A Typed Calculus of Mobile Computation

A.D. Gordon (Microsoft)

(based on joint work with L. Cardelli (Microsoft) and G. Ghelli (Pisa University))

Foundations of Mobile Computation

December 16–17, 1999

Institute of Mathematical Sciences

Chennai, India

Mobile Computation's Taking Off

Mobile hardware devices are taking off

- devices: laptops, palmtops, smartcards, . . .
- protocols: Mobile IP, WAP, Bluetooth, . . .

Mobile code and mobile software agents are taking off

- Facile, Telescript, Obliq, Java applets, ECMAscript, WAPscript, . . .
- Mobile extensions of Java: Voyager, Odyssey, Aglets, . . .

Security risks arise from both mobile devices and mobile agents

- secrecy risks: e.g., protect login credentials from smartcard reader
- integrity risks: e.g., prevent malicious applet from formatting the hard drive

Our Aims

Various kinds of places, and of navigation between places, are fundamental to programming mobile computation.

We formalize these places as **ambients**, and study a small set of mobility primitives with a precise semantics: the **ambient calculus**.

Calculi of functions, processes, and objects clarify existing styles of computation. Sometimes they suggest better programming habits too.

Our goal is that the theory and implementation of the ambient calculus will do the same for mobile computation.

Specifically, this talk uses ambients to develop type systems for mobility, adaptable for use in a bytecode verifier, for example.

The Untyped Ambient Calculus

Orientation: Ambients

An ambient is a named, bounded place where computation happens.

An ambient is both a unit of mobility—of either software and hardware—and a security perimeter.

Ambient security rests on the controlled distribution of suitable credentials, or **capabilities**, derived from unforgeable **names**.

One goal of our calculus is to develop a flexible, precise, secure, and typeful programming model for mobile software components.

Mobile Ambients: a packet from A to B

$$\overbrace{A[\underbrace{msg[\mathit{out}\ A.\mathit{in}\ B \mid \langle M \rangle]}_{A \to B : M}]}^{\text{Machine } A} \mid \overbrace{B[\underbrace{\mathit{open}\ msg.(x).P}_{\text{receive } x;\ P}]}^{\text{Machine } B}$$

- Ambients may model both machines and packets
- Ambients are mobile: $msg[\cdots]$ moves out of A and into B
- Ambients are boundaries: passage is regulated by **capabilities**

  You need capability *out* A to exit A; you need capability *in* B to enter B

Ambient Behaviour, By Example

We illustrate the four basic reduction rules of the calculus:

$$
\begin{aligned}
& A[\mathit{msg}[\mathit{out}\ A.\mathit{in}\ B \mid \langle M \rangle]] \mid B[\mathit{open}\ \mathit{msg}.(x).P] \\
\rightarrow\ & A[] \mid \mathit{msg}[\mathit{in}\ B \mid \langle M \rangle] \mid B[\mathit{open}\ \mathit{msg}.(x).P] \\
\rightarrow\ & A[] \mid B[\mathit{msg}[\langle M \rangle] \mid \mathit{open}\ \mathit{msg}.(x).P] \\
\rightarrow\ & A[] \mid B[\langle M \rangle \mid (x).P] \\
\rightarrow\ & A[] \mid B[P\{x \leftarrow M\}]
\end{aligned}
$$

Mobility and Communication Primitives:

$M ::=$	expression
n	ambient name
in M	can enter into M
out M	can exit out of M
open M	can open M
$P, Q, R ::=$	process
$(\nu n)P$	restriction
$\mathbf{0}$	inactivity
$P \mid Q$	composition
$!P$	replication
$M[P]$	ambient
$M.P$	action
$(x_1, \ldots, x_k).P$	input action
$\langle M_1, \ldots, M_k \rangle$	asynchronous output action

Example: Semantics of a Distributed Language

Programming Model

There is a flat collection of named nodes (or locations), each of which contains a group of named channels and anonymous threads:

$$
\begin{array}{l}
\textit{node}\ \mathrm{a}\ [\textit{channel}\ \mathrm{c}\ \mid \\
\qquad \textit{thread}\,[\overline{\mathrm{c}}\langle \mathrm{b} \rangle] \mid \\
\qquad \textit{thread}\,[\mathrm{c}(\mathrm{x}).\textit{go}\ \mathrm{x}]] \mid \\
\textit{node}\ \mathrm{b}\ []
\end{array}
$$

Heteregeneous models like this underly several distributed programming systems, and several distributed forms of the π-calculus.

