$ nor the source of a forwarder, clauses 3 and 5 hold in A'. Since in A clauses 8 and 9 hold (for the ambient at n), and i is new, so no other $\texttt{wait}$ may be indexed by i, then clauses 8 and 9 hold in A'. Clause 7b holds for A' since it holds for A and A_1'. Finally since h is a child of k, clause 12a holds for A'. The other clauses hold since they hold for A.
If the rule applied is [IMM-COMPL-COIN] note only that since 9 holds for A (in particular for the ambient at n), then Q does not contain any prefix $\texttt{in}$, $\texttt{out}$, $\overline{\texttt{open}}$, or $\texttt{wait}$. So 9 holds for A'. The other clauses are immediate.
If the rule applied is [IMM-COMPL-MIG]

- $A_1 = h{:}\, n[\, P \mid \texttt{wait}.\, Q\,]_k \parallel h\{\texttt{migrate}\}_i$, and
- $A_1' = h \rhd k \parallel k\{\texttt{register}\, P \mid Q\}_i$.

Since n is single threaded, the proof is as for the the corresponding case of Lemma 5.5 except that we have to show that clause 10b holds for A' instead of 10a. This derives from 11b.
If the rule applied is [IMM-COMPL-REG] then, since clause 9 holds for the ambient at n, and clause 10b holds in A then clause 9 holds in A'. The other clauses hold since they hold in A. □

Termination of administrative reductions can be proved by adding the following clauses to the definition of the function μ in Lemma 5.6:

- $\mu(h\{\mathit{MsgBody}\}_i) = d + 1$ if $\mathit{MsgBody} \neq \texttt{register}\, P$ where d is the distance of h from $\texttt{root}$,
- $\mu(h\{\texttt{register}\, P\}_i) = d + 1 + \mu(P)$ where d is the distance of h from $\texttt{root}$,
- $\mu(\texttt{wait}_i.\, P) = \mu(P) + 1$.

We can conclude that administrative reductions are Church-Rosser also when we have immobile ambients.

Theorem 6.5 *Suppose A is well-formed. If $A \Longmapsto_{\text{adm}} A_1$ and $A \Longmapsto_{\text{adm}} A_2$ then either $A_1 \equiv A_2$, or there is A_3 such that $A_1 \Longmapsto_{\text{adm}} A_3$ and $A_2 \Longmapsto_{\text{adm}} A_3$.*

The results go through for the combination of single threaded and immobile ambients as for the case of only single threaded ambients. Note that for the translation $[\![P]\!]$ to be well formed we need the result of well typing for the combination of the single threaded and immobile ambients, Propositions 2.1 and 6.1. The extension of the mapping $\{\!\{\cdot\}\!\}$ is straightforward. The extensions of the Lemma 5.22, and Theorem 5.23 are given in the following.

Lemma 6.6 *Suppose that P is well-typed and that $[\![P]\!] \rightsquigarrow A''$. Then A'' is well-formed.*

