University "Alexandru Ioan Cuza" of Iaşi
Faculty of Computer Science

TECHNICAL REPORT

Embedding Mobile Ambients into the π-calculus

Gabriel Ciobanu, Vladimir Zakharov

TR 05-07, November 2005

ISSN 1224-9327

Universitatea "Alexandru Ioan Cuza" Iaşi
Facultatea de Informatică
Str. Berthelot 16, 6600-Iaşi, Romania
Tel. +40-32-201090, email: bibl@infoiasi.ro

Embedding mobile ambients into the π-calculus

Gabriel Ciobanu
Institute of Computer Science, Romanian Academy
Blvd. Carol I nr.8, 700505 Iaşi, Romania
Email: gabriel@iit.tuiasi.ro

Vladimir A. Zakharov
Faculty of Computational Mathematics and Cybernetics,
Lomonosov State University, Moscow RU-119899, Russia
Email: zakh@cs.msu.su

Abstract

The mobile ambient paradigm of mobile computations is very expressive and allows various embeddings into it of other models of computations such as Turing machine, the λ-calculus, and the π-calculus. In this paper we demonstrate that we have a closer relationship between mobile ambients and the π-calculus. We present an embedding of pure mobile ambients into a fragment of the π-calculus, namely the localized sum-free synchronous monadic π-calculus with matching and mismatching. The embedding is achieved by associating a pair of localized π-terms $Structure_A$ and *Ruler* to every mobile ambient A. $Structure_A$ simulates the spatial structure of A by means of communication channels. *Ruler* is a universal π-process intended to simulate the operational semantics of the mobile ambients. This embedding of pure mobile ambients into a subset of the π-calculus opens new ways to the analysis of ambients by means of techniques used in the π-calculus.

Keywords: concurrent and distributed systems, mobile ambients, π-calculus.

1 Introduction

We have many formalisms to describe mobile computation. Among them, the π-calculus [8] is a widely accepted model of interacting systems with dynamically evolving communication topology. Its mobility is expressed by the changing configuration and connectivity among processes. This mobility increases the expressive power of the π-calculus, and the π-calculus is a general model of computation which takes interaction as a primitive. The π-calculus models computing by reduction and interactive matching. Both the sender and receiver offer their availability for communication. Similar mechanisms work in computation and in biology [4, 5].

Another successful formalism for mobility is provided by mobile ambients [2]. The mobility is given by the movement of computational environments called ambients which possibly contain interacting processes. It is known that the Mobile Ambient calculus can be used for simulating the π-calculus computations [2, 3, 10]. There are very few results revealing the relationship between the mobile ambients and the π-calculus [6]. In this paper we present the embedding of the mobile ambient calculus into a subset of π-calculus, namely into the localized sum-free synchronous monadic π-calculus. In this way we emphasize a close link between two important formalisms for mobility, and show the high expressive power of the localized sum-free π-calculus. The localized π-calculus is adequate and sufficient to simulate the behaviour of pure mobile ambients.

2 Mobile Ambients

We present here a short description of pure mobile ambient calculus (MA) as it was introduced in [2].

Definition 1 *Given the infinite set of names $\mathcal{N}$ (ranged over by $m, n, \ldots$) we define the sets $\mathcal{A}$ of MA-processes (denoted by $A, A', B, \ldots$) and capabilities (denoted by $M, M', \ldots$) as follows*

$$\begin{array}{lcl} M & ::= & in\ n \mid out\ n \mid open\ n \\ A & ::= & \mathbf{0} \mid A|B \mid\ !A \mid M.A \mid n[A] \mid (\nu n)A \end{array}$$

Processes of the form $M.A$ and $n[A]$ are called *actions* and *ambients*, respectively. The free names of MA-process A are denoted by $fn(A)$. Hereafter, it will be convenient to assume an external ambient $\top \notin \mathcal{N}$. For every process we define the set of top-level subprocesses as follows:

1. $TL(\mathbf{0}) = \emptyset$; if $A = M.B$ or $A = n[B]$, then $TL(A) = \{A\}$;
2. if $A = B_1|B_2$, then $TL(A) = TL(B_1) \cup TL(B_2)$;
3. if $A =!B$ or $A = (\nu n)B$, then $TL(A) = TL(B)$.

For each action $A = M.B$ or ambient $A = n[B]$ we will say that $TL(B)$ is the *lower context* of A and A is the *upper context* of every process from $TL(B)$.

Definition 2 *The* structural congruence $\equiv$ *on MA-processes is the least congruence satisfying the following requirements.*

- $(\mathcal{A}, |, \mathbf{0})$ *is a commutative monoid;*
- $(\nu m)A \equiv (\nu n)A\{n/m\}$, $(\nu n)(B|A) \equiv A|(\nu n)B$;
- $n \neq m \Rightarrow (\nu n)m[A] \equiv m[(\nu n)A]$;
- $(\nu n)\mathbf{0} \equiv \mathbf{0}$, $(\nu n)(\nu m)A \equiv (\nu m)(\nu n)A$, $!A \equiv A|!A$.

Definition 3 *The operational semantics of pure ambient calculus is defined in terms of a reduction relation* $\rightarrow_a$ *by the following axioms and rules.*

Axioms:

(In) $n[in\ m.A|A']|m[B] \ \rightarrow_a \ m[n[A|A']|B]$;

(Out) $m[n[out\ m.A|A']|B] \ \rightarrow_a \ n[A|A']|m[B]$;

(Open) $open\ n.A|n[B] \ \rightarrow_a \ A|B$.

Rules:

(Res) $\dfrac{A \ \rightarrow_a \ A'}{(\nu n)A \ \rightarrow_a \ (\nu n)A'}$; (Comp) $\dfrac{A \ \rightarrow_a \ A'}{A|B \ \rightarrow_a \ A'|B}$;

(Amb) $\dfrac{A \ \rightarrow_a \ A'}{n[A] \ \rightarrow_a \ n[A']}$; (Struc) $\dfrac{A \equiv A',\ A' \ \rightarrow_a \ B',\ B' \equiv B}{A \ \rightarrow_a \ B}$.

3 The π-calculus

In this section we review the syntax and operational semantics of synchronous monadic π-calculus without sum and τ-prefix operators.

