Encoding Named Channels Communication by Behavioral Schemes

Martin Tomášek

Department of Computers and Informatics, Faculty of Electrical Engineering and Informatics, Technical University of Košice
Letná 9, 042 00 Košice, Slovakia
e-mail: martin.tomasek@tuke.sk

Abstract: Our new approach to the calculus of mobile ambients is suitable for expressing the dynamic properties of mobile code applications, where the main goal is to avoid the ambiguities and possible maliciousness of some constructions. We define a behavioral scheme assigned to process types that statically specifies and checks access rights for authorization of ambients and threads to communicate and move. As an expressiveness test, we showed that the well-known π-calculus of concurrency and mobility can be encoded in our calculus in a natural way.

Keywords: process calculi; mobile code; type system

1 Introduction

The calculus of mobile ambients [1] is based on a concurrency paradigm represented by the π-calculus [2]. It introduces the notion of an ambient as a bounded place where concurrent computation takes place, which can contain nested subambients in a hierarchical structure, and which can move in and out of other ambients, i.e., up and down the hierarchy that rearranges the structure of ambients. Communication can only occur locally within each ambient through a common anonymous channel. Communication between different ambients has to be performed by movement and by dissolution of ambient boundaries.

Mobile ambients model several computational entities: mobile agents, mobile processes, messages, packets or frames, physical or virtual locations, administrative and security domains in a distributed system and also mobile devices. This variety means in principle there are no differences among various kinds of software components when expressing by mobile ambients. In mobile ambients there are implicitly two main forms of entities which we will respectively call *threads* and *ambients*. Threads are unnamed sequences of

primitive actions to be executed sequentially, generally in concurrence with other threads. They can perform communication and drive their containers through the spatial hierarchy but cannot individually go from one ambient to another. Ambients are named containers of concurrent threads. They can enter and exit other ambients, driven by their internal processes, but cannot directly perform communication. It is very important to ensure indivisibility and the autonomous behavior of ambients (this is also important e.g. for objects).

Communication between ambients is represented by the movement of other ambient of usually shorter life, which have their boundaries dissolved by an *open* action to expose their internal threads performing local communication operations. Such capability of opening an ambient is potentially dangerous [3, 4, 5]. It could be used inadvertently to open and thus destroy the individuality of an object or mobile agent. Remote communication is usually emulated as a movement of such ambients (communication packages) in the hierarchy structure.

Table 1
Abstract syntax

$M ::=$	**mobility operations**
$\mid\ n$	name
$\mid\ in\ M$	move ambient into M
$\mid\ out\ M$	move ambient out of M
$\mid\ move\ M$	move thread into M
$\mid\ M.M'$	path
$P ::=$	**processes**
$\mid\ \mathbf{0}$	inactive process
$\mid\ P \mid P'$	parallel composition
$\mid\ !P$	replication
$\mid\ M[P]$	ambient
$\mid\ (\nu n : \mathbf{P}[\mathcal{B}])P$	name restriction
$\mid\ M.P$	action of the operation
$\mid\ \langle M \rangle.P$	synchronous output
$\mid\ (n : \mu).P$	synchronous input

We explore a different approach, where we intend to keep the purely local character of communication so that no hidden costs are present in the communication primitives, but without *open* operation. This solves the problem of dissolving boundaries of ambients but disables interactions of threads from separate ambients. We must introduce a new operation, *move,* for moving threads

between ambients. The idea comes from mobile code programming paradigms [6], where moving threads can express a strong mobility mechanism, by which the procedure can (through *move* operation) suspend its execution on one machine and resume it exactly from the same point on another (remote) machine. This solves the problem of threads mobility and by moving threads between ambients we can emulate communication between the ambients.

2 Overview of the Calculus

1.1 Syntax

The abstract syntax of the terms of our calculus in Table 1 is the same as that of mobile ambients, except for the absence of *open* and the presence of the new operation *move* for moving threads between ambients.

Table 2
Free (a) and bound (b) names

(a)	(b)
$fn(n) = \{n\}$	$bn(n) = \varnothing$
$fn(in\ M) = fn(M)$	$bn(in\ M) = bn(M)$
$fn(out\ M) = fn(M)$	$bn(out\ M) = bn(M)$
$fn(move\ M) = fn(M)$	$bn(move\ M) = bn(M)$
$fn(M.M') = fn(M) \cup fn(M')$	$bn(M.M') = bn(M) \cup bn(M')$
$fn(\mathbf{0}) = \varnothing$	$bn(\mathbf{0}) = \varnothing$
$fn(P \mid P') = fn(P) \cup fn(P')$	$bn(P \mid P') = bn(P) \cup bn(P')$
$fn(!P) = fn(P)$	$bn(!P) = bn(P)$
$fn(M[P]) = fn(M) \cup fn(P)$	$bn(M[P]) = bn(M) \cup bn(P)$
$fn((\nu n : \mathbf{P}[\mathcal{B}])P) = fn(P) - \{n\}$	$bn((\nu n : \mathbf{P}[\mathcal{B}])P) = bn(P) \cup \{n\}$
$fn(M.P) = fn(M) \cup fn(P)$	$bn(M.P) = bn(M) \cup bn(P)$
$fn(\langle M \rangle.P) = fn(M) \cup fn(P)$	$bn(\langle M \rangle.P) = bn(M) \cup bn(P)$
$fn((n : \mu).P) = fn(P) - \{n\}$	$bn((n : \mu).P) = bn(P) \cup \{n\}$

1.2 Operational Semantics

The operational semantics is given by reduction relation along with a structural congruence, in the same way as those for mobile ambients.