Proof: Let $[\![P]\!]^\star = A$. Since $[\![P]\!]$ is well formed, by Lemma 6.4 we derive that A is well formed. Let $A \longmapsto_{\text{pr}} A'$; we show that A' is well formed. Therefore, since $A' \Longmapsto_{\text{adm}} A''$, we can conclude, again from Lemma 6.4, that A'' is well formed.
Let $A \equiv \boldsymbol{\nu}\overline{h}\, A_1 \parallel A_2$, $A' \equiv \boldsymbol{\nu}\overline{h}\, A_1' \parallel A_2$, and $A_1 \longmapsto_{\text{pr}} A_1'$. Consider the derivations corresponding to the application of the [LOCAL-...] rules. For all the rules involving single threaded ambients the proof is as for the corresponding case of Lemma 5.22. We consider the rules involving immobile ambients.
<u>[IMM-LOCAL-IN].</u> Let $A_1 = h{:}\,n[\,\{\mathtt{in}\, m, h'\} \mid \{\overline{\mathtt{in}}\, m, k'\}_i \mid Q'\,]_k$, and $A_1' = h{:}\,n[\,Q'\,]_k \parallel h'\{\mathtt{go}\, k'\} \parallel k'\{\mathtt{OK}\overline{\mathtt{in}}\}_i$. The proof is as for the case of [LOCAL-IN] of Lemma 5.22 .
<u>[IMM-LOCAL-OPEN].</u> Let $A_1 = h{:}\,n[\,\{\overline{\mathtt{open}}\, m, h'\} \mid \mathtt{open}\ m.\, Q \mid Q'\,]_k$, where n is an immobile ambient, and $A_1' = h{:}\,n[\,\mathtt{wait}.\, Q \mid Q'\,]_k \parallel h'\{\mathtt{migrate}\}_i$, where i is new. We verify only clauses 7, and 11; all the others are either trivial or have a proof similar to one of the previous cases. Since A is well-formed, $A_2 = h'{:}\,m[\,\mathtt{wait}.\, P \mid P'\,]_{k'} \parallel A_2'$ and h is the father of h'. Consider clause 7. The ambient in $\mathtt{wait}$ state located at h' was, in A, the source of a request (that does not occur in A') and is in A' is the target of a migrate completion (so 7a holds for it). The new ambient in $\mathtt{wait}_i$ state (located at h) is the ambient parent of the target of a migrate message, so clause 7b holds for it. Finally we want to verify clause 11 for A'. From the well-formedness of A, we have that Q' cannot contain the unboxed prefixes $\mathtt{in}, \mathtt{out}, \overline{\mathtt{open}}$ and $\mathtt{wait}$. From Lemma 5.20.1, in P we have that the (immobile) ambient n is $n[\,m[\,\overline{\mathtt{open}}\ m.\, R \mid R' \mid \cdots] \mid \mathtt{open}\ m.\, Q \mid Q' \mid \cdots]$, so $P \longrightarrow P'$ where P' contains $[\,Q \mid Q' \mid R \mid R' \mid \cdots]$ and P' is well-typed with n an immobile ambient. So, since from Proposition 6.1 we know that n cannot have the unguarded prefixes: $\mathtt{in}$, $\mathtt{out}$, $\overline{\mathtt{open}}$, and $\mathtt{wait}$, we have that clause 11 holds for A'. $\square$

Using the previous lemmas we can prove the main theorem as for the case of single threaded ambients.

Theorem 6.7 *1. If P is a well-typed SA process, and $P \longrightarrow P'$, then $[\![P]\!] \rightsquigarrow A >^\star_{\text{fw}} [\![P']\!]^\star$.*

2. If A is well-formed, $\{\!\{A\}\!\}$ is well-typed, and $A \rightsquigarrow A'$, then $\{\!\{A\}\!\} \longrightarrow \{\!\{A'\}\!\}$.

The corollary of *adequacy* again derives from Theorem 6.7 by considering the following definition of observability that takes into account also immobile ambients.

Definition 6.8 (observability) *We write $A \downarrow_n$ if*

- $A \equiv \boldsymbol{\nu}\widetilde{p}\,(\mathtt{root}{:}\,\mathtt{rootname}[\,\{M, h\} \mid P\,]_{\mathtt{rootparent}} \parallel A')$, *or*
- $A \equiv \boldsymbol{\nu}\widetilde{p}\,(\mathtt{root}{:}\,\mathtt{rootname}[\,\{M, h\}_i \mid P\,]_{\mathtt{rootparent}} \parallel A')$ *(for some i),*

where $M \in \{\overline{\mathtt{in}}\, n, \overline{\mathtt{open}}\, n\}$ and $n \notin \widetilde{p}$. We write $A \Downarrow_n$ if $A \Longmapsto\ \downarrow_n$.
Similarly for SA.

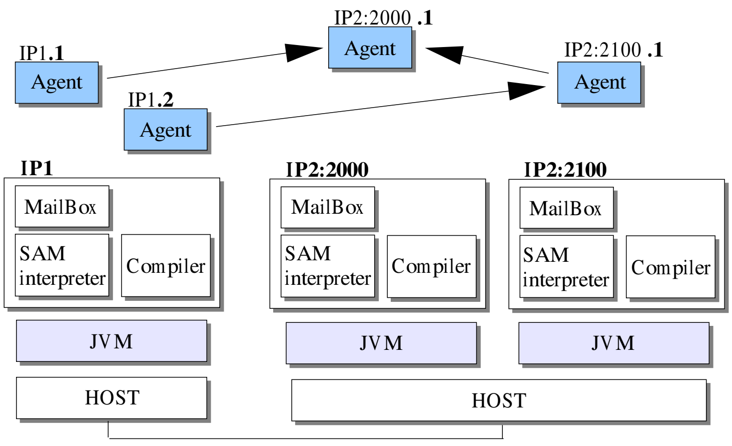

Figure 6: The three layers: agents, nodes and network (JVM plus Hosts)

7 Implementation architecture

For simplicity, in the presentation of the abstract machine we have assumed that the names of locations are atomic. In an implementation, the names of location can of course be composite (for instance, url), so that clustering of ambient (ie, multiple ambient running on the same site) is possible. What is important is that the address of an ambient (its location) univocally identifies the ambient.

