Types for Mobile Ambients

Andy Gordon, Microsoft Research

(based on joint work with Luca Cardelli, Microsoft Research)
Ambients describe the mobility of software and hardware, including the passage across administrative boundaries.

Ambients are named, bounded places where computation happens.

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

One goal of this work is to develop a flexible, precise, secure, and typeful programming model for mobile software components.
Mobile Ambients: a packet from $A$ to $B$

Machine $A$

$A[\text{msg}[\text{out } A.\text{in } B \mid \langle M \rangle]] \mid B[\text{open } \text{msg.}(x).P]$

$A \rightarrow B : M$

Machine $B$

receive $x; P$

- 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 $\text{out } A$ to exit $A$; you need capability $\text{in } B$ to enter $B$
There are four basic reduction rules in the calculus:

\[
A[\text{msg[ out } A . \text{in } B \mid \langle M \rangle \rangle] \mid B[\text{open msg.}(x).P]
\]

\[\rightarrow A[] \mid \text{msg[ in } B \mid \langle M \rangle \rangle \mid B[\text{open msg.}(x).P]\]

\[\rightarrow A[] \mid B[\text{msg}[\langle M \rangle] \mid \text{open msg.}(x).P]\]

\[\rightarrow A[] \mid B[\langle M \rangle \mid (x).P]\]

\[\rightarrow A[] \mid B[P[x \leftarrow M]]\]
**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

- \( (\forall n)P \) restriction
- \( 0 \) inactivity
- \( P \parallel Q \) composition
- \( !P \) replication
- \( M[P] \) ambient
- \( M.P \) action
- \( (n_1, \ldots, n_k).P \) input action
- \( \langle M_1, \ldots, M_k \rangle \) async output action
Subjective versus Objective Moves

We base ambient mobility on subjective moves:

\[ n[in \ M \ Q] \ | \ m[R] \rightarrow m[n[P \ Q] \ | \ R] \]
\[ m[n[out \ M \ P] \ | \ Q] \ | \ R \rightarrow n[P \ Q] \ | \ m[R] \]

Instead, we might have adopted primitives for objective moves:

\[ mv \ in \ n.P \ | \ n[Q] \rightarrow n[P \ Q] \]
\[ n[mv \ out \ n.P \ | \ Q] \rightarrow P \ | \ n[Q] \]

But objective moves only move still ambients, and they allow kidnap:

\[ m[P] \ | \ (\forall k)(k[] \ | \ mv \ in \ m.in \ k) \rightarrow^* (\forall k)k[m[P]] \]
The special case of objective movement of an ambient is safe, convenient, and derivable from subjective movement:

\[\text{move } M.n[P] \overset{\triangle}{=} (\forall k)k[M.n[\text{out } k.P]] \quad \text{for } k \text{ not free in } M.P\]

For example:

\[\text{move in } m.n[P] \mid m[R] \rightarrow (\forall k)m[k[n[\text{out } k.P]] \mid R]\]
\[\rightarrow (\forall k)m[k] \mid n[P] \mid R]\]
\[\simeq m[n[P] \mid R]\]

The relation \(\simeq\) is a semantic equivalence used for garbage collection.
Regulating Input/Output
Related Work

Our work borrows ideas from previous treatments of types for the \( \lambda \)-calculus and the \( \pi \)-calculus.

In particular, Milner’s sorts for \( \pi \), Pierce and Sangiorgi’s type system for \( \pi \), and Kobayashi, Pierce, and Turner’s linear type system for \( \pi \).

Some quite sophisticated type systems are being investigated for mobile computation, e.g., by Sewell, Hennessy and Riely, Jeffrey, …

Our approach is to investigate several simple systems that regulate exchanges, mobility, security levels, etc., and attempt to integrate them into a coherent whole.
In the untyped calculus, certain processes arise that make no sense:

- Process $in \, \pi[P]$ uses a capability as an ambient name
- Process $\pi.P \mid \pi[Q]$ 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 $(x : W).P : W$. 
Process 0 exchanges messages of any type, since it exchanges none.

\[ 0 : T \text{ for all } T. \]

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

\[ \text{If } P : T \text{ and } Q : T \text{ then } P \mid Q : T. \]
\[ \text{If } P : T \text{ then } !P : T. \]

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

An expression of type $\text{Amb}[T]$ 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 $M[P]$ is correctly typed.