Table 3
Structural congruence

equivalence:	
$P \equiv P$	(SRefl)
$P \equiv Q \Rightarrow Q \equiv P$	(SSymm)
$P \equiv Q, Q \equiv R \Rightarrow P \equiv R$	(STrans)
congruence:	
$P \equiv Q \Rightarrow P \mid R \equiv Q \mid R$	(SPar)
$P \equiv Q \Rightarrow !P \equiv !Q$	(SRepl)
$P \equiv Q \Rightarrow M[P] \equiv M[Q]$	(SAmb)
$P \equiv Q \Rightarrow (\nu n : \mathbf{P}[\mathcal{B}])P \equiv (\nu n : \mathbf{P}[\mathcal{B}])Q$	(SRes)
$P \equiv Q \Rightarrow M.P \equiv M.Q$	(SAct)
$P \equiv Q \Rightarrow \langle M \rangle.P \equiv \langle M \rangle.Q$	(SCommOut)
$P \equiv Q \Rightarrow (n : \mu).P \equiv (n : \mu).Q$	(SCommIn)
sequential composition (associativity):	
$(M.M').P \equiv M.M'.P$	(SPath)
parallel composition:	
$P \mid Q \equiv Q \mid P$	(SParComm)
$(P \mid Q) \mid R \equiv P \mid (Q \mid R)$	(SParAssoc)
$P \mid \mathbf{0} \equiv P$	(SParNull)
replication:	
$!P \equiv P \mid !P$	(SReplPar)
$!\mathbf{0} \equiv \mathbf{0}$	(SReplNull)
restriction and scope extrusion:	
$n \neq m \Rightarrow (\nu n : \mathbf{P}[\mathcal{B}])(\nu m : \mathbf{P}[\mathcal{B}'])P \equiv (\nu m : \mathbf{P}[\mathcal{B}'])(\nu n : \mathbf{P}[\mathcal{B}])P$	(SResRes)
$n \notin fn(Q) \Rightarrow (\nu n : \mathbf{P}[\mathcal{B}])P \mid Q \equiv (\nu n : \mathbf{P}[\mathcal{B}])(P \mid Q)$	(SResPar)
$n \neq m \Rightarrow (\nu n : \mathbf{P}[\mathcal{B}])m[P] \equiv m[(\nu n : \mathbf{P}[\mathcal{B}])P]$	(SResAmb)
$(\nu n : \mathbf{P}[\mathcal{B}])\mathbf{0} \equiv \mathbf{0}$	(SResNull)
garbage collection:	
$(\nu n : \mathbf{P}[\mathcal{B}])n[\mathbf{0}] \equiv \mathbf{0}$	(SAmbNull)

Each name of the process term can figure either as free (Table 2a) or bound (Table 2b).

We write $P\{n \leftarrow M\}$ for a substitution of the capability M for each free occurrences of the name n in the term P. The similarly for $M\{n \leftarrow M\}$.

Structural congruence is shown in Table 3 and it is standard for mobile ambients.

In addition, we identify processes up to renaming of bound names (α-conversion) as shown in Table 4.

Table 4
α-conversion

$(\nu n : \mathbf{P}[\mathcal{B}])P = (\nu m : \mathbf{P}[\mathcal{B}])P\{n \leftarrow m\} \quad m \notin fn(P)$	(SAlphaRes)
$(n : \mu)P = (m : \mu)P\{n \leftarrow m\} \quad m \notin fn(P)$	(SAlphaCommIn)

The reduction rules in Table 5 are those for mobile ambients, with the obvious difference consisting in the synchronous output and the missing *open* operation, and with the new rule for the *move* operation similar to the "migrate" instructions for strong code mobility in software agents.

Table 5
Reduction rules

basic reductions:	
$n[in\ m.P \mid Q] \mid m[R] \rightarrow m[n[P \mid Q] \mid R]$	(RIn)
$m[n[out\ m.P \mid Q] \mid R] \rightarrow n[P \mid Q] \mid m[R]$	(ROut)
$n[move\ m.P \mid Q] \mid m[R] \rightarrow n[Q] \mid m[P \mid R]$	(RMove)
$(n : \mu).P \mid \langle M \rangle.Q \rightarrow P\{n \leftarrow M\} \mid Q$	(RComm)
structural reductions:	
$P \rightarrow Q \Rightarrow P \mid R \rightarrow Q \mid R$	(RPar)
$P \rightarrow Q \Rightarrow n[P] \rightarrow n[Q]$	(RAmb)
$P \rightarrow Q \Rightarrow (\nu n : \mathbf{P}[\mathcal{B}])P \rightarrow (\nu n : \mathbf{P}[\mathcal{B}])Q$	(RRes)
$P' \equiv P, P \rightarrow Q, Q \equiv Q' \Rightarrow P' \rightarrow Q'$	(RStruct)

3 Overview of the Type System

The restriction of the mobility operations is defined by types applying a *behavioral scheme*. The scheme allows setting up the access rights for traveling of threads and ambients in the ambient hierarchy space of the system.

3.1 Types and Behavioral Scheme

Types are defined in Table 6 where we present communication types and message types.

Table 6
Types with behavioral schemes

$\kappa ::=$	**communication type**
$\mid \perp$	no communication
$\mid \mu$	communication of messages of type μ
$\mu ::=$	**message type**
$\mid$ **P**$[\mathcal{B}]$	process with behavioral scheme $\mathcal{B}$
$\mid$ **O**$[\mathcal{B} \mapsto \mathcal{B}']$	operation which changes behavioral scheme $\mathcal{B}$ to $\mathcal{B}'$

The behavioral scheme is the structure $\mathcal{B}=(\kappa, Reside, Pass, Move)$ which contains four components:

- κ is the communication type of the ambient's threads
- *Reside* is the set of behavioral schemes of other ambients where the ambient can stay
- *Pass* is the set of behavioral schemes of other ambients that ambient can go through, it must be $Pass \subseteq Reside$
- *Move* is the set of behavioral schemes of other ambients where ambient can move its containing thread

3.2 Typing Rules

The type environment is defined as a set $\Gamma = \{n_1 : \mu_1, \ldots, n_l : \mu_l\}$ where each $n_i : \mu_i$ assigns a unique type μ_i to a name n_i.

The domain of the type environment is defined by:

1 $Dom(\varnothing) = \varnothing$

2 $Dom(\Gamma, n : \mu) = Dom(\Gamma) \cup \{n\}$

We define two type formulas for our ambient calculus:

1 $\Gamma \vdash M : \mu$

2 $\Gamma \vdash P : \mathbf{P}[\mathcal{B}]$

Typing rules are shown in Table 7 and they are used to derive type formulas of ambient processes. We say the process is *well-typed* when we are able to derive a type formula for it using our typing rules. Well-typed processes respect the communication and mobility restrictions defined in all behavioral schemes of the system.