An Encoding $[\![-]\!]$ in the Ambient Calculus

Ambients model nodes, channels, and threads. For example:

$$
\begin{aligned}
&a[[\![\mathit{channel}\ c]\!]_a \mid \\
&\quad [\![\mathit{thread}[\overline{c}\langle b\rangle]]\!]_a \mid \\
&\quad [\![\mathit{thread}[c(x).\mathit{go}\ x]]\!]_a] \mid \\
&b[]
\end{aligned}
$$

A channel consists of a buffer ambient c^b that opens up any packets named c^p sent into it:

$$[\![\mathit{channel}\ c]\!]_a = c^b[!\mathit{open}\ c^p.\mathbf{0}]$$

A thread is an anonymous ambient, with a fresh name.

An output is a packet that exits its thread, and enters a channel buffer:

$$[\![\mathit{thread}[\overline{c}\langle b\rangle]]\!]_a = (\nu t)t[\mathit{go}(\mathit{out}\ t.\mathit{in}\ c^b).c^p[\langle b, b^p\rangle]]$$

In the untyped calculus, *go* $M.n[P]$ is short for:

$$\mathit{go}\ M.n[P] \triangleq (\nu k)k[M.n[\mathit{out}\ k.P]]$$

An input is a packet that exits its thread, enters the buffer, gets opened, inputs a message, then returns to its thread. A move to x executes capabilities to exit the current node, then enter the destination node x.

$$[\![\mathit{thread}[c(x).\mathit{go}\ x]]\!]_a =$$
$$(\nu t)t[(\nu s)(\mathit{go}(\mathit{out}\ t.\mathit{in}\ c^b).c^p[(x, x^p).$$
$$\mathit{go}(\mathit{out}\ c^b.\mathit{in}\ t).s[\mathit{open}\ s.\mathit{out}\ a.\mathit{in}\ x.\mathbf{0}]] \mid$$
$$\mathit{open}\ s.s[])]$$

The name s is for synchronisation ambients $s[]$, used to delay the move until the input has completed.

A fragment of a distributed programming language:

Net ::=	network
node n [*Cro*]	node
Net | *Net*	composition of networks
Cro ::=	crowd of channels and threads
channel c	channel
thread[*Th*]	thread
Cro | *Cro*	composition of crowds
Th ::=	thread
go n.*Th*	migration
$\overline{c}\langle n_1, \ldots, n_k \rangle$	output to a channel
$c(x_1, \ldots, x_k)$.*Th*	input from a channel
...	imperative features (omitted)

Summary of the Untyped Calculus

The core calculus (without I/O) is Turing complete. The full calculus (with I/O) can naturally model the π-calculus.

It offers a simple, abstract description of classical distributed languages, where ambients model both the unit of mobility (threads) and security perimeters (network nodes).

This description of mobility is more direct and explicit than possible in most other process calculi.

Several implementations now exist.

Ambient Types I:

Exchange Types

Orientation: Types

The purpose of a type system is to prevent execution errors during the running of well-typed programs.

Typed languages emerged in the 1960s and 70s: Pascal, Algol 68, Simula, ML. Mostly, typing in these languages prevents **accidental** execution errors, e.g., $1.0 + $ "fred".

Recently, Java has popularised typing for mobile code. As well as preventing accidents, typing in Java prevents **malicious** execution errors, e.g., formatting the C drive.

Motivation for Exchange Types

In the untyped calculus, certain processes arise that make no sense:

- Process *in* $n[P]$ uses a capability as an ambient name
- Process $(\nu n)n.P$ uses an ambient name as a capability

In an implementation, these processes are execution errors.

To avoid these errors, we regulate the types of messages a process may **exchange**, that is, input or output.

Typing Input and Output

If a message M has message type W, then $\langle M \rangle$ is a process that exchanges W messages.

If $M : W$ then $\langle M \rangle : W$.

If P is a process that exchanges W messages, then $(x{:}W).P$ is also a process that exchanges W messages.

If $P : W$ then $(x{:}W).P : W$.

Typing Parallelism

Process **0** exchanges messages of any type, since it exchanges none.

0 : T for all T.

If P and Q are processes that exchange T messages, so is $P \mid Q$.

If $P : T$ and $Q : T$ then $P \mid Q : T$.

If $P : T$ then $!P : T$.

These rules ensure matching of the types of inputs and outputs from processes running in parallel.

Typing Ambients

An expression of type *Amb*$[\mathrm{T}]$ names an ambient inside which T messages are exchanged.