Our implementation, written in Java, follows the definition of the abstract machine. As usual in implementation of process calculi, rules for arbitrary changing the order of parallel components can be taken into account with some randomisation mechanism; in our implementation we do not adopt this, which may reduce non-determinism. The main difference is that the implementation allows clustering of agents on the same IP node, i.e., a physical machine. Therefore the implementation is made of three layers: agents, nodes and the network (Figure 6). The address k of an agent is composed of the *IP-name* of the node on which its resides, plus a suffix, which is different for each agent in that node. This ensures that each agent has a unique location name. Each agent is executed by an independent Java thread; the processes local to an ambient are scheduled using a round-robin policy. Each agent knows its name, its address, its parent's address, and keeps a link to its node.

From a physical point of view, the messages exchanged between agents are of two kinds: local, when both agents reside on the same node, and remote, when two distinct nodes are involved. In each node a special Java *RMI object*, with its own thread of execution, takes care of inter-node communications. For this, nodes act alternatively as clients (requiring that a message is sent to another computer) and as servers (receiving a message and pushing it into a local mail-box). A node can also spawn a new local agent in response to a remote creation request. The node layer is implemented using Java RMI and *serialization*, and the network layer simply provides IP-name registry for RMI communications to take place (using Java *RMIregistry*).

7.1 Agents from creation to destruction.

An agent acts as an interpreter for the ambient expressions that constitute its local processes. When the agent wants to create a subambient, it sends a special message to its node, which will spawn a new agent hosting the subambient code. We also allow *remote creation* of new agents: an agent

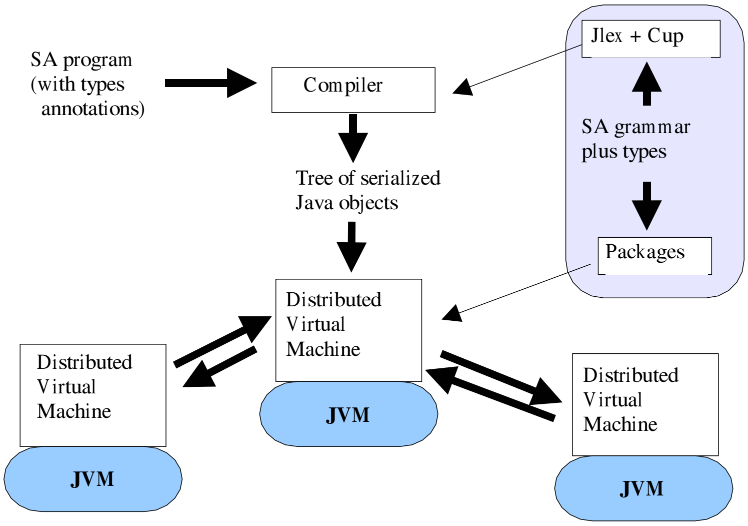

Figure 7: The software architecture of PAN

may send a message to a node different from its own, to demand the creation of a subambients. This corresponds to the addition of a primitive $\texttt{create}\, n[\,P\,]\, \texttt{at}\, h$, where h is the IP-name of a node, to the abstract machine. When the execution of an ambient expression begins on a given node, the first action is the local creation of a *root* agent. An agent resides on the same node until it is opened; then, its processes are serialised and sent via RMI to the parent agent.

A forwarder is a special kind of agent with the form $h \rhd k$, and its intended meaning is that every message arriving at h is sent to k; agents of this kind do not contain running processes, nor subambients, so they absorb very little memory and computational power, but there is no way to remove them from the system once they are created.

As already noted *in* and *out* operations simply change the logical topology between agents, and do not alter the physical distribution of processes.

8 Comparisons and remarks

Cardelli [2, 3] has produced the first implementation, called *Ambit*, of an ambient-like language; it is a single-machine implementation of the untyped Ambient calculus, written in Java. The algorithms are based on locks: all the ambients involved in a movement (three ambients for an `in` or `out` movement, two for an `open`) have to be locked for the movement to take place.

