EXPRESS 2001 Preliminary Version

Encoding Distributed Areas and Local Communication into the π-Calculus

Tom Chothia [1] and Ian Stark

Laboratory for Foundations of Computer Science
Division of Informatics, The University of Edinburgh
Mayfield Road, Edinburgh EH9 3JZ, Scotland, UK
{stark,tpcc}@dcs.ed.ac.uk

Abstract

We show how the π-calculus can express local communications within a distributed system, through an encoding of the *local area π-calculus*, an enriched system that explicitly represents names which are known universally but always refer to local information. Our translation replaces point-to-point communication with a system of shared local ethers; we prove that this preserves and reflects process behaviour. We give an example based on an internet service dæmon, and investigate some limitations of the encoding.

1 Introduction

Part of the power of the π-calculus is that names serve a dual rôle: as well as carriers of communication, they have unique *identity*. By default the scope of these coincide, so that any two processes that know a common name can also use it to communicate. In many distributed systems, however, this is not so natural: widely-known names may be intended to refer always to local information. For example, the standard `finger` service operates on well-known port number 79, but should of course give a different answer on different machines. We can even take this as a defining characteristic of a distributed system: that a single name may refer to different things depending on where it is used.

The *local area π-calculus* ($la\pi$) captures this phenomenon of names which are known universally but always refer to local information. It extends the π-calculus so that a channel name can have within its scope several disjoint *local areas*. Such a channel name may be used for communication within an area, it may be sent between areas, but it cannot itself be used to transmit

[1] Supported by the UK Engineering and Physical Sciences Research Council

This is a preliminary version. The final version will be published in Electronic Notes in Theoretical Computer Science
URL: www.elsevier.nl/locate/entcs

information from one area to another. Areas are arranged in a hierarchy of *levels*, distinguishing for example between a single application, a machine, or a whole network.

In previous work we introduced the local area π-calculus and showed how a combination of static typing and dynamic checks can give flexible scope for name identity while suitably restricting communication to local areas [4]. In this paper we give a compositional translation of $la\pi$ into the plain π-calculus, and prove that it correctly encodes process behaviour.

The main challenge for representing local areas in the π-calculus is how to prevent communication on a channel between different areas, while still preserving the identity of names. Our solution is to replace communication inside an area with communication on a new channel created just for that area. As well as sending the original data, an encoded output sends the channel name as well. Output action $\bar{a}\langle b\rangle$ becomes $\bar{e}\langle a, b\rangle$ where e is a name that corresponds to the appropriate local area — a shared *ether*. An input action on a channel translates to a process that listens for communication on the relevant ether. When it receives a message it tests the first element against the desired channel name; if they match it accepts the input, otherwise it rebroadcasts the message.

This translation makes explicit the different rôles of names, identity and communication, by mapping them to two distinct sets of names. Names like a and b serve purely as data, for identification; ether names like e are for communication only. The scope of data names manages who knows what, while the scope of ether names handles locality of communication.

The evident motivation for this model is packet communication on an ethernet: instead of sending data directly to its destination, we drop a packet into the ether. Listening processes pick up all packets and sift out the ones they are interested in. In our case the tree of nested areas (applications, machines, networks) gives rise to a hierarchy of ethers, with a process using a different ether for each level of communication (local to an application; within a machine; over the network).

There is a close match between the behaviour of a process in $la\pi$ and its π-calculus translation; we show a form of weak bisimilarity on outputs. There is some loss of information though, in that translated terms may make additional silent moves, as packets pass over the ether, and an ether may indiscriminately accept and rebroadcast packets in which no receiver is interested.

The rest of the paper is arranged as follows. Section 2 presents the local area π-calculus, its type system and operational semantics. Section 3 reviews the version of the π-calculus we use. Section 4 gives the encoding between these, and Section 5 outlines the result that a process and its encoding are weakly bisimilar on output. Section 6 presents an example of the translation at work, and Section 7 concludes.

Related work

Local areas are in some sense a more regular form of CCS *restriction* or CHOCS *blocking*. Vivas and Dam have studied the effect of these on the higher-order π-calculus, and given an encoding into the π-calculus [14]. However, their encoding relies on blocking being carried out explicitly on individual names, and is (in their own words) both complex and indirect (though chiefly because it also handles higher-order operations).

Cardelli, Ghelli and Gordon take a different approach to limiting communication with their notion of name *groups* [**?**]. Ingeniously, introducing these into the type system allows one to check statically that a process never passes out certain names. In our system, by contrast, names may be passed anywhere — only their action is limited.