If M is such an expression, and P is a process that exchanges T messages, then $\mathrm{M}[\mathrm{P}]$ is correctly typed.

If $\mathrm{M} : \mathit{Amb}[\mathrm{T}]$ and $\mathrm{P} : \mathrm{T}$ then $\mathrm{M}[\mathrm{P}] : S$ for all S.

An ambient exchanges no messages, so it may be assigned any type.

Typing Capabilities

An expression of type $\mathit{Cap}[\mathrm{T}]$ is a capability that may unleash exchanges of type T.

If $\mathrm{M} : \mathit{Cap}[\mathrm{T}]$ and $\mathrm{P} : \mathrm{T}$ then $\mathrm{M}.\mathrm{P} : \mathrm{T}$.

If ambients named n exchange T messages, then the capability $\mathit{open}\ \mathrm{n}$ may unleash these exchanges.

If $\mathrm{n} : \mathit{Amb}[\mathrm{T}]$ then $\mathit{open}\ \mathrm{n} : \mathit{Cap}[\mathrm{T}]$.

Capabilities $\mathit{in}\ \mathrm{n}$ and $\mathit{out}\ \mathrm{n}$ unleash no exchanges.

If $\mathrm{n} : \mathit{Amb}[\mathrm{S}]$ then $\mathit{in}\ \mathrm{n} : \mathit{Cap}[\mathrm{T}]$ for all T.

If $\mathrm{n} : \mathit{Amb}[\mathrm{S}]$ then $\mathit{out}\ \mathrm{n} : \mathit{Cap}[\mathrm{T}]$ for all T.

Exchange Types

Types:

$W ::=$	message types
$\mathit{Amb}[\mathrm{T}]$	ambient name allowing T exchanges
$\mathit{Cap}[\mathrm{T}]$	capability unleashing T exchanges
$S, \mathrm{T} ::=$	exchange types
Shh	no exchange
$W_1 \times \cdots \times W_k$	tuple exchange

- A quiet ambient, $\mathit{Amb}[\mathit{Shh}]$, and a harmless capability, $\mathit{Cap}[\mathit{Shh}]$
- An ambient allowing exchange of harmless capabilities: $\mathit{Amb}[\mathit{Cap}[\mathit{Shh}]]$
- A capability unleashing exchanges of names of quiet ambients: $\mathit{Cap}[\mathit{Amb}[\mathit{Shh}]]$

Properties of Exchange Types

Formally, we base our type system on judgments $E \vdash M : W$ and $E \vdash P : T$, where $E = x_1{:}W_1, \ldots, x_k{:}W_k$.

Theorem (Soundness) If $E \vdash P : T$ and $P \rightarrow Q$ then $E \vdash Q : T$.

Hence, execution errors like *in* $n[P]$ and $(\nu n)n.P$ cannot arise during a computation, since they are not typeable.

Typing the Packet Example

Packet from A to B:

If A : *Amb*[*Shh*], B, *msg* : *Amb*[W], and $M, P : W$ then

$$A[\mathit{msg}[\underbrace{\mathit{out}\ A.\mathit{in}\ B}_{\mathit{Cap}[W]} \mid \langle M \rangle]]: \mid B[\underbrace{\mathit{open}\ \mathit{msg}}_{\mathit{Cap}[W]}.(x{:}W).P] : \mathit{Shh}.$$

Example: The Distributed Language

Each name has a type Ty, either $Node$ or $Ch[Ty_1, \ldots, Ty_k]$.

Two ambient names represent each source name; e.g., each channel name is represented by a buffer name and a packet name.

We translate these to ambient types so that $[\![Node]\!] = Amb[Shh]$ and $[\![Ch[Ty_1, \ldots, Ty_k]]\!] = Amb[[\![Ty_1]\!] \times [\![Ty_1]\!] \times \cdots \times [\![Ty_k]\!] \times [\![Ty_k]\!]]$.

We can prove that if a program in the distributed language is well-typed, so is its translation to the ambient calculus.