Table 7
Typing rules

$\dfrac{n : \mu \in \Gamma}{\Gamma \vdash n : \mu}$	(TName)
$\dfrac{\Gamma \vdash M : \mathbf{P}[\mathcal{B}] \quad \mathcal{B} \in Pass(\mathcal{B}')}{\Gamma \vdash in\ M : \mathbf{O}[\mathcal{B}' \mapsto \mathcal{B}']}$	(TIn)
$\dfrac{\Gamma \vdash M : \mathbf{P}[\mathcal{B}] \quad \mathcal{B} \in Pass(\mathcal{B}') \quad Reside(\mathcal{B}) \subseteq Reside(\mathcal{B}')}{\Gamma \vdash out\ M : \mathbf{O}[\mathcal{B}' \mapsto \mathcal{B}']}$	(TOut)
$\dfrac{\Gamma \vdash M : \mathbf{P}[\mathcal{B}] \quad \mathcal{B} \in Move(\mathcal{B}')}{\Gamma \vdash move\ M : \mathbf{O}[\mathcal{B} \mapsto \mathcal{B}']}$	(TMove)
$\dfrac{\Gamma \vdash M : \mathbf{O}[\mathcal{B}'' \mapsto \mathcal{B}'] \quad \Gamma \vdash M' : \mathbf{O}[\mathcal{B} \mapsto \mathcal{B}'']}{\Gamma \vdash M.M' : \mathbf{O}[\mathcal{B} \mapsto \mathcal{B}']}$	(TPath)
$\dfrac{}{\Gamma \vdash \mathbf{0} : \mathbf{P}[\mathcal{B}]}$	(TNull)
$\dfrac{\Gamma \vdash P : \mathbf{P}[\mathcal{B}] \quad \Gamma \vdash P' : \mathbf{P}[\mathcal{B}]}{\Gamma \vdash P \mid P' : \mathbf{P}[\mathcal{B}]}$	(TPar)
$\dfrac{\Gamma \vdash P : \mathbf{P}[\mathcal{B}]}{\Gamma \vdash !P : \mathbf{P}[\mathcal{B}]}$	(TRepl)
$\dfrac{\Gamma \vdash P : \mathbf{P}[\mathcal{B}] \quad \Gamma \vdash M : \mathbf{P}[\mathcal{B}] \quad \mathcal{B}' \in Reside(\mathcal{B})}{\Gamma \vdash M[P] : \mathbf{P}[\mathcal{B}']}$	(TAmb)
$\dfrac{\Gamma, n : \mathbf{P}[\mathcal{B}'] \vdash P : \mathbf{P}[\mathcal{B}]}{\Gamma \vdash (\nu n : \mathbf{P}[\mathcal{B}'])P : \mathbf{P}[\mathcal{B}]}$	(TRes)
$\dfrac{\Gamma \vdash M : \mathbf{O}[\mathcal{B} \mapsto \mathcal{B}'] \quad \Gamma \vdash P : \mathbf{P}[\mathcal{B}]}{\Gamma \vdash M.P : \mathbf{P}[\mathcal{B}']}$	(TAct)
$\dfrac{\Gamma \vdash P : \mathbf{P}[\mathcal{B}] \quad \Gamma \vdash M : \mu \quad \kappa(\mathcal{B}) = \mu}{\Gamma \vdash \langle M \rangle.P : \mathbf{P}[\mathcal{B}]}$	(TCommOut)
$\dfrac{\Gamma, n : \mu \vdash P : \mathbf{P}[\mathcal{B}] \quad \kappa(\mathcal{B}) = \mu}{\Gamma \vdash (n : \mu).P : \mathbf{P}[\mathcal{B}]}$	(TCommIn)

4 Encoding Named Channels

A standard expressiveness test for the ambient calculus and our variant is the encoding of communication on named channels via local anonymous communication within ambients. We consider a core fragment of the typed monadic synchronous π-calculus, given by the following grammar:

$$P ::= x(y:\sigma).P \mid \overline{x}\langle z\rangle.P \mid \mathsf{new}\, a:\sigma\, P \mid P_1 \mid P_2 \mid !P$$

where P, P_1,P_2 denotes processes and x, y, z, a are named channels from the set of all names $\mathcal{N}$ and lets k be their number.

Table 8
Free and bound names in π-calculus

$fn_\pi(x(y:\sigma).P) = \{x\} \cup fn_\pi(P)$	$bn_\pi(x(y:\sigma).P) = \{y\} \cup bn_\pi(P)$
$fn_\pi(\overline{x}\langle z\rangle.P) = \{x,z\} \cup fn_\pi(P)$	$bn_\pi(\overline{x}\langle z\rangle.P) = bn_\pi(P)$
$fn_\pi(\mathsf{new}\, a:\sigma\, P) = fn_\pi(P) - \{a\}$	$bn_\pi(\mathsf{new}\, a:\sigma\, P) = bn_\pi(P) \cup \{a\}$
$fn_\pi(P_1 \mid P_2) = fn_\pi(P_1) \cup fn_\pi(P_2)$	$bn_\pi(P_1 \mid P_2) = bn_\pi(P_1) \cup bn_\pi(P_2)$
$fn_\pi(!P) = fn_\pi(P)$	$bn_\pi(!P) = bn_\pi(P)$