Definition 4 *Given the infinite set* $\mathcal{V}$ *of variables (ranged over by* $x, y, \ldots$*) and the infinite set* $\mathcal{N}$ *of channel names (ranged over by* $m, n, \ldots$*) we define the set* $\mathcal{P}$ *of* π*-processes (denoted* $P, Q, \ldots$*) as follows*

$$\begin{array}{lcl} M & ::= & x \ \mid \ n \\ P & ::= & \mathbf{0} \mid P|Q \mid !P \mid \overline{M}\langle M'\rangle.P \mid M(x).P \mid [M = M'](P,Q) \mid (\nu n).P \end{array}$$

The set of free names in a π-process P is denoted by $fn(P)$. Without loss of generality, we consider the monadic variant of the π-calculus. A π-process P is called *localized* if P has no subprocesses of the form $M(x).Q$, $[x = M](Q,R)$, or $[M = x](Q,R)$ such that the name x occurs free in Q as the subject of an input prefix $x(y).Q'$. As demonstrated in [9], the locality property may be very useful for the analysis of π-calculus terms as well as for developing distributive implementations of the language.

Definition 5 *The* structural congruence $\equiv$ *on* π*-processes is the least congruence satisfying the following requirements*

- $(\mathcal{P}, |, \mathbf{0})$ *is a commutative monoid;*
- *if* $P \equiv P'$ *then* $P|Q \equiv P'|Q$, $\overline{M}\langle M'\rangle.P \equiv \overline{M}\langle M'\rangle.P'$, $M(x).P \equiv M(x).P'$, $[M = N](P,Q) \equiv [M = N](P',Q)$, $[M = N](Q,P) \equiv [M = N](Q,P')$, $(\nu n).P \equiv (\nu n).P'$ *hold;*
- *if* $n \notin fn(P)$, *then* $(\nu n).P \equiv P$, $(\nu m).P \equiv (\nu n).P\{n/m\}$, $(\nu n).(P|Q) \equiv P|(\nu n).Q$ *hold;*
- $(\nu n).\mathbf{0} \equiv \mathbf{0}$, $(\nu n).(\nu m).P \equiv (\nu m).(\nu n).P$, $!P \equiv P|!P$, $[M = M'](P,Q) \equiv [M' = M](P,Q)$.

Definition 6 *The operational semantics of π-calculus is given in the form of a one-step reduction relation $\rightarrow_\pi$ by the following axioms and rules.*

Axioms:

(Comm) $\overline{n}\langle m\rangle.P \mid n(x).Q \ \rightarrow_\pi \ P|Q\{m/x\}$;

(Match) $[n = n](P, Q) \ \rightarrow_\pi \ P$;

(MisMatch) $[n = m](P, Q) \ \rightarrow_\pi \ Q$, *where* $n \neq m$.

Rules:

(Res) $\dfrac{P \ \rightarrow_\pi \ P'}{(\nu n).P \ \rightarrow_\pi \ (\nu n).P'}$; (Comp) $\dfrac{P \ \rightarrow_\pi \ P'}{P|Q \ \rightarrow_\pi \ P'|Q}$;

(Struc) $\dfrac{A \equiv A',\ A' \ \rightarrow_\pi \ B',\ B' \equiv B}{A \ \rightarrow_\pi \ B}$;

We denote by $\rightarrow_\pi^*$ the transitive and reflexive closure of $\rightarrow_\pi$. A π-process P is called *deterministic* if for every pair of π-calculus terms Q_1, Q_2 such that $P \rightarrow_\pi Q_1$ and $P \rightarrow_\pi Q_2$, we have $Q_1 \equiv Q_2$. Otherwise P is called *non-deterministic*. Whenever P is a deterministic (non-deterministic) and $P \rightarrow_\pi Q$ we will use notation $P \mapsto_\pi Q$ (respectively $P \hookrightarrow_\pi Q$) for a one-step reduction. We will write $P \hookrightarrow_\pi^i Q$ for a chain of i non-deterministic reduction steps and $\mapsto_\pi^*$ for the transitive and reflexive closure of $\mapsto_\pi$. If $P \rightarrow_\pi^* Q$ and there are no processes R such that $Q \rightarrow_\pi R$, then we say that $P \rightarrow_\pi^* Q$ is a *terminating run of* P.

4 Translation of Mobile Ambient into π-calculus

In this section we introduce a relationship between MA-processes and π-processes. This relationship may be thought of as a non-deterministic embedding of pure ambients into π-processes: with every MA-process A it associates a set $[\![A]\!]$ of π-processes. In the next section we will prove the correctness of the embedding by demonstrating that each π-process from $[\![A]\!]$ simulates the behaviour of A. The only purpose of considering $[\![\cdot]\!]$ as a relation is to simplify the proofs of its correctness. When using our translation in applications one may take a single (minimal) π-process from $[\![A]\!]$ as a true image of A.

A specific feature of pure ambient calculus is that an MA-process A has a spatial tree-like structure which serves a dual function. On the one hand, mobile ambients control the run of processes in A by bounding the scope of actions. On the other hand, the mobile ambients of A are acted upon the spatial structure of A. When translating A into a π-process we separate these functions of mobile ambients. An ambient A is translated into a π-process $Proc_A = Structure_A|Ruler|Environment$ which is a composition of three π-processes $Structure_A$, $Ruler$ and $Environment$. The process $Structure_A$ is designed according to the following principles.

1. The mobile ambients and capabilities from A are represented by individual subprocesses Amb_i and Act_j; we will call these π-calculus terms *nodes*. The spatial structure of A is maintained by means of specific tree-wire

subprocesses TW_k that are used for communication between nodes that represent ambients and actions. Thus, we have

$$Structure_A = Amb_1|\ldots|Amb_N|Act_1|\ldots|Act_M|TW_1|\ldots|TW_L$$

where Amb_i, Act_j, and TW_k represent generic notations for ambients, capabilities, and tree-wire structure.

2. Each subprocess Amb_i is associated with some mobile ambient $n[P]$ in A. It keeps the name n of the ambient and provides communication between the ambient and its upper and lower contexts.

3. Each subprocess Act_j is associated with some action of the form *in* $n.P$, *out* $n.P$ or *open* $n.P$ in A. It keeps the type of capability (*in*, *out* or *open*) and the name n and also provides communication between the action and its upper and lower context.