Example using Exchange Types

Assume that $c{:}\mathit{Ch}[\mathit{Node}]$. The translation of $\mathit{thread}[c(x).\mathit{go}\ x]$,

$$(\nu t)t[(\nu s)(\mathit{go}(\mathit{out}\ t.\mathit{in}\ c^b).c^p[(x, x^p).$$
$$\mathit{go}(\mathit{out}\ c^b.\mathit{in}\ t).s[\mathit{open}\ s.\mathit{out}\ a.\mathit{in}\ x.\mathbf{0}]]\ |$$
$$\mathit{open}\ s.s[])]$$

has type *Shh* assuming that:

$$a : \mathit{Amb}[\mathit{Shh}], \qquad t : \mathit{Amb}[\mathit{Shh}],$$
$$c^b, c^p : \mathit{Amb}[[\![\mathit{Node}]\!], [\![\mathit{Node}]\!]], \quad s : \mathit{Amb}[\mathit{Shh}]$$

Ambient Types II: Mobility and Locking Annotations

Regulating Mobility and Persistence

We decorate ambient types with annotations

$$Amb^{Y}[^{Z}T]$$

The locking annotation ^{Y} is either **locked** ($^{\bullet}$) or **unlocked** ($^{\circ}$).

The mobility annotation ^{Z} is either **mobile** ($^{\curvearrowright}$) or **immobile** ($^{\underline{\vee}}$).

Opening a locked ambient or moving an immobile ambient once its running is an execution error. Our type system prevents such errors.

Modifying the Type System

Let an **effect** of a process be a pair $^{Z}\mathrm{T}$, where T is the type of exchanged messages, and $^{Z} = ^{\vee}$ only if no *in* or *out* capabilities are exercised.

Types and judgments acquire the form:

Message type $W ::= \mathit{Amb}^{Y}[\mathrm{F}] \mid \mathit{Cap}[\mathrm{F}]$

Exchange type $\mathrm{T} ::= \mathit{Shh} \mid (W_1 \times \cdots \times W_k)$

Good expression $\mathrm{E} \vdash \mathrm{M} : W$

Good process $\mathrm{E} \vdash \mathrm{P} : \mathrm{F}$

As before, any state reachable from a good process is a good process.

If $n : \mathit{Amb}^{Y}[F]$ then $\mathit{in}\ n : \mathit{Cap}[\curvearrowright T]$

If $n : \mathit{Amb}^{Y}[F]$ then $\mathit{out}\ n : \mathit{Cap}[\curvearrowright T]$

If $n : \mathit{Amb}^{\circ}[F]$ then $\mathit{open}\ n : \mathit{Cap}[F]$

If $M : W$ then $\langle M \rangle : {}^{Z}W$

If $P : {}^{Z}W$ then $(x{:}W).P : {}^{Z}W$

If $M : \mathit{Amb}^{Y}[F]$ and $P : F$ then $M[P] : F'$

If $M : \mathit{Cap}[F]$ and $P : F$ then $M.P : F$

If $M : \mathit{Cap}[F]$ and $N[P] : F'$ then $\mathit{go}\ M.N[P] : F'$

If $P : F$ then $(\nu n{:}W)P : F$

If $P : F$ and $Q : F$ then $P \mid Q : F$

If $P : F$ then $!P : F$

$\mathbf{0} : F$

Examples of Type Errors

You cannot open a locked ambient:

$$(\nu n{:}\mathit{Amb}^{\bullet}[\mathrm{F}])(n[] \mid \langle n \rangle \mid (x{:}\mathit{Amb}^{\bullet}[\mathrm{F}]).\mathit{open}\ x)$$

You cannot move an immobile ambient once its running:

$$(x{:}\mathit{Amb}^{\mathrm{Y}}[\overset{\vee}{-}\mathrm{T}]).x[\mathit{out}\ m]$$

Example: Encoding Distribution, Again

Assume that $c{:}\mathit{Ch}[\mathit{Node}]$. The translation of $\mathit{thread}[c(x).\mathit{go}\,x]$,

$$(\nu t)t[(\nu s)(\mathit{go}(\mathit{out}\,t.\mathit{in}\,c^b).c^p[(x,x^p).$$
$$\mathit{go}(\mathit{out}\,c^b.\mathit{in}\,t).s[\mathit{open}\,s.\mathit{out}\,a.\mathit{in}\,x.\mathbf{0}]]\mid$$
$$\mathit{open}\,s.s[])]$$

has effect $\overset{\vee}{-}\mathit{Shh}$ assuming that:

$$a : \mathit{Amb}^{\bullet}[\overset{\vee}{-}\mathit{Shh}], \qquad t : \mathit{Amb}^{\bullet}[\curvearrowright \mathit{Shh}],$$
$$c^b : \mathit{Amb}^{\bullet}[\overset{\vee}{-}[\![\mathit{Node}]\!]^b \times [\![\mathit{Node}]\!]^p], \quad s : \mathit{Amb}^{\circ}[\curvearrowright \mathit{Shh}],$$
$$c^p : \mathit{Amb}^{\circ}[\overset{\vee}{-}[\![\mathit{Node}]\!]^b \times [\![\mathit{Node}]\!]^p]$$

Ambient Types III:

Ambient Groups

Motivating Ambient Groups

We may wish to express that an ambient n can enter the ambient m.