Table 9
Structural congruence in π-calculus

$P \equiv_\pi P$	(πSRefl)
$P \equiv_\pi Q \Rightarrow Q \equiv_\pi P$	(πSSymm)
$P \equiv_\pi Q, Q \equiv_\pi R \Rightarrow P \equiv_\pi R$	(πSTrans)
$P \equiv_\pi Q \Rightarrow x(z:\sigma).P \equiv_\pi x(z:\sigma).Q$	(πSCommIn)
$P \equiv_\pi Q \Rightarrow \overline{x}\langle z\rangle.P \equiv_\pi \overline{x}\langle z\rangle.Q$	(πSCommOut)
$P \equiv_\pi Q \Rightarrow \mathsf{new}\, a:\sigma\, P \equiv_\pi \mathsf{new}\, a:\sigma\, Q$	(πSRes)
$P \equiv_\pi Q \Rightarrow P \mid R \equiv_\pi Q \mid R$	(πSPar)
$P \equiv_\pi Q \Rightarrow !P \equiv_\pi !Q$	(πSRepl)
$P \mid Q \equiv_\pi Q \mid P$	(πSParComm)
$(P \mid Q) \mid R \equiv_\pi P \mid (Q \mid R)$	(πSParAssoc)
$!P \equiv_\pi P \mid !P$	(πSReplPar)
$a \neq b \Rightarrow \mathsf{new}\, a:\sigma\, \mathsf{new}\, b:\sigma'\, P \equiv_\pi \mathsf{new}\, b:\sigma'\, \mathsf{new}\, a:\sigma\, P$	(πSResRes)
$a \notin fn_\pi(P) \Rightarrow \mathsf{new}\, a:\sigma\, (P \mid Q) \equiv_\pi \mathsf{new}\, a:\sigma\, P \mid Q$	(πSResPar)
$a \notin fn_\pi(P) \Rightarrow \mathsf{new}\, a:\sigma\, P \equiv_\pi P$	(πSResSkip)

The set of free and bound names of π-calculus terms are in Table 8.

Table 10
Free and bound names in π-calculus

$P \to_\pi Q \Rightarrow \mathsf{new}\ a:\sigma\ P \to_\pi \mathsf{new}\ a:\sigma\ Q$	(πRRes)
$P \to_\pi Q \Rightarrow P \mid R \to_\pi Q \mid R$	(πRPar)
$x(y:\sigma).P \mid \overline{x}\langle z\rangle.Q \to_\pi P\{y \leftarrow z\} \mid Q$	(πRComm)
$P' \equiv_\pi P, P \to_\pi Q, Q \equiv_\pi Q' \Rightarrow P' \to_\pi Q'$	(πRStruct)

Structural operational semantics of π-calculus is given by the structural congruence $\equiv_\pi$ (Table 9) and reduction relation $\to_\pi$ (Table 10).

Table 11
Typing rules in π-calculus

$\dfrac{a:\sigma \in \Gamma}{\Gamma \vdash a:\sigma}$	(πTName)
$\dfrac{\Gamma \vdash x:\mathrm{CH}(\sigma) \quad \Gamma, y:\sigma \vdash P}{\Gamma \vdash x(y:\sigma).P}$	(πTCommIn)
$\dfrac{\Gamma \vdash x:\mathrm{CH}(\sigma) \quad \Gamma \vdash z:\sigma \quad \Gamma \vdash P}{\Gamma \vdash \overline{x}\langle z\rangle.P}$	(πTCommOut)
$\dfrac{\Gamma, a:\sigma \vdash P}{\Gamma \vdash \mathsf{new}\ a:\sigma\ P}$	(πTRes)
$\dfrac{\Gamma \vdash P_1 \quad \Gamma \vdash P_2}{\Gamma \vdash P_1 \mid P_2}$	(πTPar)
$\dfrac{\Gamma \vdash P}{\Gamma \vdash !P}$	(πTRepl)

The type of communication channel is denoted by σ and is given by the grammar:

$$\sigma ::= \mathrm{CH}() \mid \mathrm{CH}(\sigma)$$

The type environment of the term P and the type formula in the type system of the π-calculus are defined in the following way:

$$\Gamma = \{\forall a:\sigma \mid a \in fn_\pi(P)\}$$

$$\Gamma \vdash a:\sigma \quad \Gamma \vdash P$$

where $a:\sigma$ is pair of communication name *a* and its communication channel type σ. Formula $\Gamma \vdash a:\sigma$ denotes that $a:\sigma \in \Gamma$ (where σ is the unique type of name *a*) and formula $\Gamma \vdash P$ denotes that term *P* is correctly typed in type environment Γ.

The type formulas in the type system of π-calculus are derived using the typing rules shown in Table 11.

The main idea of communication channels encoding in the system of mobile ambients is the representation of the channel by an ambient. The process term whose prefix is a communication action on channel *x* is expressed by a thread which is at first moved to an ambient $x[\ldots]$ and then it is moved back to the original ambient. While the move operation allows moving only to the neighbor (concurrent) ambient, we must define ambient *p* concurrent to ambient of channel *x*. The ambient *p* will encode the following term *P* from the π-calculus i.e. $p[[\![P]\!]] \mid x[\ldots]$.

Table 12
Encoding of π-calculus terms

$[\![P]\!]_{\mathcal{N}} = p[[\![P]\!]] \mid [\![\mathcal{N}]\!]$
$[\![\{a_1,\ldots,a_k\}]\!] = a_1[\ldots] \mid \ldots \mid a_k[\ldots]$, ´where *k* the number of names in $\mathcal{N}$
$[\![x(y:\sigma).P]\!] = move\ x.(y:[\![\sigma]\!]).move\ p.[\![P]\!]$
$[\![\overline{x}\langle z\rangle.P]\!] = move\ x.\langle z\rangle.move\ p.[\![P]\!]$
$[\![\text{new } a:\sigma\ P]\!] = (\nu a:[\![\sigma]\!])(a[out\ p] \mid [\![P]\!])$
$[\![P_1 \mid P_2]\!] = [\![P_1]\!] \mid [\![P_2]\!]$
$[\![!P]\!] = ![\![P]\!]$
$[\![\Gamma]\!] = \{p:\mathbf{P}[\mathcal{B}_p], \forall a:[\![\sigma]\!] \mid a:\sigma \in \Gamma\}$
$[\![\sigma]\!] = \mathbf{P}[\mathcal{B}_i]$, where *i* is the level of nested $\mathrm{CH}()$ in σ