More recently, Fournet, Lèvy and Schmitt [9] have presented a distributed implementation of the untyped Ambient calculus, as a translation of the calculus into Jocaml [10] (a programming language based on the distributed Join Calculus [8]). Our abstract machine is quite different from the above mentioned implementations mainly because:

(i) We are implementing a variant of the Ambient calculus (the Safe Ambients) that has coactions and types for single-threadness and immobility.

(ii) We separate the logical and physical distribution of an ambient system.

The combination of (i) and (ii) allows us considerable simplifications, both in the abstract machine and in its correctness proof. We are not aware of correctness proofs for Ambit. The correctness proof for the Join implementation is very ingenious and makes use of sophisticated techniques, such as coupled simulation and decreasing diagram techniques. Below, we focus on the differences with the Join distributed implementation, to which we will refer as AtJ (Ambients to Join); comparations are focused on the algorithms underlying the two implementations.

Although the design of AtJ is very clever, the differences between Ambients and Join inevitably give some complications.

- In AtJ `open` is by far the most complex operation, because the underlying Jocaml language does not have primitives with a similar effect. In AtJ, every ambient has a manager that collects the requests of operations from the subambients and from the local processes. If the ambient is opened, its manager becomes a forwarder of messages towards the parent ambient. The processes local to the opened ambient are not moved.

  As a consequence, in AtJ the processes local to an ambient can be distributed on several locations (precisely: to sublocations of the location of the given ambient). Therefore, also the implementation of the communication rule R-MSG may require exchange of messages among sites, which does not occur in PAN, where forwarders are always empty.

  Moreover it is important to notice that PAN is an abstract machine, therefore it is independent of a specific target language; so many additional implementation improvements can eventually be achieved by carefully adopting different target language (similar considerations hold also for coordination languages such as Linda).

- In AtJ's algorithms, forwarders are also introduced with `in` and `out` operations, to cope with possible asynchronous messages still travelling after the move is finished (the so called *in-fligth* messages). These forwarders are not needed in PAN, nor in its implementation.

- In PAN, the presence of coactions dispenses us from having backward pointers from an ambient to its children. In the example of Figure 1, without $\overline{\texttt{in}}$, the ambient c would not know the location of b and therefore could not communicate this location to a. Backward pointers, present in AtJ, make bookkeeping and correctness proof more complex.

  In PAN, the absence of backward pointers and the presence of coactions make the implementation of forms of dynamic linking straightforward: new machines hosting ambients can be connected to existing machine running an ambient system; it suffices that the new machines know the location (the IP number) of one of the running ambients; no modifications or notifications is needed to the running ambients themselves.

- In PAN, since any moving ambient (an ambient that tries to enter or exit another ambient, or that can be opened) is single-threaded, each moving ambient requests at most one operation at a time to its parent. By contrast, in AtJ an ambient can send an unbounded number of requests to the parent (an example is $n[\,!\texttt{in}\, m_1 \mid !\texttt{out}\, m_2\,]$).

  Due to this property, in PAN no ambient needs a log of pending requests received from a given children or sent to the parent. Without the property, both forms of log are needed, as it happens in AtJ. To see why, consider two ambients a and b, where b is the parent of a. If moving ambients can request several operations concurrently, b must of course keep a log of the pending requests from a. A copy of the same log must however be kept by a, because messages exchanged among ambients are asynchronous and therefore the following situation could arise. Suppose a requests two operations, say $\texttt{in}\, n$ and $\texttt{in}\, m$. The request for $\texttt{in}\, n$

could reach b first. The request for $\mathtt{in}\,m$ could reach b only when the movement for $\mathtt{in}\,n$ has been completed (indeed, a might have completed other movements). The request $\mathtt{in}\,m$ must now be resent to the new parent of a, but b does not possess this information. This task must therefore be accomplished by a, which, for this, must have stored $\mathtt{in}\,m$ in its log of pending requests to the parent.

The example also shows that, aside from message retransmission in forwarders, some requests may have to be retransmitted several times, to different parents (in the example, $\mathtt{in}\,m$); in PAN every request is sent at most once.

If we consider the extension to immobile ambients (with the bang) situations as $n[\,!\overline{\mathtt{in}}\,n \mid P\,]$ could occur, but in PAN the immobile ambient will not *move*, therefore it will be there to recieve any answer for its parent, and again no logs are needed and messages are sent just once.