4. A subprocess TW_k is a set of wires arranged into a tree-like structure. TW_k delivers requests from its leaf nodes to the root and sends back replies from the root to the leaves. A tree-wire subprocess is intended to provide message exchange between the nodes and to accumulate consumed capabilities and dissolved ambients of A.

5. When a capability is consumed or an ambient is dissolved, a corresponding node becomes passive. A passive node rearranges to a wire and adds itself to some tree-wire. Thus, the wires of $Structure_A$ take account of computation steps generated by A. Since there are many different ways to derive the same mobile ambient term A, it may be encoded into a whole set of π-calculus terms which have the same nodes and differ only in structures of their tree-wires.

The subprocess *Ruler* does not depend on A. It is a universal π-process intended for simulating the operational semantics of mobile ambients. An execution of *Ruler* conforms to the following scenario.

1. *Ruler* selects from $Structure_A$ an arbitrary triple of nodes *Act*, Amb_{i_1} and Amb_{i_2} and collects the information about their types, names and links.

2. If the type of *Act* is *in*, *Act* is linked with Amb_{i_1} and keeps the same name as Amb_{i_2}, whereas Amb_{i_1} and Amb_{i_2} are linked with the same node in $Structure_A$, then the subprocess $Act|Amb_{i_1}|Amb_{i_2}$ corresponds to a mobile ambient pattern $n[in\ m.P|Q]|m[R]$. In this case *Ruler* simulates the implementation of an entry instruction by switching the link of Amb_{i_1} to Amb_{i_2} and converting the node *Act* into a wire. This changes the entry-pattern $n[in\ m.P|Q]|m[R]$ into $m[n[P|Q]|R]$.

3. If the type of *Act* is *out*, *Act* is linked with Amb_{i_1} and keeps the same name as Amb_{i_2}, whereas Amb_{i_1} is linked with Amb_{i_2}, then the subprocess $Act|Amb_{i_1}|Amb_{i_2}$ corresponds to a pattern $m[n[out\ m.P|Q]|R$. In this case *Ruler* simulates the implementation of an exit instruction by converting

the node *Act* into a wire, and directing the link of Amb_{i_1} to the same destination where the link of Amb_{i_2} is directed to. This changes the exit-pattern $m[n[out\ m.P|Q]|R$ into $m[R]|n[P|Q]$.

4. If the type of *Act* is *open*, *Act* keeps the same name as Amb_{i_1} and both *Act* and Amb_{i_1} are linked with Amb_{i_2}, then the subprocess $Act|Amb_{i_1}|Amb_{i_2}$ corresponds to a pattern $m[open\ n.P|n[Q]]$. In this case *Ruler* simulates the implementation of an open instruction by converting both *Act* and Amb_{i_1} into wires. This changes the open-pattern $m[open\ n.P|n[Q]]$ into $m[P|Q]$.

5. If none of the above cases 2), 3) and 4) holds, then *Ruler* tries another triple of nodes Act', Amb_{j_1} and Amb_{j_2} from $Structure_A$.

The subprocess *Environment* plays a role of a virtual external environment of mobile ambients. It bounds every MA-process and can not be dissolved. When *Ruler* simulates an open operation, it can select *Environment* for Amb_{i_2}.

Now we present the formal description of the processes involved in *Structure*, *Ruler* and *Environment* and define the embedding relation between pure mobile ambients and π-processes.

4.1 Tree-wire processes

A tree-wire process $TW_{I,k}$, where I is a set of names and k is a name such that $k \notin I$, is composed of basic wires $TW_{I,k} = W_{x_1,y_1}|\ldots|W_{x_m,y_m}$. Each wire $W_{x,y}$ serves the message passing from the agent named x to the agent named y and back. The wires $W_{x_1,y_1},\ldots,W_{x_m,y_m}$ are joined together so that $TW_{I,k}$ has a tree-like communication structure which provides message exchange between the set of leaf nodes I and the root k. The formal description of a wire $W_{x,y}$, $x \neq y$, is as follows.

Definition 7 $W_{x,y} = x(v).\ (\nu u).\ \overline{y}\langle u\rangle.\ u(t).\ \overline{v}\langle t\rangle.\ W_{x,y}$

Definition 8 *The set of tree-wires is the minimal set of π-processes satisfying the following requirements.*

1. *Every wire $W_{x,y}$ is a tree-wire $TW_{\{x\},y}$.*

2. *If $W_{x,y}$ is a wire, $TW_{I,x}$ is a tree-wire and $y \notin I$, then the π-calculus term $(\nu x).\ (W_{x,y}|TW_{I,x})$ is a tree-wire $TW_{I,y}$.*

3. *If $TW_{I,y}$ and $TW_{J,y}$ are tree-wires such that $I \cap J = \emptyset$, then the π-calculus term $TW_{I,y}|TW_{J,y}$ is a tree-wire $TW_{I\cup J,y}$.*

When dealing with a tree-wire $TW_{I,y}$, where $I = \{x_1, x_2, \ldots, x_n\}$, we use the shorthand notation $(\nu I).(P|TW_{I,k})$ for $(\nu x_1).(\nu x_2).\ldots.(\nu x_n).(P|TW_{I,k})$.

Proposition 1 *If $TW_{I,y}$ is a tree-wire, then $y \notin I$ & $fn(TW_{I,y}) = I \cup \{y\}$.*

When appending a tree-wire to a tree-wire we get a tree-wire.

Proposition 2 *Let TW_{I_1,y_1} and TW_{I_2,y_2} be a pair of tree-wires such that $I_1 \cap I_2 = \emptyset$, $y_1 \notin I_2$, $y_2 \in I_1$. Then $(\nu y_2).\ (TW_{I_1,y_1}|TW_{I_2,y_2}) \equiv TW_{I_3,y_1}$, where $I_3 = (I_1 - \{y_2\}) \cup I_2$.*

A tree-wire delivers requests from its leaf nodes to the root and replies from the root to leaf nodes.