The sequence of hierarchical types $\mathrm{CH}(), \mathrm{CH}(\mathrm{CH}()), \ldots, \mathrm{CH}^l()$ is expressed as the same sequence of behavioral schemes $\mathcal{B}_0, \mathcal{B}_1, \ldots, \mathcal{B}_l$, where *l* is the deepest level of nested $\mathrm{CH}()$. The behavioral schemes have following structure:

$$\begin{aligned}
\mathcal{B}_0 &= (\bot, \{\mathcal{B}, \mathcal{B}_p\}, \{\mathcal{B}_p\}, \{\mathcal{B}_p\}) \\
\mathcal{B}_1 &= (\mathbf{P}[\mathcal{B}_0], \{\mathcal{B}, \mathcal{B}_p\}, \{\mathcal{B}_p\}, \{\mathcal{B}_p\}) \\
&\vdots \\
\mathcal{B}_i &= (\mathbf{P}[\mathcal{B}_{i-1}], \{\mathcal{B}, \mathcal{B}_p\}, \{\mathcal{B}_p\}, \{\mathcal{B}_p\}) \\
&\vdots \\
\mathcal{B}_l &= (\mathbf{P}[\mathcal{B}_{l-1}], \{\mathcal{B}, \mathcal{B}_p\}, \{\mathcal{B}_p\}, \{\mathcal{B}_p\})
\end{aligned}$$

where $\mathcal{B}_p$ is the behavioral scheme of ambient p and $\mathcal{B}$ is the behavioral scheme of the whole encoded process term, i.e. term $p[[\![P]\!]] \mid a_1[\ldots] \mid \ldots \mid a_k[\ldots]$. Behavioral scheme $\mathcal{B}_p$ of ambient p has following structure:

$$\mathcal{B}_p = (\bot, \{\mathcal{B}\}, \varnothing, \{\mathcal{B}_0, \mathcal{B}_1, \ldots, \mathcal{B}_l\})$$

The encoding of term P with the set of name $\mathcal{N}$ and type system Γ of π-calculus to mobile ambient is given by Table 12.

The correctness of π-calculus encoding is shown in following two theorems.

Theorem 1: (respecting types) Let P is the term of π-calculus with the set of names $\mathcal{N}$ and $\Gamma \vdash P$. Then $[[\![\Gamma]\!] \vdash [[\![P]\!]_{\mathcal{N}} : \mathbf{P}[\mathcal{B}]$ for some behavioral scheme $\mathcal{B}$.

Proof: By induction on the structure of the process.

Let $P = x(y:\sigma).P'$ and according (πTCommIn) there is $\Gamma \vdash x(y:\sigma).P'$ by assumption $\Gamma \vdash x : \mathrm{CH}(\sigma)$ and $\Gamma, y:\sigma \vdash P'$. After encoding we get $[[\![P]\!]_{\mathcal{N}} = p[move\ x.(y:[[\![\sigma]\!]).move\ p.[[\![P']\!]] \mid [[\![\mathcal{N}]\!]$, $[[\![\Gamma]\!] = \{p : \mathbf{P}[\mathcal{B}_p], \forall a : [[\![\sigma]\!] \mid a : \sigma \in \Gamma\}$, $[[\![\sigma]\!] = \mathbf{P}[\mathcal{B}_i]$, and $[[\![\mathrm{CH}(\sigma)]\!] = \mathbf{P}[\mathcal{B}_{i-1}]$. Then according (TPar), (TAmb), (TAct), (TMove), and (TCommIn) there is $[[\![\Gamma]\!] \vdash [[\![P]\!]_{\mathcal{N}} : \mathbf{P}[\mathcal{B}]$ for some $\mathcal{B}$.

Let $P = \overline{x}\langle z\rangle.P'$ and according (πTCommOut) there is $\Gamma \vdash \overline{x}\langle z\rangle.P'$ by assumption $\Gamma \vdash x : \mathrm{CH}(\sigma)$, $\Gamma \vdash z : \sigma$, and $\Gamma \vdash P'$. After encoding we get $[[\![P]\!]_{\mathcal{N}} = p[move\ x.\langle z\rangle.move\ p.[[\![P']\!]] \mid [[\![\mathcal{N}]\!]$, $[[\![\Gamma]\!] = \{p : \mathbf{P}[\mathcal{B}_p], \forall a : [[\![\sigma]\!] \mid a : \sigma \in \Gamma\}$, $[[\![\sigma]\!] = \mathbf{P}[\mathcal{B}_i]$, and $[[\![\mathrm{CH}(\sigma)]\!] = \mathbf{P}[\mathcal{B}_{i-1}]$. Then according (TPar), (TAmb), (TAct), (TMove), and (TCommOut) there is $[[\![\Gamma]\!] \vdash [[\![P]\!]_{\mathcal{N}} : \mathbf{P}[\mathcal{B}]$ for some $\mathcal{B}$.

Let $P = \mathsf{new}\ a:\sigma\ P'$ and according (πTRes) there is $\Gamma \vdash \mathsf{new}\ a:\sigma\ P'$ if $\Gamma, a:\sigma \vdash P'$. After encoding we get $[[\![P]\!]_{\mathcal{N}} = p[(\nu a : [[\![\sigma]\!])(a[out\ p] \mid [[\![P']\!])] \mid [[\![\mathcal{N}]\!]$, $[[\![\Gamma]\!] = \{p : \mathbf{P}[\mathcal{B}_p], \forall a : [[\![\sigma]\!] \mid a : \sigma \in \Gamma\}$, and $[[\![\sigma]\!] = \mathbf{P}[\mathcal{B}_i]$. Then according (TPar), (TAmb), (TRes), again according (TPar), (TAmb), and according (TOut) there is $[[\![\Gamma]\!] \vdash [[\![P]\!]_{\mathcal{N}} : \mathbf{P}[\mathcal{B}]$ for some $\mathcal{B}$.

Let $P = P' \mid P''$ and according (πTPar) there is $\Gamma \vdash P' \mid P''$ by assumption $\Gamma \vdash P'$ and $\Gamma \vdash P''$. After encoding we get $[[\![P]\!]_{\mathcal{N}} = p[[[\![P']\!] \mid [[\![P'']\!])] \mid [[\![\mathcal{N}]\!]$, $[[\![\Gamma]\!] = \{p : \mathbf{P}[\mathcal{B}_p], \forall a : [[\![\sigma]\!] \mid a : \sigma \in \Gamma\}$, and $[[\![\sigma]\!] = \mathbf{P}[\mathcal{B}_i]$. Then according (TPar), (TAmb), and again according (TPar) there is $[[\![\Gamma]\!] \vdash [[\![P]\!]_{\mathcal{N}} : \mathbf{P}[\mathcal{B}]$ for some $\mathcal{B}$.