This might be formalised as a property $\mathrm{n} : \mathit{CanEnter}(\mathrm{m})$. But this would divert us into the realm of dependent types.

Instead, we introduce type-level groups of names G, H, and formalise this property as:

> The name n belongs to group G; the name m belongs to group H. Any ambient of group G can enter any ambient of group H.

Generalizing Locking and Immobility Annotations

We decorate an ambient type with its group G, the set **G** of groups it may cross once its running, the set **H** of groups it may open, and the type T of exchanges within it:

$$\mathrm{G}[\curvearrowright\mathbf{G}, {}^{\circ}\mathbf{H}, \mathrm{T}]$$

Moreover, a new operation, $(\nu \mathrm{G})\mathrm{P}$, creates a new group G. Within P, new names of group G can be created. In a well-typed situation, scoping rules dictate that such names may only be handled within P.

Adding Groups to the Type System

Types and judgments acquire the form:

Effect $\mathrm{F} ::= \overset{\frown}{\mathbf{G}}, {}^{\circ}\mathbf{G}, \mathrm{T}$ where $\mathbf{G} ::= \{\mathrm{G}_1, \ldots, \mathrm{G}_n\}$

Message type $W ::= \mathrm{G}[\mathrm{F}] \mid \mathit{Cap}[\mathrm{F}]$

Exchange type $\mathrm{T} ::= \mathit{Shh} \mid (W_1 \times \cdots \times W_k)$

Good expression $\mathrm{E} \vdash \mathrm{M} : W$

Good process $\mathrm{E} \vdash \mathrm{P} : \mathrm{F}$

As before, any state reachable from a good process is a good process.

The effect of a good process is an upper bound on the ambients it may cross or open, and the messages it may exchange.

If $n : G[F]$ and $G \in \mathbf{G}$ then *in* $n : \mathit{Cap}[\curvearrowright\mathbf{G}, \circ\mathbf{H}, T]$

If $n : G[F]$ and $G \in \mathbf{G}$ then *out* $n : \mathit{Cap}[\curvearrowright\mathbf{G}, \circ\mathbf{H}, T]$

If $n : G[\curvearrowright\mathbf{G}, \circ\mathbf{H}, T]$ and $G \in \mathbf{H}$ then *open* $n : \mathit{Cap}[\curvearrowright\mathbf{G}, \circ\mathbf{H}, T]$

If $M : W$ then $\langle M \rangle : \curvearrowright\mathbf{G}, \circ\mathbf{H}, W$

If $P : \curvearrowright\mathbf{G}, \circ\mathbf{H}, W$ then $(x{:}W).P : \curvearrowright\mathbf{G}, \circ\mathbf{H}, W$

If $M : \mathit{Amb}[F]$ and $P : F$ then $M[P] : F'$

If $M : \mathit{Cap}[F]$ and $P : F$ then $M.P : F$

If $M : \mathit{Cap}[F]$ and $N[P] : F'$ then *go* $M.N[P] : F'$

If $P : F$ then $(\nu n{:}W)P : F$

If $P : F$ and $Q : F$ then $P \mid Q : F$

If $P : F$ then $!P : F$

$\mathbf{0} : F$

Example: Encoding Distribution, with Groups

Assume that $c{:}\mathit{Ch}[\mathit{Node}]$. The translation of $\mathit{thread}[c(x).\mathit{go}\,x]$,

$$(\nu \mathit{Sync})(\nu t)\,t[(\nu s)(\mathit{go}(\mathit{out}\,t.\mathit{in}\,c^b).c^p[(x, x^p).$$
$$\mathit{go}(\mathit{out}\,c^b.\mathit{in}\,t).s[\mathit{open}\,s.\mathit{out}\,a.\mathit{in}\,x.\mathbf{0}]] \mid$$
$$\mathit{open}\,s.s[])]$$

has effect $\curvearrowright\emptyset, {}^\circ\emptyset, \mathit{Shh}$ assuming that:

$a : \mathit{Node}[\curvearrowright\emptyset, {}^\circ\mathit{Aux}, \mathit{Shh}]$, $\quad t : \mathit{Thr}[\curvearrowright\mathit{Node}, {}^\circ\mathit{Sync}, \mathit{Shh}]$,