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

An ambient exchanges no messages, so it may be assigned any type.
An expression of type $\text{Cap}[T]$ is a capability that may unleash exchanges of type $T$.

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

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

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

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

If $n : \text{Amb}[S]$ then $\text{in } n : \text{Cap}[T]$ for all $T$.
If $n : \text{Amb}[S]$ then $\text{out } n : \text{Cap}[T]$ for all $T$. 
Types: 

\[ W ::= \]

- \( Amb[T] \) : ambient name allowing \( T \) exchanges
- \( Cap[T] \) : capability unleashing \( T \) exchanges

\[ S, T ::= \]

- \( Shh \) : no exchange
- \( W_1 \times \cdots \times W_k \) : tuple exchange

- A quiet ambient, \( Amb[Shh] \), and a harmless capability, \( Cap[Shh] \)
- An ambient allowing exchange of harmless capabilities: \( Amb[Cap[Shh]] \)
- A capability unleashing exchanges of names of quiet ambients: \( Cap[Amb[Shh]] \)
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 $\text{in } n[P]$ and $n.P \mid n[Q]$ cannot arise during a computation, since they are not typeable.
Examples of Typing

Packet from $A$ to $B$:


$$\text{Cap}[W] \quad \text{Cap}[W]$$

Objective ambient move:

If $M : Cap[T]$ and $n[P] : S$ then $move$ $M.n[P] : S.$
Typed Semantics of a Distributed Language
There is a flat collection of named nodes (or locations), each of which contains a group of named channels and anonymous threads:

\[
\begin{align*}
\text{node } A & \mid \text{thread}[\text{go } B.b(\text{node, ch}).\text{go } \text{node.ch}(A)] \mid \\
\text{node } B & \mid \text{channel } b \mid \text{thread}[b(C, c)] \mid \\
\text{node } C & \mid \text{channel } c
\end{align*}
\]

Heterogeneous models like this underly several distributed programming systems, and several distributed forms of the $\pi$-calculus.
A fragment of a typed, distributed programming language:

\[ Net ::= \]
\[ (\forall n: Ty) Net \] network
\[ Net \mid Net \] restriction
\[ node \ n \ [Grp] \] composition of networks
\[ Grp ::= \]
\[ (\forall n: Ty) Grp \] node
\[ Grp \mid Grp \] group of channels and threads
\[ channel \ c \] restriction
\[ thread[Th] \] composition of groups
\[ Th ::= \]
\[ go \ n.\ Th \] channel
\[ c(x_1: Ty_1, \ldots, x_k: Ty_k).\ Th \] thread
\[ fork(Grp).\ Th \] migration
\[ \overline{c}(n_1, \ldots, n_k) \] output to a channel
\[ c(x_1: Ty_1, \ldots, x_k: Ty_k).\ Th \] input from a channel
\[ fork(Grp).\ Th \] fork a group
Much as in distributed forms of $\pi$, each name has a type, $\bar{Ty}$.

**Types of names:**

\[ Ty ::= \]

- **Node**
  - name of a node
- \( Ch[\bar{Ty_1}, \ldots, \bar{Ty}_k] \)
  - name of a channel

**Translation of a type $\bar{Ty}$, $\llbracket \bar{Ty} \rrbracket$**

\[
\llbracket Node \rrbracket \triangleq Amb[Shh]
\]
\[
\llbracket Ch[\bar{Ty_1}, \ldots, \bar{Ty}_k] \rrbracket \triangleq Amb[\llbracket \bar{Ty_1} \rrbracket \times \cdots \times \llbracket \bar{Ty}_k \rrbracket]
\]
Translation of a network $Net, [[Net]]$

\[
\begin{align*}
[[\forall n:Ty)Net] & \triangleq (\forall n:[Ty])[Net] \\
[Net \mid Net] & \triangleq [[Net] \mid [[Net]] \\
[[node n [Grp]] & \triangleq n[!open n \mid [[Grp]]_n]
\end{align*}
\]