Let $P = !P'$ and according (πTRepl) there is $\Gamma \vdash !P'$ if $\Gamma \vdash P'$. After encoding we get $[[\![P]\!]_{\mathcal{N}} = p[![[\![P']\!])] \mid [[\![\mathcal{N}]\!]$, $[[\![\Gamma]\!] = \{p : \mathbf{P}[\mathcal{B}_p], \forall a : [[\![\sigma]\!] \mid a : \sigma \in \Gamma\}$, and

$[\![\sigma]\!] = \mathbf{P}[\mathcal{B}_i]$. Then according (TPar), (TAmb), and (TRepl) there is $[\![\Gamma]\!] \vdash [\![P]\!]_{\mathcal{N}} : \mathbf{P}[\mathcal{B}]$ for some $\mathcal{B}$.

Theorem 2: (encoding correctness) Let P is the term of π-calculus with the set of name $\mathcal{N}$. If $P \equiv_\pi Q$ then $[\![P]\!]_{\mathcal{N}} \equiv [\![Q]\!]_{\mathcal{N}}$ and if $P \to_\pi Q$ then $[\![P]\!]_{\mathcal{N}} \to^* [\![Q]\!]_{\mathcal{N}}$.

Proof: By induction on the structure of the process.

1. Let P is the term of π-calculus with the set of names $\mathcal{N}$. If $P \equiv_\pi Q$ then $[\![P]\!]_{\mathcal{N}} \equiv [\![Q]\!]_{\mathcal{N}}$.

(πSRefl) Let $P \equiv_\pi P$ then $[\![P]\!]_{\mathcal{N}} \equiv [\![P]\!]_{\mathcal{N}}$.

(πSSymm) Let $Q \equiv_\pi P$ then $[\![Q]\!]_{N} \equiv [\![P]\!]_{N}$.

(πSTrans) Let $P \equiv_\pi R$ and $R \equiv_\pi Q$ for some R and let $[\![P]\!]_{\mathcal{N}} \equiv [\![R]\!]_{\mathcal{N}}$ and $[\![R]\!]_{\mathcal{N}} \equiv [\![Q]\!]_{\mathcal{N}}$ then $[\![P]\!]_{\mathcal{N}} \equiv [\![Q]\!]_{\mathcal{N}}$.

(πSCommIn) Let $P = x(y:\sigma).P'$, $Q = x(y:\sigma).Q'$, $P' \equiv_\pi Q'$ end let $[\![P']\!]_{\mathcal{N}} \equiv [\![Q']\!]_{\mathcal{N}}$. According definition of the structural congruence $\equiv$ there is $p[move\ x.(y:[\![\sigma]\!]).move\ p.[\![P']\!]] \mid [\![\mathcal{N}]\!] \equiv p[move\ x.(y:[\![\sigma]\!]).move\ p.[\![Q']\!]] \mid [\![\mathcal{N}]\!]$, what is $[\![x(y:\sigma).P']\!]_{\mathcal{N}} \equiv [\![x(y:\sigma).Q']\!]_{\mathcal{N}}$.

(πSCommOut) Let $P = \overline{x}\langle z\rangle.P'$, $Q = \overline{x}\langle z\rangle.Q'$, $P' \equiv_\pi Q'$ and let $[\![P']\!]_{\mathcal{N}} \equiv [\![Q']\!]_{\mathcal{N}}$. According definition of the structural congruence $\equiv$ there is $p[move\ x.\langle z\rangle.move\ p.[\![P']\!]] \mid [\![\mathcal{N}]\!] \equiv p[move\ x.\langle z\rangle.move\ p.[\![Q']\!]] \mid [\![\mathcal{N}]\!]$, what is $[\![\overline{x}\langle z\rangle.P']\!]_{\mathcal{N}} \equiv [\![\overline{x}\langle z\rangle.Q']\!]_{\mathcal{N}}$.

(πSRes) Let $P = \mathsf{new}\ a:\sigma\ P'$, $Q = \mathsf{new}\ a:\sigma\ Q'$, $P' \equiv_\pi Q'$ and let $[\![P']\!]_{\mathcal{N}} \equiv [\![Q']\!]_{\mathcal{N}}$. According definition of the structural congruence $\equiv$ there is $p[(\nu a:[\![\sigma]\!])(a[out\ p] \mid [\![P']\!])] \mid [\![\mathcal{N}]\!] \equiv p[(\nu a:[\![\sigma]\!])(a[out\ p] \mid [\![Q']\!])] \mid [\![\mathcal{N}]\!]$, what is $[\![\mathsf{new}\ a:\sigma\ P']\!]_{\mathcal{N}} \equiv [\![\mathsf{new}\ a:\sigma\ Q']\!]_{\mathcal{N}}$.

(πSPar) Let $P = P' \mid R$, $Q = Q' \mid R$, $P' \equiv_\pi Q'$ and let $[\![P']\!]_{\mathcal{N}} \equiv [\![Q']\!]_{\mathcal{N}}$. According definition of the structural congruence $\equiv$ there is $p[[\![P']\!] \mid [\![R]\!]] \mid [\![\mathcal{N}]\!] \equiv p[[\![Q']\!] \mid [\![R]\!]] \mid [\![\mathcal{N}]\!]$, what is $[\![P' \mid R]\!]_{\mathcal{N}} \equiv [\![Q' \mid R]\!]_{\mathcal{N}}$.

(πSRepl) Let $P = !P'$, $Q = !Q'$, $P' \equiv_\pi Q'$ and let $[\![P']\!]_{\mathcal{N}} \equiv [\![Q']\!]_{\mathcal{N}}$. According definition of the structural congruence $\equiv$ there is $p[![\![P']\!]] \mid [\![\mathcal{N}]\!] \equiv p[![\![Q']\!]] \mid [\![\mathcal{N}]\!]$, what is $[\![!P']\!]_{\mathcal{N}} \equiv [\![!Q']\!]_{\mathcal{N}}$.