Proposition 3 *Let $TW_{I,y}$ be a tree-wire and $x \in I$. Then the π-process*

$$(\nu v).\ \overline{x}\langle v\rangle.\ v(z).\ z|TW_{I,y}|y(u).\ \overline{u}\langle t\rangle$$

has a deterministic terminating run

$$(\nu v).\ \overline{x}\langle v\rangle.\ v(z).\ z|TW_{I,y}|y(u).\ \overline{u}\langle t\rangle \ \mapsto^*\ t|TW_{I,y}$$

4.2 Ambients and actions

The main difficulty of embedding pure ambient processes into π-processes is that of simulating the consumption of capabilities, dissolving the boundaries of ambients, and changing the structure accordingly. In an MA-process, when an action *in* $n.P$, *out* $n.P$ or *open* $n.P$ is executed, the corresponding capability just disappears from the process (it is consumed). The same effect manifests itself when the boundary of an ambient named $m[Q]$ is dissolved. But when simulating actions and ambients as individual π-subprocesses of $Structure_A$ it is not possible just to reduce the consumed capabilities or dissolved ambients to inactive processes $\mathbf{0}$ since in this case we lose the links between the processes in $Structure_A$. The simplest way to make such processes inactive while preserving a tree-like structure of links between the remaining processes is to convert consumed capabilities and dissolved ambients into wires and use them merely to maintain links between the active processes. In this case the process $Structure_A$ corresponding to an MA-process A will depend not only on the spatial structure of A, but also on the way A is computed (the history of A). That is why instead of using deterministic embedding which maps every MA-process into a single π-process we introduce an embedding relation $\models$ which associates every MA-process A with a set of π-processes $[\![A]\!]$. Each process $Structure_A$ from $[\![A]\!]$ keeps along with the spatial structure of A the possible history of A, i.e. the way the MA-process A can be computed from the other processes. This history is represented by wires which keep track of consumed capabilities and dissolved ambients. The history of A does not influence the functionality of $Structure_A$; its only purpose is to maintain the links between active nodes of $Structure_A$.

The π-processes *Act* and *Amb* associated with actions *cap* $n.P$, where *cap* $\in \{in, out, open\}$, and ambients $n[P]$ in MA-process A have a similar arrangement. An action represented by a capability *cap* n is encoded into the π-process $Node(cap, n, up, down, s_{act}, label)$, where *up* and *down* are names of channels used for communication with upper and lower contexts of the action, s_{act} is a channel name shared by all action-type nodes of $Structure_A$ for communications with *Ruler*, and *label* is an individual label of the action in A. An ambient named n is encoded into the π-process $Node(amb, n, up, down, s_{amb}, label)$,

where *up*, *down* and *label* have the same meaning as above, *amb* is the key word for distinguishing ambients from capabilities, and s_{amb} is a channel name shared by all ambient-type nodes of $Structure_A$ for communications with *Ruler*.

Definition 9 *The formal description of a π-process $Node(a,n,u,d,s,l)$ is as follows.*

$$
\begin{array}{lcll}
Node(a,n,u,d,s,l) & = & Reply(d,s,l)|Main(a,n,u,d,s,l) & \\
 & & & \\
Reply(d,s,l) & = & d(y).\ [y=l] & (1)\\
 & & (& \\
 & & \quad d(u).\ \ \overline{s}\langle l\rangle.\ W_{d,u}\ , & (2)\\
 & & \quad \overline{y}\langle l\rangle.\ Reply(d,s,l) & (3)\\
 & &) & \\
Main(a,n,u,d,s,l) & = & (\nu v).\ (\nu w).\ (\nu k). & (4)\\
 & & \overline{s}\langle v\rangle.\ v(x). & (5)\\
 & & \overline{u}\langle w\rangle.\ w(l_u). & (6)\\
 & & \overline{x}\langle a,n,l,u,d,l_u\rangle.\ v(y,z). & (7)\\
 & & [y=l] & (8)\\
 & & (& \\
 & & \quad \overline{d}\langle l\rangle.\ \overline{d}\langle z\rangle\ , & (9)\\
 & & \quad \overline{d}\langle k\rangle.\ k(w'). & (10)\\
 & & \quad \overline{s}\langle l\rangle.\ Main(a,n,z,d,s,l) & (11)\\
 & &) &
\end{array}
$$

The π-process $Node(a,n,u,d,s,l)$ is a recursive process composed of two subprocesses $Reply(d,s,l)$ and $Main(a,n,u,d,s,l)$. The subprocess *Reply* serves the dual function of providing communication with the lower context of a node (which is a set of nodes) and also of converting (if necessary) the node into a wire. The subprocess $Main$ keeps the information about the node (its type, name and context) and communicates with *Ruler*. Below we briefly explain the intended meaning of some fragments of *Reply* and $Main$.

(1) $Reply(d,s,l)$ can receive via the channel d either a request from the lower context of the node, or an instruction from $Main$ which alters the whole node into a wire. In the former case y is evaluated into a private channel name for emitting at y the label l of the node. In the latter case the main process evaluates y into the label l (see line (9)) which does not match any private name. Therefore, after receiving the label l from $Main$ the process *Reply* is reduced to the line (2), whereas after receiving a request from the lower context of the node it is reduced to the line (3).

(2) After receiving the label l of the node from the main subprocess of the node, *Reply* receives an updated channel name for communication with the upper context of the node (see line (9)), sends a synchronization message to *Ruler* indicating thus the completion of the instruction processing, and evolves into a wire which connects the lower and the upper contexts of the node. This may happen when the node becomes passive since the corresponding capability is consumed or the ambient boundary is dissolved.

(3) If *Reply* receives a request from the lower context of the node it considers the value of y as a private name of a channel and sends via this channel the label

l of the node. As a consequence, the label of the node becomes available to its lower context. Afterwards *Reply* reverts to the original state.

(4) The subprocess $Main(a, n, u, d, s, l)$ uses the following private names:
   - v as a name of a channel for receiving acknowledgements and instructions from *Ruler*,
   - w as a name of a channel for receiving the label of a node that precedes our node in $Structure_A$,
   - k as an arbitrary fresh name different from the label of the node to switch *Reply* to the line (3).

(5) $Main$ begins with sending a message to *Ruler* to indicate the readiness of the node to participate in the simulation of some MA operation. *Ruler* will consider this message as a private name of a channel for communication with the node. If *Ruler* selects the node for simulating MA operation it sends via v another private name (see lines (14),(15)). This name will be used as a channel for sending to *Ruler* additional information: the name, the type and the environment of the node.