- In PAN, any movement for a given ambient is requested to the parent, which (assuming this is not a forwarder) makes decisions and gives authorisations; the grandparent is never contacted. This homogeneity property breaks in presence of backward pointers from an ambient to its children. For instance, the simulation of the `out` reduction of Figure 2 would then need also the involvement of the grandparent c.
- In AtJ, the domain of physical distribution is a tree. The `in` and `out` operations produce physical movements in which an ambient, and all its tree of subambients, must move. To achieve this, the tree of ambients is first "frozen" so that all the activities in the ambients of the tree stop while the movement takes place. In PAN, where the domain of physical distribution is flat, `in` and `out` only give logical movement; no freezing of ambients is required. On the other hand in PAN`open` gives physical movement, which is not the case in AtJ.

9 Further developments

In this section we present some improvements and extensions that may be done, on one side to the abstract machine, and on the other to its implementation.

9.1 Immobile ambients

To handle immobile ambients, we have extended the syntax of PAN, and the set of reduction rules. We believe that these extensions are not necessary: the same syntax and rules presented for single-threaded ambients work also in the presence of immobile ambients. However, the correctness proof is harder.

We describe an improvement to the solution of adopting the rules of Section 4. Consider the process

$$P \stackrel{\text{def}}{=} n[\,\mathtt{rec}\,X.\,(\overline{\mathtt{in}}\,n \mid \boldsymbol{\nu} m\,(\mathtt{open}\,m.\,X \mid m[\,\overline{\mathtt{open}}\,m\,]))\,]$$

(Using replication, the behaviour of P can be expressed as $n[\,!\overline{\mathtt{in}}\,n\,]$.) where $\simeq$ is behavioural equivalence (Section 4). With the rules of Section 4, ambient n could flood its parent with $\overline{\mathtt{in}}$ requests. We can avoid the problem by modifying PAR-PROC thus:

$$\frac{P \xrightarrow[h:n]{k} P' \gg \widetilde{Msg} \quad \begin{array}{l} n \text{ is an immobile ambient} \\ Q \text{ does not have unguarder ambients} \\ Q \text{ or } P' \text{ do not contain any } \mathtt{wait} \end{array}}{P \mid Q \xrightarrow[h:n]{k} P' \mid Q \gg \widetilde{Msg}} \qquad \text{[IMM-PAR-PROC]}$$

We then have to modify also LOCAL-OPEN and PAR-PROC, so that an immobile ambient does not go into a $\mathtt{wait}$ state while opening a child ambient:

$$\frac{n \text{ is an immobile ambient}}{\mathtt{open}\, m.\, P \mid \{\overline{\mathtt{open}}\, m, h\} \xrightarrow[-:n]{-} P \gg h\{\mathtt{migrate}\}} \qquad \text{[IMM-LOCAL-OPEN]}$$

$$\frac{n \text{ is an immobile ambient}}{h\colon n[\,P\,]_k \parallel h\{\mathtt{register}\, R\} \longmapsto h\colon n[\,P \mid R\,]_k} \qquad \text{[IMM-COMPL-REG]}$$

The original rules LOCAL-OPEN, PAR-PROC, and COMPL-REG are now used only for ST ambients, therefore the corresponding side conditions is added.

Rule IMM-LOCAL-OPEN could actually be used also for ST ambients. This however makes the correctness proof harder and under some more refined notion of behavioural equivalence (example: a real-time model) would not be correct.

The effect of adopting rule IMM-PAR-PROC in place of PAR-PROC is that an immobile ambient sends to its father only one $\overline{\mathtt{in}}$ request at a time. This property already holds for ST ambient with the rules of Section 4. An immobile ambient can exercise several capabilities at the same time. Sending one request at a time to the parent is correct because the only capability that may produce a request from an immobile ambient named n to its parent is $\overline{\mathtt{in}}\, n$ (the protocol for $\overline{\mathtt{in}}$ can however be executed in parallel with several protocols for $\overline{\mathtt{out}}$ and $\mathtt{open}$ operations).