(πSParComm) Let $P = P' \mid P''$ a $Q = P'' \mid P'$. According definition of the structural congruence $\equiv$ there is $p[[\![P']\!] \mid [\![P'']\!]] \mid [\![\mathcal{N}]\!] \equiv p[[\![P'']\!] \mid [\![P']\!]] \mid [\![\mathcal{N}]\!]$, what is $[\![P' \mid P'']\!]_{\mathcal{N}} \equiv [\![P'' \mid P']\!]_{\mathcal{N}}$.

(πSParAssoc) Let $P = (P' \mid P'') \mid P'''$ and $Q = P' \mid (P'' \mid P''')$. According definition of the structural congruence $\equiv$ there is $p[[\![P']\!] \mid ([\![P'']\!] \mid [\![P''']\!])] \mid [\![\mathcal{N}]\!] \equiv p[([\![P']\!] \mid [\![P'']\!]) \mid [\![P''']\!]] \mid [\![\mathcal{N}]\!]$, what is $[\![(P' \mid P'') \mid P''']\!]_{\mathcal{N}} \equiv [\![P' \mid (P'' \mid P''')]\!]_{\mathcal{N}}$.

(πSReplPar) Let $P = !P'$ and $Q = P' \mid !P'$. According definition of the structural congruence $\equiv$ there is $p[![\![P']\!]] \mid [\![\mathcal{N}]\!] \equiv p[[\![P']\!] \mid ![\![P']\!]] \mid [\![\mathcal{N}]\!]$, what is $[\![!P']\!]_{\mathcal{N}} \equiv [\![P' \mid !P']\!]_{\mathcal{N}}$.

(πSResRes) Let $P = \mathsf{new}\ a : \sigma\ \mathsf{new}\ b : \sigma'\ P'$, $Q = \mathsf{new}\ b : \sigma' \mathsf{new}\ a : \sigma\ P'$ and $a \neq b$. According definition of the structural congruence $\equiv$ there is $p[(\nu a : [\![\sigma]\!])(a[out\ p] \mid (\nu b : [\![\sigma']\!])(b[out\ p] \mid [\![P']\!]))] \mid [\![\mathcal{N}]\!] \equiv p[(\nu b : [\![\sigma']\!])(b[out\ p] \mid (\nu a : [\![\sigma]\!])(a[out\ p] \mid [\![P']\!]))] \mid [\![\mathcal{N}]\!]$, what is $[\![\mathsf{new}\ a : \sigma\ \mathsf{new}\ b : \sigma'\ P']\!]_{\mathcal{N}} \equiv [\![\mathsf{new}\ b : \sigma'\ \mathsf{new}\ a : \sigma\ P']\!]_{\mathcal{N}}$.

(πSResPar) Let $P = \mathsf{new}\ a : \sigma\ (P' \mid P'')$, $Q = \mathsf{new}\ a : \sigma\ P' \mid P''$ a $a \notin fn_{\pi}(P')$. There is $fn([\![P']\!]) = fn_{\pi}(P') \cup \{p\}$ and $a \neq p$. If $a \notin fn_{\pi}(P')$, then $a \notin fn([\![P']\!])$. According definition of the structural congruence $\equiv$ there is $p[(\nu a : [\![\sigma]\!])([\![P']\!] \mid [\![P'']\!])] \mid [\![\mathcal{N}]\!] \equiv p[(\nu a : [\![\sigma]\!])[\![P']\!] \mid [\![P'']\!]] \mid [\![\mathcal{N}]\!]$, what is $[\![\mathsf{new}\ a : \sigma\ (P' \mid P'')]\!]_{\mathcal{N}} \equiv [\![\mathsf{new}\ a : \sigma\ P' \mid P'']\!]_{\mathcal{N}}$.

(πSResSkip) Let $P = \mathsf{new}\ a : \sigma\ P'$, $Q = P'$ and $a \notin fn_{\pi}(P')$. There is $fn([\![P']\!]) = fn_{\pi}(P') \cup \{p\}$ and $a \neq p$. If $a \notin fn_{\pi}(P')$, then $a \notin fn([\![P']\!])$. According definition of the structural congruence $\equiv$ there is $p[(\nu a : [\![\sigma]\!])(a[out\ p] \mid [\![P']\!])] \mid [\![\mathcal{N}]\!] \equiv p[(\nu a : [\![\sigma]\!])a[out\ p] \mid [\![P']\!] \mid] \mid [\![\mathcal{N}]\!]$ and $a \notin fn([\![P']\!])$, what is $[\![\mathsf{new}\ a : \sigma\ P']\!]_{\mathcal{N}} \equiv [\![P']\!]_{\mathcal{N}}$.

2. Let P is the term of π-calculus with the set of names $\mathcal{N}$. If $P \rightarrow_{\pi} Q$ then $[\![P]\!]_{\mathcal{N}} \rightarrow^* [\![Q]\!]_{\mathcal{N}}$.

(πRRes) Let $P = \mathsf{new}\ a : \sigma\ P'$, $Q = \mathsf{new}\ a : \sigma\ Q'$ and $P' \rightarrow_{\pi} Q'$. We need to show, if $fn_{\pi}(\mathsf{new}\ a : \sigma\ P') \subseteq \mathcal{N}$, then $[\![\mathsf{new}\ a : \sigma\ P']\!]_{\mathcal{N}} \rightarrow^* [\![\mathsf{new}\ a : \sigma\ Q']\!]_{\mathcal{N}}$. If $fn_{\pi}(\mathsf{new}\ a : \sigma\ P') \subseteq \mathcal{N}$, then $fn_{\pi}(P') \subseteq \mathcal{N} \cup \{a\}$, what means that $a \notin \mathcal{N}$. According $[\![P']\!]_{\mathcal{N} \cup \{a\}} \rightarrow^* [\![Q']\!]_{\mathcal{N} \cup \{a\}}$ and by repeat usage of (RRes) and structural congruence $\equiv$ we get $(\nu a : [\![\sigma]\!])[\![P']\!]_{\mathcal{N} \cup \{a\}} \rightarrow^* (\nu a : [\![\sigma]\!])[\![Q']\!]_{\mathcal{N} \cup \{a\}}$. Then $(\nu a : [\![\sigma]\!])[\![P']\!]_{\mathcal{N} \cup \{a\}} = (\nu a : [\![\sigma]\!])(p[[\![P']\!] \mid [\![\mathcal{N} \cup \{a\}]\!]) \equiv (\nu : [\![\sigma]\!])(p[[\![P']\!] \mid [\![\mathcal{N}]\!] \mid a[\ldots]) \equiv p[[\![\mathsf{new}\ a : \sigma\ P']\!]] \mid [\![\mathcal{N}]\!] = [\![\mathsf{new}\ a : \sigma\ P']\!]_{\mathcal{N}}$ and the same way we get