(6) The node sends a request to its upper context to know the label of preceding node in the $Structure_A$. The upper context replies via w and evaluates l_u to the label of the predecessor see (see lines (1), (3) and Proposition 3).

(7) The node sends to *Ruler* its type, name and label, the channel names for communication with the upper and lower context, the label of its predecessor in $Structure_A$. After processing this information *Ruler* replies via v to the node and informs it about its new status (active or passive) and its new upper context (see lines (17), (19), (21) and (22)). If the node does not match a pattern for simulating an MA operation or corresponds to a component of MA which is not consumed or dissolved along the operation, then *Ruler* emits at v some private name which does not match l. The node should consider this private name as an instruction to remain active. Otherwise *Ruler* evaluates y into the label l of the node and this is considered as an instruction to alter the node into a wire. In both cases *Ruler* evaluates z into a name of a new channel for communication with the upper context of the node (since the context of the node may be also changed as a result of simulation of MA operation).

(8) $Main$ checks the instruction.

(9) If the node becomes passive due to the consumption of a capability or dissolution of an ambient, then *Reply* is instructed to become a wire and receives an updated channel name for communication with its upper context. In this case the main subprocess of the node is reduced to $\mathbf{0}$.

(10) Otherwise the subprocess *Reply* is informed that the node remains active. The input action $k(w')$ is used just for the sake of uniformity.

(11) Afterwards the main subprocess sends to *Ruler* a synchronization message which indicates completion of the instruction processing, updates its upper context and reverts to the original state.

4.3 Simulating the operational semantics of pure ambients

The π-process $Structure_A$ represents only the spatial structure of MA process A. The behaviour of A is simulated by a universal π-process *Ruler* which does not

depend on A. This process has two parameters s_{act} and s_{amb} as channel names for receiving submissions from nodes corresponding to actions and ambients. The received submissions indicate the readiness of the nodes to participate in the simulation of some MA operation (entering, exiting or opening).

Definition 10 *The formal description of a π-process Ruler is as follows.*

$$
\begin{array}{lll}
Ruler(s_{act}, s_{amb}) & = & (\nu x_0).\ (\nu x_1).\ (\nu x_2).\ (\nu y). \qquad (12)\\
& & s_{act}(v_0).\ s_{amb}(v_1).\ s_{amb}(v_2). \qquad (13)\\
& & \overline{v}_0\langle x_0\rangle.x_0(t_c, n_c, l_c, u_c, d_c, ul_c). \qquad (14)\\
& & \overline{v}_1\langle x_1\rangle.x_1(t_{a,1}, n_{a,1}, l_{a,1}, u_{a,1}, d_{a,1}, ul_{a,1}). \qquad (15)\\
& & \overline{v}_2\langle x_2\rangle.x_2(t_{a,2}, n_{a,2}, l_{a,2}, u_{a,2}, d_{a,2}, ul_{a,2}).\\
& & [t_c = in \wedge n_c = n_{a,2} \wedge ul_c = l_{a,1} \wedge ul_{a,1} = ul_{a,2}] \qquad (16)\\
& & (\\
& & \quad \overline{v}_0\langle l_c, u_c\rangle.\overline{v}_1\langle y, d_{a,2}\rangle.\overline{v}_2\langle y, u_{a,2}\rangle\ , \qquad (17)\\
& & \quad [t_c = out \wedge n_c = n_{a,2} \wedge ul_c = l_{a,1} \wedge ul_{a,1} = l_{a,2}] \qquad (18)\\
& & \quad (\\
& & \quad\quad \overline{v}_0\langle l_c, u_c\rangle.\overline{v}_1\langle y, u_{a,2}\rangle.\overline{v}_2\langle y, u_{a,2}\rangle\ , \qquad (19)\\
& & \quad\quad [t_c = open \wedge n_c = n_{a,1} \wedge ul_c = l_{a,2} \wedge ul_{a,1} = \qquad (20)\\
& & l_{a,2}]\\
& & \quad\quad (\\
& & \quad\quad\quad \overline{v}_0\langle l_c, u_c\rangle.\overline{v}_1\langle l_{a,1}, u_{a,1}\rangle.\overline{v}_2\langle y, u_{a,2}\rangle\ , \qquad (21)\\
& & \quad\quad\quad \overline{v}_0\langle y, u_c\rangle.\overline{v}_1\langle y, u_{a,1}\rangle.\overline{v}_2\langle y, u_{a,2}\rangle \qquad (22)\\
& & \quad\quad)\\
& & \quad)\\
& &).s_{act}(z_0).\ s_{amb}(z_1).\ s_{amb}(z_2).\ Ruler(s_{act}, s_{amb}) \qquad (23)
\end{array}
$$

The intended meaning of the lines in the definition of $Ruler(s_{act}, s_{amb})$ is explained below.

(12) The subprocess *Ruler* uses the following private names:

- x_0, x_1, x_2 for communication with the nodes selected for simulating MA operations, and
- y for instructing the selected nodes to remain active.

(13) *Ruler* selects non-deterministically three nodes representing an action and a pair of ambients in an MA-process that could participate in the simulation of MA operation. Selection is put into effect by receiving requests via public channels s_{act} and s_{amb} (see line (5)). The first selected node is an action-type node since the request from this node is received via the channel s_{act} which is shared by action-type-nodes only. Two others are ambient-type nodes. The requests include private channel names for communication with the selected nodes.

(14) Using this private channel, *Ruler* sends a fresh channel name x_0 to the action-type node. The node considers this message as an inquiry about its characteristics (type t_c, name n_c, label l_c, channel names u_c, d_c for communication with upper and lower contexts, label ul_c of the preceding node). It delivers the required names to *Ruler* via the private channel x_0 (see line (7)).

(15) As in the case of actions (see line (14)), *Ruler* asks the selected ambient-type nodes to provide the information on the names $n_{a,1}$ and $n_{a,2}$, labels $l_{a,1}$ and $l_{a,2}$, channel names for communication with lower contexts $d_{a,1}$ and $d_{a,2}$, and labels of the preceding nodes $ul_{a,1}$ and $ul_{a,2}$ of these nodes.

