Expressing Dynamics of Mobile Programs by Typing

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: This paper presents a tool for expressing dynamics of mobile code applications, which is based on mobile ambients. The new primitives for the ambient calculus are suitable for expressing different kinds of mobility and they avoid ambiguities and possible maliciousness of some constructions. The type system statically defines and checks access rights for authorization of ambients and threads to move by application of the process behavioral scheme. We proved the soundness theorem for the type system and we demonstrated the system by showing how to model some common applications.

Keywords: ambient calculus, mobile code, type system

1 Introduction

A huge amount of computational entities distributed worldwide, exchanging data, moving from one place to another, interacting with each other (either cooperatively or competitively), give rise to global computing activity. Computation has therefore to be abstractly described as something that develops not only in time and in memory space, either sequentially (λ-calculus) or as a dynamic set of concurrent processes (π-calculus), but also in a wide geographical and administrative space (mobile ambients).

The calculus of mobile ambients [1], building upon the concurrency paradigm represented by the π-calculus [2], introduced the notion of an ambient as a bounded place where concurrent computation happens, which can contain nested subambients in a tree structure, and which can move in and out of other ambients, i.e., up and down the tree (thus rearranging the structure itself). Direct communication can only occur locally within each ambient (through a common anonymous channel). Communication between different ambients has to be mediated by movement and by dissolution of ambient boundaries.

The ambition of mobile ambients is in general to express mobile computation and mobile computing. Mobile ambients can express in natural way dynamic properties (communication and mobility) of mobile code systems, but there is still question of deeper control and verification of mobility properties (like access rights or mobility control).

2 Additional Requirements for Mobile Ambients

Mobile ambients allows modeling 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 makes that in principle there are no deferences 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 sequence of primitive actions to be executed sequentially, generally in concurrency with other threads. They can perform communication and drive their containers through the spatial hierarchy, but cannot individually go from one ambient to another. Ambints 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 autonomous behavior of ambients (this is also important e.g. for objects).

Communication between ambients is represented by the exchange of other ambient of usually shorter life, which have their boundaries dissolved by an *open* action so as to expose their internal threads performing the 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 a mobile agent. Remote communication is usually emulated as a movement of such ambients (communication packages) in the hierarchy structure.

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 have to introduce new operation *move* for moving threads between ambients. The idea comes from mobile code programming paradigms [6] where moving threads can express 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.

Such adaptations of mobile ambients operations we can express computational entities of mobile programs in more natural way. Another purpose for this approach is to prefer simplicity and understandability of designed type system for mobile ambients later on.

3 The Calculus

We define abstract syntax and operational semantics of our calculus. It is based on abstract syntax and operational semantics of ambient calculus including our new constructions

3.1 Abstract 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 primitive *move* for moving threads between ambients. We allow synchronous output and the asynchronous version is its particular case.

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

$(n:\mu).P$	synchronous input

Table 1
Abstract syntax

We introduce types already in the term syntax, in the synchronous input and in the name restriction. The defined terms are not exactly the terms of our calculus, since the type constructions are not yet taken into account, this is done by the typing rules in the next section.

3.2 Operational Semantics

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

Each name of the process term can figure either as free (Table 2) or bound (Table 3).

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

Table 2
Free names

$bn(n) = \varnothing$
$bn(in\ M) = bn(M)$

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

Table 3
Bound names

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 4 and it is standard for mobile ambients. The (SAmbNull) rule is added to get a form of garbage collection, because of absence of the *open* operation.

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 (associativity, commutativity and inactivity):	
$P \mid Q \equiv Q \mid P$	(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)

Table 4
Structural congruence

In addition, we identify processes up to renaming of bound names (α-conversion) as shown in Table 5. By this we mean that these processes are understood to be identical (e.g. by choosing an appropriate representation), as opposed to structurally equivalent.

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

Table 5
α-conversion

The reduction rules in Table 6 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.

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)

$(n:\mu).P \mid \langle M \rangle.Q \to P\{n \leftarrow M\} \mid Q$	(RComm)
structural reductions:	
$P \to Q \Rightarrow P \mid R \to Q \mid R$	(RPar)
$P \to Q \Rightarrow n[P] \to n[Q]$	(RAmb)
$P \to Q \Rightarrow (\nu n:\mathbf{P}[\mathcal{B}])P \to (\nu n:\mathbf{P}[\mathcal{B}])Q$	(RRes)
$P' \equiv P, P \to Q, Q \equiv Q' \Rightarrow P' \to Q'$	(RStruct)

Table 6
Reduction rules

4 Type System

From the huge amount of complex behavioral properties of mobile processes we abstract (extract) the type system that is simple enough to be easily used for expressing communication and mobility properties of mobile ambients.

4.1 Types and Behavioral Scheme

We define communication types where both peers, receiver and sender, must be of the same message type. This allows to keep the sense of communication. It also secures the communication while only exchange of the correct messages is allowed.

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.

Types are defined in Table 7 where we present communication types and message types.

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

Table 7
Types

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

4.2 Typing Rules

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 8. 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. It means such a process has the correct behavior. The type assignment system is clearly syntax-directed and keeps the system simple enough.

$\dfrac{n : \mu \in \Gamma}{\Gamma \vdash n : \mu}$	(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)

$\dfrac{\Gamma \vdash M : \mathbf{P}[\mathcal{B}] \quad \mathcal{B} \in \mathit{Move}(\mathcal{B}')}{\Gamma \vdash \mathit{move}\ M : \mathbf{O}[\mathcal{B} \mapsto \mathcal{B}']}$	(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)
$\dfrac{\Gamma \vdash P : \mathbf{P}[\mathcal{B}] \quad \Gamma \vdash M : \mathbf{P}[\mathcal{B}] \quad \mathcal{B}' \in \mathit{Reside}(\mathcal{B})}{\Gamma \vdash M[P] : \mathbf{P}[\mathcal{B}']}$	(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)

Table 8
Typing rules

4.3 Soundness of the System

The usual property of subject reduction holds, which guarantees the soundness of the system by ensuring that typing is preserved by computation.

Soundness theorem: Let $\Gamma \vdash P : \mathbf{P}[\mathcal{B}]$ for some $\mathcal{B}$. Then:

1. $P \equiv Q$ implies $\Gamma \vdash Q : \mathbf{P}[\mathcal{B}]$
2. $P \to Q$ implies $\Gamma \vdash Q : \mathbf{P}[\mathcal{B}]$

Proof: The proof is standard, by induction on the derivations of $P \equiv Q$ and $P \to Q$.

Conclusions

We defined formal tool for expressing dynamics of mobile code applications, which is based on theory of mobile ambients. Presented changes to the ambient calculus are suitable for expressing different kinds of mobility and they avoid ambiguities and possible maliciousness of some constructions. The type system statically defines and checks access rights for authorization of ambients and threads to move by application of the process behavioral scheme. The usage of type system is limited by its very simplicity and it does not prevent more restrictive properties from being checked at runtime. We proved the soundness theorem for the type system and we demonstrated the system by showing how to model some common applications. We provided a simple language for distributed system of mobile agents. As an expressiveness test, we showed that well-known π-calculus of concurrency and mobility can be encoded in our calculus in a natural way.

More results, examples and proofs can be found in our detailed work [7].

Acknowledgement

This paper was supported by the grants Nr. 1/3135/06, Nr. 1/2176/05 and Nr. 1/4073/07 of the Slovak Grant Agency.

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] TOMÁŠEK, M.: Expressing dynamics of mobile programs. PhD thesis, Technical university of Košice, 2004 (in Slovak)