Translation of a group $Grp$ located at $n, [[Grp]]_n$

\[
\begin{align*}
[[\forall m:Ty)Grp]_n & \triangleq (\forall m:[Ty])[Grp]_n \text{ for } m \neq n \\
[Grp \mid Grp]_n & \triangleq [[Grp]_n \mid [[Grp]]_n \\
[[channel c]]_n & \triangleq c[!open c] \\
[[thread Th]]_n & \triangleq (\forall t:Amb[Shh])t[[Th]]_n \text{ for } t \neq n \text{ not free in } Th
\end{align*}
\]
Translation of a thread \(Th\) named \(t\) located at \(n\), \(\llbracket Th \rrbracket^t_n\):

\[
\begin{align*}
\llbracket go \ m. Th \rrbracket^t_n & \triangleq out \ n. in \ m. \llbracket Th \rrbracket^t_m \\
\llbracket C\langle n_1, \ldots, n_k \rangle \rrbracket^t_n & \triangleq move (out t. in c). c[\langle n_1, \ldots, n_k \rangle] \\
\llbracket C(x_1:T_{y_1}, \ldots, x_k:T_{y_k}). Th \rrbracket^t_n & \triangleq (\forall s:Amb[Shh]) \\
& \quad (move (out t. in c). \\
& \quad \quad c[(x_1:T_{y_1}), \ldots, x_k:T_{y_k}]). \\
& \quad move (out c. in t). s[\llbracket Th \rrbracket^t_n] | \\
& \quad open s) \\
\llbracket fork(Grp). Th \rrbracket^t_n & \triangleq (\forall m:Amb[Shh]) \\
& \quad (move out t.n[open m.\llbracket Grp \rrbracket_n] | \\
& \quad move out t.m[move in t.t] | \\
& \quad open t.\llbracket Th \rrbracket^t_n) 
\end{align*}
\]
Exchange Types Support a Typed Semantics

Let $\llbracket E \rrbracket \triangleq n_1 : \llbracket Ty_1 \rrbracket, \ldots, n_k : \llbracket Ty_k \rrbracket$ if $E = n_1 : Ty_1, \ldots, n_k : Ty_k$.

1. If $E \vdash Net$ then $\llbracket E \rrbracket \vdash \llbracket Net \rrbracket : Shh$.

2. If $E \vdash Grp$ and $E \vdash n : Node$ then $\llbracket E \rrbracket \vdash \llbracket Grp \rrbracket_n : Shh$.

3. If $E \vdash Th$ and $E \vdash n : Node$ and $t \notin \text{dom}(E)$, then $\llbracket E \rrbracket, t : Amb[Shh] \vdash \llbracket Th \rrbracket_t^n : Shh$. 
Regulating Mobility
Motivation for Mobility Types

Although in general ambients should be mobile, some important classes of ambients (like network nodes or channels) should be immobile.

We annotate the syntax of ambients to draw the distinction:

- $n^P$ may move, whereas
- $n^x P$ may not.

The purpose of the mobility type system is to rule out execution errors such as $n^x [in m]$ or $n^x [out m]$. 
Extending the System of Exchange Types

**Types:**

\[
Z ::= \checkmark \mid \times \quad \text{mobility annotations}
\]

\[
W ::= \text{Amb}^Z[T] \mid \text{Cap}^Z[T] \quad \text{message types}
\]

\[
T ::= Shh \mid W_1 \times \cdots \times W_k \quad \text{exchange types}
\]

- A quiet immobile ambient, \(\text{Amb}^\times[Shh]\).
- A quiet mobile ambient \(\text{Amb}^\checkmark[Shh]\).
- A capability unleashing no mobility effects and no exchanges \(\text{Cap}^\times[Shh]\).
- A capability that may unleash mobility effects, but no exchanges, \(\text{Cap}^\checkmark[Shh]\).
Judgments:

\[ E \vdash M : W \quad \text{good expression of message type } W \]
\[ E \vdash P : Z \quad T \quad \text{process with mobility } Z \text{ exchanging } T \]

Good expressions:

\[ E \vdash n : Amb^Z[T] \]
\[ E \vdash \text{in } n : \text{Cap}^\check{[T']} \]
\[ E \vdash \text{out } n : \text{Cap}^\check{[T']} \]
\[ E \vdash \text{open } n : \text{Cap}^Z[T] \]

\[ E \vdash \diamond \]
\[ E \vdash e : \text{Cap}^Z[T] \]
\[ E \vdash M : \text{Cap}^Z[T] \]
\[ E \vdash M' : \text{Cap}^Z[T] \]
\[ E \vdash M.M' : \text{Cap}^Z[T] \]