(16) ¿From this point *Ruler* begins to check which operation on MA can be executed by means of the capability represented by the selected action-type node on the ambients represented by the selected ambient-type nodes. There are three conditions to ensure that *Ruler* simulates the application of entering reduction step to the MA process. First, the selected action-type node labelled with l_c should represent an action which has the capability for entering ($t_c = in$) into the ambient named $n_{a,2}$ ($n_c = n_{a,2}$); secondly, it lies on the top level in the ambient named $n_{a,1}$ ($ul_c = l_{a,1}$); and finally, the ambients named $n_{a,1}$ and $n_{a,2}$ are siblings ($ul_{a,1} = ul_{a,2}$).

(17) The entering reduction step is simulated by changing the communication net in the π-process $Structure_A$ which represents the spatial structure of MA-process A. Since the capability represented by the selected action-type node is consumed, *Ruler* sends to this node its label l_c via the private channel. After receiving this message the node evolves into a wire (see lines (7),(8),(9),(1),(2)). Since the ambient named $n_{a,1}$ enters the sibling ambient named $n_{a,2}$, the corresponding ambient-type node has to change the upper context. *Ruler* sends to this node the channel name $d_{a,2}$ for communication with its new upper context which is the node corresponding to the ambient named $n_{a,2}$. The upper context of the node corresponding to the ambient named $n_{a,2}$ remains the same, and *Ruler* acknowledges this by communicating back the value $u_{a,2}$. A private name y which does not match the labels $l_{a,1}$ and $l_{a,2}$ is sent to the ambient-type nodes as an instruction to remain active (see lines (7),(8),(10),(11),(1),(3)).

(18) If the selected nodes do not match an entry-pattern, then *Ruler* checks for an exit-pattern. There are three conditions to ensure that *Ruler* simulates the application of exiting reduction step to the MA process. First, the selected action-type node labelled with l_c should represent an action which has the capability for exiting ($t_c = out$) out of the ambient named $n_{a,2}$ ($n_c = n_{a,2}$); secondly, it lies on the top level in the ambient named $n_{a,1}$ ($ul_c = l_{a,1}$); and finally, the ambient named $n_{a,1}$ lies on the top level of the ambient named $n_{a,2}$ ($ul_{a,1} = l_{a,2}$).

(19) Since the capability represented by the selected action-type node is consumed, *Ruler* sends to this node its label l_c to evolve the node into a wire (see lines (7),(8),(9), (1),(2)). Since the ambient named $n_{a,1}$ is transformed into a sibling of the ambient named $n_{a,2}$, it changes the upper context from $u_{a,1}$ to $u_{a,2}$. The upper context of the node corresponding to the ambient named $n_{a,2}$ remains the same. *Ruler* sends a private name y which does not match the labels $l_{a,1}$ and $l_{a,2}$ to instruct the ambient-type nodes to remain active (see lines (7),(8),(10),(11),(1),(3)).

(20) If the selected nodes do not match exit-pattern, then *Ruler* checks for an open-pattern. If the selected action-type node labelled with l_c represents an action which has the capability for dissolving the boundary ($t_c = out$) of the ambient named $n_{a,1}$ ($n_c = n_{a,1}$), lies on the top level in the ambient named $n_{a,2}$ ($ul_c = l_{a,2}$), and the ambient named $n_{a,1}$ also lies on the top level of the ambient named $n_{a,2}$ ($ul_{a,1} = l_{a,2}$), then *Ruler* simulates the application of an opening reduction step to the MA process.

(21) To simulate an opening MA-operation, *Ruler* sends l_c to the action-type node and $l_{a,1}$ to the ambient-type node named $n_{a,1}$ to evolve these nodes into wires (see lines (7),(8),(9),(1),(2)) since the capability is consumed and the boundary of the ambient is dissolved. *Ruler* sends a private name y which does not match the label $l_{a,2}$ to instruct the ambient-type node named $n_{a,2}$ to remain active (see lines (7),(8),(10),(11),(1),(3)).

(22) If the selected nodes do not match any pattern, then *Ruler* informs them via private channels to remain active and to keep their channels for communication with upper contexts unchanged.

(23) After this *Ruler* waits till the nodes participated in this round of simulation send their synchronization messages (the labels) to indicate that instructions sent to them (see lines (17), (19), (21), or (22)) are performed (see lines (2) and (11)), reverts to the initial state and tries another triple of nodes.

4.4 Environment

The environment is considered as a top-most ambient encompassing an MA-process A. But unlike conventional ambients, it can not be dissolved and no ambient can exit out of it. The environment plays the role of an upper context for unguarded actions and ambients; it can also participate in the simulation of open operations as Amb_2. Moreover, for the sake of uniformity it is convenient to compose an environment out of two ambient-type processes. Only one of these processes can actually participate in simulation of some MA-operation. The other is just a dummy which gives *Ruler* a possibility to operate till at least one active action-type node remains in $Structure_A$.

Definition 11 *The formal description of a π-process Environment is as follows.*

$$Environment(env, \top, d, s_{amb}) = Top(env, \top, d, s_{amb}) \mid (\nu d').Top(env, \top, d', s_{amb}) \quad (24)$$

$$\begin{aligned} Top(env, \top, d, s) = {} & (\nu v).\ (\nu l).\ (\nu u).\ (\nu w). && (25)\\ & \overline{s}\langle v\rangle.\ v(x). && (26)\\ & \overline{x}\langle env, \top, l, u, d, w\rangle. && (27)\\ & v(y, z).\ \overline{s}\langle l\rangle.\ Top(env, \top, d, s) && (28) \end{aligned}$$

We comment briefly on the intended meaning of the above processes.

(24) The environment is composed of two processes Top. They have the same functionality, but only the first one has a global name for communication with its lower context (top-level actions and ambients encompassed by the environment). Nevertheless both processes can communicate with *Ruler* via the channel s_{amb} shared by ambient-type nodes. The name $\top$ is an arbitrary name which is different from any free name in an ambient encompassed by the environment.

(25) A process Top uses the following private names:

- v as a name of a channel for receiving acknowledgements and instructions from *Ruler*,

– l, u, w as dummy names that are used only for the sake of uniformity in communications with *Ruler* (they stand for the label, upper context and label of the predecessor of the node that are of no importance for the environment).