$c^b : \mathit{Ch}[\curvearrowright\emptyset, {}^\circ\mathit{Pkt}, [\![\mathit{Node}]\!]^b \times [\![\mathit{Node}]\!]^p]$, $\quad s : \mathit{Sync}[\curvearrowright\mathit{Node}, {}^\circ\mathit{Sync}, \mathit{Shh}]$,

$c^p : \mathit{Pkt}[\curvearrowright\emptyset, {}^\circ\mathit{Pkt}, [\![\mathit{Node}]\!]^b \times [\![\mathit{Node}]\!]^p]$

Conclusions,

Related Work

Related Work

Several process calculi model distribution and mobility (Boudol; Amadio and Prasad; Hennessy and Riely; Sewell; Fournet, Gonthier, and Lévy).

Zimmer has proposed algorithms for our system with mobility and locking annotations. Few other type systems regulate process mobility.

The idea of groups is related to Milner's sorts for π, to channels and binders found in flow analyses for π, and to the regions used for memory management in ML.

Theory of Ambients

Untyped ambient calculus (Cardelli and Gordon, FoSSaCS'98)

Abstractions for mobile computation (Cardelli, ICALP'99)

Equational properties (Gordon and Cardelli, FoSSaCS'99)

Safe ambients (Levi and Sangiorgi, POPL'00)

Modal logics (Cardelli and Gordon, POPL'00)

Exchange types (Cardelli and Gordon, POPL'99)

Mobility types (Cardelli, Ghelli, and Gordon, ICALP'99)

Subtyping and algorithms for mobility types (Zimmer, dissertation)

Ambient groups (Cardelli, Ghelli, and Gordon, submitted)

Implementations of Ambients

Ambit applet (Cardelli)

Ambient language design (Cardelli and Torgersen)

Ambients in Jocaml (Fournet, Lévy, Schmitt)

Reactive ambients (Sangiorgi and Boussinot)

Ambients in Haskell (Peyton Jones)

Model checker for the logic (Gordon)

Summary

A goal of our calculus is to prototype a flexible, precise, secure, and typeful programming model for mobile software components.

Types regulate aspects of mobile computation such as exchanging messages and exercising capabilities for mobility.

Type systems like these could be checked by a bytecode verifier to better constrain mobile code.

An intriguing possibility: typings for XML. . .

Papers and software available from:

http://www.luca.demon.co.uk/Ambit/Ambit.html

http://research.microsoft.com/users/adg/Publications

Exiting an Ambient

The capability *out* A allows the ambient *msg* to exit the ambient A:

$$\mathrm{A}[\mathit{msg}[\mathit{out}\ \mathrm{A}.\mathit{in}\ \mathrm{B} \mid \langle \mathrm{M} \rangle]]$$
$$\rightarrow\ \ \mathrm{A}[] \mid \mathit{msg}[\mathit{in}\ \mathrm{B} \mid \langle \mathrm{M} \rangle]$$

Ambient *msg* is the unit of mobility, which crosses the perimeter A.

Entering an Ambient

The capability *in* B allows the ambient *msg* to enter the ambient B:

$$\mathit{msg}[\mathit{in}\ \mathrm{B} \mid \langle \mathrm{M} \rangle] \mid \mathrm{B}[\mathit{open}\ \mathit{msg}.(\mathrm{x}).\mathrm{P}]$$

$$\rightarrow\ \ \mathrm{B}[\mathit{msg}[\langle \mathrm{M} \rangle] \mid \mathit{open}\ \mathit{msg}.(\mathrm{x}).\mathrm{P}]$$

Ambient *msg* is the unit of mobility, which crosses the perimeter B.

Opening an Ambient

The capability *open msg* dissolves the boundary around ambient *msg*:

$$msg[\langle M \rangle] \mid \textit{open}\ msg.(x).P$$
$$\rightarrow \quad \langle M \rangle \mid (x).P$$

The ambient *msg* is the unit of mobility in that as its perimeter is breached, its subprocesses become subprocesses of the top-level.

Exchanging a Message

If there is no intervening boundary, messages may be exchanged:

$$\langle M \rangle \mid (x).P \;\rightarrow\; P\{x \leftarrow M\}$$

In the processes below, the boundary n prevents exchange of M:

$$n[\langle M \rangle] \mid (x).P$$

$$\langle M \rangle \mid n[(x).P]$$