A capability for moving a mobile ambient in and out of an immobile ambient:

\[ n : Amb^x[Shh] \vdash \text{in } n. \text{out } n : \text{Cap}^\check{[T]} \text{ for all } T. \]
Good processes:

\[
\begin{align*}
E \vdash M : Amb^Z[T] & \quad \quad \quad E \vdash P : Z \ T \\
E \vdash M^Z[P] : Z' \ T' & \quad \quad \quad E \vdash M \cdot P : Z \ T \\
E \vdash P : Z \ T \quad E \vdash Q : Z \ T & \quad \quad \quad E, p : Amb^Z[T] \vdash P : Z' \ T' \\
E \vdash P \mid Q : Z \ T & \quad \quad \quad E \vdash (\forall p)P : Z' \ T' \\
E, n_1 : W_1, \ldots, n_k : W_k \vdash P : Z \ W_1 \times \cdots \times W_k & \quad \quad \quad E \vdash 0 : Z \ T \\
E \vdash (n_1, \ldots, n_k).P : Z \ W_1 \times \cdots \times W_k & \\
E \vdash M_1 : W_1 \quad \cdots \quad E \vdash M_k : W_k & \quad \quad \quad E \vdash \langle M_1, \ldots, M_k \rangle : Z \ W_1 \times \cdots \times W_k
\end{align*}
\]
This basic system is sound, but immobile ambients cannot easily interact with mobile ambients.

For example, we cannot type a packet $p$ being sent to an immobile ambient $q$, whether we make the packet $\checkmark$ or $\times$:

- $p^\checkmark[\text{in } q.\langle M \rangle] \mid q^\times[\text{open } p.(x).R]$
- $p^\times[\text{in } q.\langle M \rangle] \mid q^\times[\text{open } p.(x).R]$

But, in fact, no execution error can arise in either case.
Objective Ambient Movement as Primitive

We add \( \text{move} \ M \cdot N \rightarrow^Z [P] \) to the syntax of typed processes.

The reduction rules are derived from the untyped encoding \((\forall k) k[M \cdot N [\text{out} \ k.P]]\). For example:

\[
\text{move in} \ q.p^x[\langle M \rangle \mid q^x[\text{open} \ p.(x).R] \rightarrow q^x[p^x[\langle M \rangle \mid \text{open} \ p.(x).R]
\]

An objective ambient move exchanges no messages, and has no mobility effects, so we have:

If \( M : \text{Cap}^x[S] \) and \( N : \text{Amb}^Z[T] \) and \( P : ^Z T \)

then \( \text{move} \ M \cdot N \rightarrow^Z [P] : ^Z' T' \).

A subject reduction theorem holds for this extended system.
Threads are Mobile; Nodes, Channels are Immobile

A more accurate typed semantics of the distributed language:

\[
\begin{align*}
\llbracket \text{Node} \rrbracket & \triangleq \text{Amb}^\times [\text{Shh}] \\
\llbracket \text{Ch}[T_1, \ldots, T_k] \rrbracket & \triangleq \text{Amb}^\times \llbracket [T_1] \times \cdots \times [T_k] \rrbracket \\
\llbracket \text{node } n \llbracket \text{Grp} \rrbracket & \triangleq n^\times [\text{open } n \mid \llbracket \text{Grp} \rrbracket_n] \\
\llbracket \text{channel } c \rrbracket_n & \triangleq c^\times [\text{open } c] \\
\llbracket \text{thread } Th \rrbracket_n & \triangleq (\forall t: \text{Amb}^\checkmark [\text{Shh}]) t^\checkmark [\llbracket Th \rrbracket_n^t]
\end{align*}
\]

If \( E \vdash \text{Net} \) then \( \llbracket E \rrbracket \vdash \llbracket \text{Net} \rrbracket :^\times \text{Shh} \).

If \( E \vdash \text{Grp} \) and \( E \vdash n : \text{Node} \) then \( \llbracket E \rrbracket \vdash \llbracket \text{Grp} \rrbracket_n :^\times \text{Shh} \).

If \( E \vdash Th, E \vdash n : \text{Node}, t \not\in \text{dom}(E) \) then \( \llbracket E \rrbracket, t: \text{Amb}^\checkmark [\text{Shh}] \vdash \llbracket Th \rrbracket_n^t :^\checkmark \text{Shh} \).
A goal of developing 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.

Hence, the ambient calculus serves as a typed metalanguage for describing mobile computation.