(26) The environment processes begin with sending to *Ruler* their requests for participation in the simulation of MA operations. When participation is granted *Top* receives a private channel name for communication with *Ruler* (see line (15)).

(27) Using this channel *Top* sends to *Ruler* the information about its type (a keyword *env*), name (it should be different from any name in $Structure_A$), channel names for communication with its upper context (since it does not exist any private name is possible) and lower context (d), the label of the preceding node (it does not exist also and *Top* uses any private name for this purpose).

(28) As any other node participating in the simulation of MA operation as seen in line (7), *Top* receives a pair of names (y and z) which are interpreted as instructions for changing its status and updating its context. But since the environment can not be dissolved and it has no upper context, these names do not affect on its functionality. This input is used only for the sake of uniformity which gives *Ruler* a possibility not to distinguish *Top* as a specific node. Therefore, *Top* just acknowledges the receipt of these names by sending a synchronization message to *Ruler* and reverts to the original state.

5 Embedding Ambients into π-processes

The embedding of pure ambients into π-processes is defined in terms of two relations $\models_0$ and $\models$. We use $\models_0$ for constructing $Structure_A$ corresponding to MA process A and $\models$ for constructing the ultimate π-process out of $Structure_A$, *Ruler* and *Environment*.

Definition 12 *The relation $\models_0$ is defined inductively by the following axioms and rules. In every pair $[P, k]$ to the right of $\models_0$ the second component k stands for the free channel name used in the π-process P for communication with its upper context.*

Axioms:

Ax1 (Simple Inactivity) $\mathbf{0} \models_0 [\mathbf{0}, k]$, where $k \in \mathcal{N}$;

Ax2 (Tree-wire) $\mathbf{0} \models_0 [(\nu I).TW_{I,k}, k]$, where $I \subset \mathcal{N},\ k \in \mathcal{N}$;

Rules:

R1 (Add tree-wire) $\dfrac{A \models_0 [P, k]}{A \models_0 [(\nu I).(W_{I,m}|P), m]}$,

where $k \in I, fn(P) \cap I \subseteq \{k\},\ m \notin fn(P) \cup I$;

R2 (Composition) $\dfrac{A_1 \models_0 [P_1, k]\ ,\ A_2 \models_0 [P_2, k]}{A_1|A_2 \models_0 [P_1|P_2, k]}$;

R3 (Restriction) $\dfrac{A \models_0 [P, k]}{(\nu n)A \models_0 [(\nu n).P, k]}$; R4 (Replication) $\dfrac{A \models_0 [P, k]}{!A \models_0 [!P, k]}$;

R5 (Action) $\dfrac{A \models_0 [P, k]}{cap\ n.\ A \models_0 [(\nu k).\ (\nu l).(Node(cap, n, m, k, s_{act}, l)|P), m]}$,

where $cap \in \{in, out, open\}$, $l, m \notin fn(P) \cup \{n\}$;

R6 (Ambient) $\dfrac{A \models_0 [P, k]}{n[A] \models_0 [(\nu k).(\nu l).(Node(amb, n, m, k, s_{amb}, l)|P), m]}$,

where $l, m \notin fn(P) \cup \{n\}$.

The embedding relation $\models$ is defined by the single rule

R0 (MA-to-π)

$$\frac{A \models_0 [Structure_A, k]}{A \models (\nu\Sigma).(Structure_A|Ruler(s_{act}, s_{amb})|Environment(env, \top, k, s_{amb}))}$$

where $\nu\Sigma$ stands for the prefix

$$(\nu in).\ (\nu out).\ (\nu open).\ (\nu amb).\ (\nu env).\ (\nu s_{act}).\ (\nu s_{amb}).\ (\nu \top).\ (\nu k).\ ,$$

and $\top$ is any name from $\mathcal{N} - fn(A)$.

Proposition 4 *Let A, B be two MA-processes such that $A \equiv B$, and $A \models P$. Then there exists a derivation $B \models Q$ such that $P \equiv Q$.*

The proof is given by showing that each structural congruence in Definition 2 holds for our translation $\models_0$.

Proposition 5 *If $A \models P$, then $fn(A) = fn(P)$.*

6 Completeness and Correctness

In this section we will demonstrate that the embedding of pure ambients into π-calculus is complete and correct. By completeness we mean that any π-process P associated with an MA-process A through the embedding relation $A \models P$ admits only those π-calculus reductions $P \rightarrow_\pi^* P'$ that can be interpreted in terms of pure ambient reductions $A \rightarrow_a A'$ such that $A' \models P'$. Correctness means that any reduction $A \rightarrow_a A'$ corresponds to some chain of π-calculus reductions $P \rightarrow_\pi^* P'$ of P such that $A' \models P'$. Thus, we may speak of a homomorphic embedding of pure mobile ambients into π-calculus.

Theorem 1 (Completeness) *Let A_0, A_1 be MA-processes and P_0 be a π-process such that $A_0 \rightarrow_a A_1$ and $A_0 \models P_0$. Then there exists a chain of π-calculus reduction steps*

$$P_0 \ \hookrightarrow_\pi^3 \ P_1' \ \mapsto_\pi^* \ P_1$$

such that $A_1 \models P_1$.

The proof follows straightforward from the description of processes $Main$, *Reply*, *Ruler*, *Top* and Proposition 3. The only nondeterministic steps in the reduction $P_0 \rightarrow_\pi^* P_1$ are three communications steps when *Ruler* selects nodes representing a capability and a pair of ambients for simulating a reduction step $A_0 \rightarrow_a A_1$. Afterwards the reduction of P_0 is completely deterministic until all communication actions in the the bodies of subprocesses *Ruler*, $Main$ and *Reply* are executed to an end.