9.2 Forwarders

In the abstract machine presented, a message may have to go through a chain of forwarders before getting to destination. A (partial) solution to this problem is a modification of the rules that guarantees the following property: every agent sends a message to a given forwarder at most once. The modification consists in adding the source field to the completion messages $h\{\mathtt{OK}\overline{\mathtt{in}}\}$, which thus becomes $h\{\mathtt{OK}\overline{\mathtt{in}}, k\}$, where k is the ambient that is authorising the move. Thus the rules LOCAL-IN and COMPL-COIN become

$$\{\mathtt{in}\, n, h\} \mid \{\overline{\mathtt{in}}\, n, k\} \xrightarrow[h':-]{-} \mathbf{0} \gg h\{\mathtt{go}\, k\} \parallel k\{\mathtt{OK}\overline{\mathtt{in}}, h'\} \qquad \text{[LOCAL-IN2]}$$

$$h\colon n[\,P \mid \mathtt{wait}.\, Q\,]_k \parallel h\{\mathtt{OK}\overline{\mathtt{in}}, h'\} \longmapsto h\colon n[\,P \mid Q\,]_{h'} \qquad \text{[COMPL-COIN2]}$$

The reason why these rules may be useful is that the parent of an ambient that has sent a $\overline{\mathtt{in}}$ request may have become a forwarder; thus the real parent is another ambient further up in the hierarchy. With the new rules, the parent of the ambient that has sent the $\overline{\mathtt{in}}$ request is updated and hence this ambient will not go through the forwarder afterwards. With the other capabilities that may originate a request from an ambient to his parent ($\mathtt{open}$, $\mathtt{out}$, $\mathtt{in}$), the issue does not arise, because either the requesting ambient is dissolved ($\mathtt{open}$), or its parent is modified ($\mathtt{out}$, $\mathtt{in}$).

Even with the rules above, however, the forwarder introduced by an $\mathtt{open}$ operation is permanent. We plan to study the problem of the garbage-collection of forwarders. We also plan to experiment the addition of backwards pointers, from an ambient to its children; this should avoid the introduction of forwarders in an $\mathtt{open}$, however, it is likely to complicate other parts of the abstract machine. A more long-term goal is the study of optimisations of the abstract machine to enhance parallelism.

9.3 Other issues

At the moment, we have an implementation of the absatrct machine, a compiler from SA to Java, and a simple user interface that allows the execution of SA terms. We hope that this will be helpful for the design of implementations of programming languages based on ambients. To be useful, in assessing the practical impact of the SA calculus, however, we have to embed this core language into a real language. We intend to pursue this direction, taking advantage also of the experience of π-calculus-based programming languages such as Pict and Join. An orthogonal direction would be to make the ambient constructs into a framework that could be used in conjunction with, e.g., Java.

Acknowledgements.

We have benefitted from comments by Jean-Jacques Lèvy and Alan Schmitt.

References

[1] M. Bugliesi and G. Castagna. Secure safe ambients. In *Proc. 28th POPL*. ACM Press, 2001.

[2] L. Cardelli. Ambit. `http://www.luca.demon.co.uk/Ambit/Ambit.html` 1997.

[3] L. Cardelli. Mobile ambient synchronisation. Technical Report 1997-013, Digital SRC, 1997.

[4] L. Cardelli and A.D. Gordon. Mobile ambients. *Proc. FoSSaCS '98*, LNCS 1378, pages 140–155. Springer Verlag, 1998.

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

[6] L. Cardelli and A.D. Gordon. Types for mobile ambients. In *Proc. 26th POPL*, pages 79–92. ACM Press, 1999.

[7] L. Cardelli and A.D. Gordon. Anytime, anywhere: Modal logics for mobile ambients. In *Proc. 27th POPL*. ACM Press, 2000.

[8] C. Fournet, G. Gonthier, J.-J. Lévy, L. Maranget, and D. Rémy. A calculus of mobile processes. *Proc. CONCUR '96*, LNCS 1119, pages 406–421. Springer Verlag, 1996.

[9] C. Fournet, J.-J. Lévy, and A. Schmitt. An asynchronous distributed implementation fo mobile ambients. *IFIP TCS2000*, LNCS 1872, pages 348–364. Springer Verlag, 2000.

[10] F. Le Fessant. The Jocaml system prototype. `http://join.inria.fr/jocaml`. 1998.

[11] F. Levi and D. Sangiorgi. Controlling interference in ambients. In *Proc. 27th POPL*. ACM Press, 2000.

[12] F. Nielson and H.R. Nielson Shape analysis for mobile ambients. In *Proc. 27th POPL*, pages 142–154, N.Y., January 19–21 2000. ACM Press.

[13] D. Sangiorgi and R. Milner. The problem of "Weak Bisimulation up to". *Proc. CONCUR '92*, LNCS 630, pages 32–46. Springer Verlag, 1992.