$(\nu a:[\![\sigma]\!])[\![Q']\!]_{\mathcal{N}\cup\{a\}} \equiv [\![\text{new } a:\sigma\ Q']\!]_{\mathcal{N}}$, what is $[\![\text{new } a:\sigma\ P']\!]_{\mathcal{N}\cup\{a\}} \rightarrow^* [\![\text{new } a:\sigma\ Q']\!]_{\mathcal{N}}$.

(πRPar) Let $P = P' \mid R$, $Q = Q' \mid R$, $P' \rightarrow Q'$ and let $[\![P']\!]_{\mathcal{N}} \rightarrow^* [\![Q']\!]_{\mathcal{N}}$ and by repeat usage of (RPar) and structural congruence $\equiv$ we get $[\![P']\!]_{\mathcal{N}} \mid [\![R]\!] \rightarrow^* [\![Q']\!]_{\mathcal{N}} \mid [\![R]\!]$, what is $[\![P' \mid R]\!]_{\mathcal{N}} \rightarrow^* [\![Q' \mid R]\!]_{\mathcal{N}}$.

(πRComm) Let $P = x(y:\sigma).P' \mid \overline{x}\langle z\rangle.P''$ and $Q = P'\{y \leftarrow z\} \mid P''$. We need to show, if $fn_\pi(x(y:\sigma).P' \mid \overline{x}\langle z\rangle.P'') \subseteq \mathcal{N}$, then $[\![x(y:\sigma).P' \mid \overline{x}\langle z\rangle.P'']\!]_{\mathcal{N}} \rightarrow^* [\![P'\{y \leftarrow z\} \mid P'']\!]_{\mathcal{N}}$. Let $[\![x(y:\sigma).P' \mid \overline{x}\langle z\rangle.P'']\!]_{\mathcal{N}} = p[move\ x.(y:[\![\sigma]\!]).move\ p.[\![P']\!] \mid move\ x.\langle z\rangle.move\ x.[\![P'']\!]] \mid [\![\mathcal{N}]\!]$. By assumption $x \in \mathcal{N}$, $[\![\mathcal{N}]\!]$ must contain $x[\ldots]$. After reduction we get $[\![P']\!]\{y \leftarrow z\} \mid [\![P'']\!] \mid [\![\mathcal{N}]\!]$, where $[\![P']\!]\{y \leftarrow z\}$ is equivalent to $[\![P'\{y \leftarrow z\}]\!]$. That gives $[\![x(y:\sigma).P' \mid \overline{x}\langle z\rangle.P'']\!]_{\mathcal{N}} \rightarrow^* [\![P'\{y \leftarrow z\} \mid P'']\!]_{\mathcal{N}}$.

(πRStruct) Let $P \equiv_\pi P'$, $Q \equiv_\pi Q'$, $P' \rightarrow_\pi Q'$ and let $[\![P']\!]_{\mathcal{N}} \equiv [\![P]\!]_{\mathcal{N}}, [\![P]\!]_{\mathcal{N}} \rightarrow^* [\![Q]\!]_{\mathcal{N}}, [\![Q]\!]_{\mathcal{N}} \equiv [\![Q']\!]_{\mathcal{N}}$. According transitivity of structural congruence $\equiv$ we get $[\![P']\!]_{\mathcal{N}} \rightarrow^* [\![Q']\!]_{\mathcal{N}}$.

Conclusions

The main choice in designing a calculus with mobile (lightweight) processes is one of the mobility primitives for them. We have chosen to introduce, for the moment, only one primitive *move* since it is already present, though in a context of immobile locations, in well established concurrent calculus, such as Dπ [7]. Also, this primitive might be argued to naturally model the elementary instruction by which an agent moves from one location to another at the same level. A natural alternative, or a natural extension, would be a thread mobility analogous to that for ambients, i.e., capabilities to go one step up or down the tree hierarchy, by exiting or entering an ambient.

We used this approach to encode standard π-calculus which expresses the communication of named channels by our approach in a mobile ambient system. The encoding was presented as an expressiveness test of our ambient calculus with behavioral schemes [8].

References

[1] Cardelli, L., Gordon, A. D.: Mobile Ambients. Theoretical Computer Science, Vol. 240, No. 1, 2000, pp. 177-213

[2] Milner, R., Parrow, J., Walker, D.: A Calculus of Mobile Processes, Part 1 – 2. Information and Computation, Vol. 100, No. 1, 1992, pp. 1-77

[3] Levi, F., Sangiorgi, D.: Controlling Interference in Ambients. Proceedings of POPL'00, ACM Press, New York, 2000, pp. 352-364

[4] Bugliesi, M., Castagna, G.: Secure Safe Ambients. Proceedings of POPL'01, ACM Press, New York, 2001, pp. 222-235

[5] Bugliesi, M., Castagna, G., Crafa, S.: Boxed Ambients. In B. Pierce (ed.): TACS'01, LNCS 2215, Springer Verlag, 2001, pp. 38-63

[6] Fuggeta, A., Picco, G. P., Vigna, G.: Understanding Code Mobility. IEEE Transactions on Software Engineering, Vol. 24, No. 5, May 1998, pp. 342-361

[7] Hennessey, M., Riely, J.: Resource Access Control in Systems of Mobile Agents. Technical Report 2/98, Computer Science Department, University of Sussex, 1998

[8] Tomasek, M.: Expressing Dynamics of Mobile Programs. PhD thesis, Technical university of Kosice, 2004