Theorem 2 (Correctness) *Let A_0 be an MA-process and P_0 be a π-process such that $A_0 \models P_0$. Let $P_0 \rightarrow_\pi^* P$ be a chain of π-calculus reduction steps. Then there exist an integer N, $0 \leq N \leq 2$, a sequence of π-calculus terms $P_1', P_1, P_2', P_2, \ldots, P_n', P_n$ and a sequence of pure ambient terms $A_1, A_2, \ldots, A_n$ such that the following conditions hold*

1. $P \hookrightarrow_\pi^N P_n' \mapsto_\pi^* P_n$;
2. *The chain of π-calculus reductions $P_0 \rightarrow_\pi^* P \rightarrow_\pi^* P_n$ can be refined as*

$$P_0 \hookrightarrow_\pi^3 P_1' \mapsto_\pi^* P_1 \hookrightarrow_\pi^3 P_2' \mapsto_\pi^* P_2 \hookrightarrow_\pi^3 \cdots P_{n-1} \hookrightarrow_\pi^{3-N} P \hookrightarrow_\pi^N P_n' \mapsto_\pi^* P_n$$

such that

   (a) *$A_i \models P_i$ for every i, $0 \leq i \leq n$;*
   (b) *for every i, $0 \leq i < n$, either $A_i \equiv A_{i+1}$ or $A_i \rightarrow_a A_{i+1}$.*

This means that if a π-process P_0 encodes an MA-process A_0, then every chain of reductions $P_0 \rightarrow_\pi^* P$ can be considered as a prefix of a π-calculus simulation $P_0 \rightarrow_\pi^* P_n$ of an MA computation $A_0 \rightarrow_a^* A_n$.

The proof of this theorem is by induction on the number of nondeterministic steps $\hookrightarrow_\pi$ in a reduction of P_0. Each triple of nondeterministic steps in such reduction is followed by a chain of deterministic reduction steps that either simulate the execution of some MA-reduction step if the selected nodes in a π-process P_i comply to one of MA reduction rules, or restore P_i otherwise.

We may note that our translation has diverging reductions whenever the selected nodes do not conform any MA reduction rule. In this case we may obtain an infinite chain $P \hookrightarrow_\pi^3 P' \mapsto_\pi^* P \hookrightarrow_\pi^3 P' \mapsto_\pi^* P \hookrightarrow_\pi^3 \ldots$

Theorem 3 (Complexity) *Let A_0, A_1 be MA-processes such that $A_0 \rightarrow_a^N A_1$ (i.e. A_0 can be reduced to A_1 in N steps). Let P_0 be a π-process such that $A_0 \models P_0$ and neither the axiom Ax2 (Tree-wire), nor the rule R1 (Add tree-wire) are used in the derivation of P_0 (i.e. P_0 is the minimal counterpart of A_0). Then there exists a chain of π-calculus reductions $P_0 \rightarrow_\pi^L P_1$ such that $A_1 \models P_1$ and $L = O(N^2)$.*

Proposition 6 *Let A be an MA-process and P be a π-process such that $A \models P$. Then P is a localized π-calculus term.*

7 Conclusions

Both ambient calculus and π-calculus are very expressive and allow the natural embedding into them of many other universal models of computations such as Turing machine and λ-calculus. In [2] and [3], the π-calculus is encoded into the full mobile ambient calculus; [10] has proposed an embedding of the π-calculus only into the pure ambients calculus.

As far as we know, [6] is the first paper describing an embedding of the mobile ambients into the π-calculus. In this paper we extend [6], and present an embedding of the pure mobile ambient calculus in a subset of π-calculus, namely the localized sum-free synchronous monadic π-calculus. We prove the completeness and correctness of our embedding by showing that each reduction step of an MA process can be uniformly simulated by chains of π-calculus reductions of corresponding the π-terms. Two consequences of the embedding are worth to be emphasized. Our embedding gives a possibility to analyze some properties of mobile ambients by means of static analysis and congruence-checking machinery developed for the π-calculus [1]. The fact that we restrict ourselves with localized and plus-free π-terms alleviates substantially the analysis. On the other hand, our embedding does not involve any sophisticated structures that can spoil the precision of such analysis. The embedding is close to an interpreter, and it can be used for implementing the ambient calculus (in Pict, for instance).

The embedding can be extended to a full ambient calculus, adding a communication channel per ambient. This implies a "merging" of channels when an ambient is opened; we may use the same mechanism: the *Ruler* randomly selects an input and an output, checks if they belong to the same ambient, and performs communication.

The proof that such a translation preserves the meaning is usually given by showing that some congruence in one model is mapped into a congruence in the other model. We do not relate the embedding to any observation on the the ambient processes and their π-calculus translations. Here we emphasize on the technique of embedding, and proving some related results. This translation opens some interesting questions; for instance whether the barbed congruence for ambient calculus is related to a barbed congruence for the π-calculus. It may also open a way of defining a spectrum of congruences in mobile ambients, relating them with the corresponding congruences in the π-calculus (π-calculus has many congruences, but pure mobile ambient calculus still have only one reasonable congruence, namely the contextual equivalence [2]).

References

[1] C. Bodei, P. Degano, F. Nielson, H.R. Nielson. Control flow analysis for the π-calculus. *Proc. CONCUR'98*, LNCS 1466, p.84-98, Springer, 1998.

[2] L. Cardelli, A. Gordon. Mobile Ambients, *Proc. Foundations of Software Science and Computation Structures*, Springer, 1998.

[3] W. Charatonik, A. Gordon, J.-M. Talbot. Finite-control mobile ambients. *Proc. European Symposium on Programming*, LNCS 2305, p.295-313, Springer, 2002.

[4] G. Ciobanu, M. Rotaru. A π-calculus machine. *J. of Universal Computer Science* vol.6(1), Springer, 2000.

[5] G. Ciobanu, M. Rotaru. Molecular Interaction. *Theoretical Computer Science* vol.289(1), 801-827, Elsevier, 2002.

[6] G. Ciobanu, V. Zakharov. Mobile ambients, pi-calculus and their relationship. *Proceedings 5th Congress of Romanian Mathematicians*, 30-32, 2003.

[7] F. Levi, S. Maffeis. An abstract interpretation framework for mobile ambients. *Proc. SAS'01*, LNCS 2126, p.395-411, Springer, 2001.

[8] R. Milner. *Communicating and mobile systems: the π-calculus*, Cambridge University Press, 1999.

[9] D. Sangiorgi, D. Walker, *The π-calculus: a theory of mobile processes*, Cambridge University Press, 2001.

[10] P. Zimmer, On the expressiveness of pure safe ambients, Report INRIA No.4350, 53p., 2002.