There are numerous projects addressing *locations* in the π-calculus [**?**,**?**,8], and distributed systems more generally [**?**,6,**?**,12]. These overlap with our approach to varying degrees; some look at issues of when locations fail, others limit communication in particular ways. In the extreme, systems like mobile ambients make all interaction local: remote agents must move around to talk to each other [3].

Translating miscellaneous concurrent systems into each other is also a popular sport. Nestmann and Pierce give a good overview of what makes for a good encoding in their work on deconstructing choice [**?**]. Among many examples, Fournet and others have implemented mobile ambients in the JoCaml languages [**?**]; this translates one notion of distributed areas into another, whereas we make areas disappear entirely. Moreover, their focus is on providing a basis for a implementation of Mobile Ambients, so much attention is paid to making it run efficiently. Sangiorgi describes in great detail a rather different encoding of locations in order to express non-interleaving semantics [11].

2 A π-calculus with local areas

The local area π-calculus extends a standard π-calculus with nested *local areas* arranged in *levels*. The π-calculus part is unexceptional: it happens to be polyadic (channels carry tuples rather than single values [9]) and asynchronous (output actions always succeed [1]). To illustrate the extensions, we present a brief example, based on a mechanism for selecting internet services.

When a browser contacts a web server to fetch a page, or a person operates `finger` to list the users on another machine, both connect to a numbered "port" on the remote host: port 80 for the web page, port 79 for the finger listing. Of course, this only works if both sides agree; and there is a real-world committee to set this up [7]. Under Unix, the file `/etc/services` holds a list mapping numbers to services. There is also a further level of indirection: most machines run only a general meta-server `inetd`, the Internet dæmon, which listens on all ports. When `inetd` receives a connection, it looks up the port in `/etc/services`, and then consults a second file which identifies the program to provide that service. The `inetd` starts the program and hands it a connection to the caller. A model of this procedure in the local area π-calculus looks like this (we omit detailed type information).

$$\begin{array}{rl}
\text{Client} & \mathit{Carp} = \mathit{host}[\,\nu c.(\overline{\mathit{pike}}\langle \mathit{finger}, c\rangle \mid c(x).\overline{\mathit{print}}\langle x\rangle)\,] \\
\text{Server} & \mathit{Pike} = \mathit{host}[\,!\mathit{pike}(s,r).\bar{s}\langle r\rangle \mid !\mathit{finger}(y).\bar{y}\langle \mathit{PikeUsers}\rangle \\
& \qquad\qquad\qquad\qquad\qquad \mid !\mathit{daytime}(z).\bar{z}\langle \mathit{PikeDate}\rangle\,] \\
\text{System} & \mathit{net}[\mathit{Carp} \mid \mathit{Pike}] \\
\text{Names} & \mathit{pike} \text{ and } c \text{ operate at } \mathit{net} \text{ level;} \\
& \mathit{finger}, \mathit{daytime} \text{ and } \mathit{print} \text{ operate at } \mathit{host} \text{ level}
\end{array}$$

This shows a client machine *Carp* that wishes to contact a server *Pike* with a finger request; the $\mathit{host}[-]$ and $\mathit{net}[-]$ markers indicate local areas. The client has two components: the first transmits the request, the second prepares to print the result. Server *Pike* comprises three replicating processes: a general Internet dæmon, a Finger dæmon, and a time-of-day dæmon. Channel *pike* is the internet address of the server machine, while the free names *finger* and *daytime* represent well-known port numbers. In operation, *Carp* sends its request to *Pike* naming the finger service and a reply channel c. The Internet dæmon on *Pike* handles this by retransmitting the contact c over the channel named *finger*. The Finger dæmon collects this and passes information on *PikeUsers* back to the waiting process at *Carp*. Figure 1 gives a graphical representation of the interaction.

In the plain π-calculus, this models leaks: because the names *finger* and *daytime* are visible everywhere, even when the Internet dæmon on *Pike* has collected the request there is no protection against a Finger dæmon on some different server actually handling it. Restricting the scope of *finger* to host *Pike* would be no solution, because then *Carp* could not formulate the request because it has to know the name of the service.

In the local area π-calculus, each channel has an assigned level of operation, which limits how far communication on that channel may travel. In this case, although *finger* is globally known, messages over it remain within a single *host*. This breaks the Catch-22: *Carp* and *Pike* agree on the name for the *finger* service, but different Finger dæmons on separate machines do not interfere with each other.

2.1 Syntax

The calculus is built around two classes of identifiers:

$$\begin{array}{rr}
\text{channels} & a, b, c, x, y, \mathit{query}, \mathit{reply}, \ldots \in \mathit{Chan} \\
\text{and levels} & \ell, m, \mathit{app}, \mathit{host}, \mathit{net}, \ldots \in \mathit{Level}.
\end{array}$$

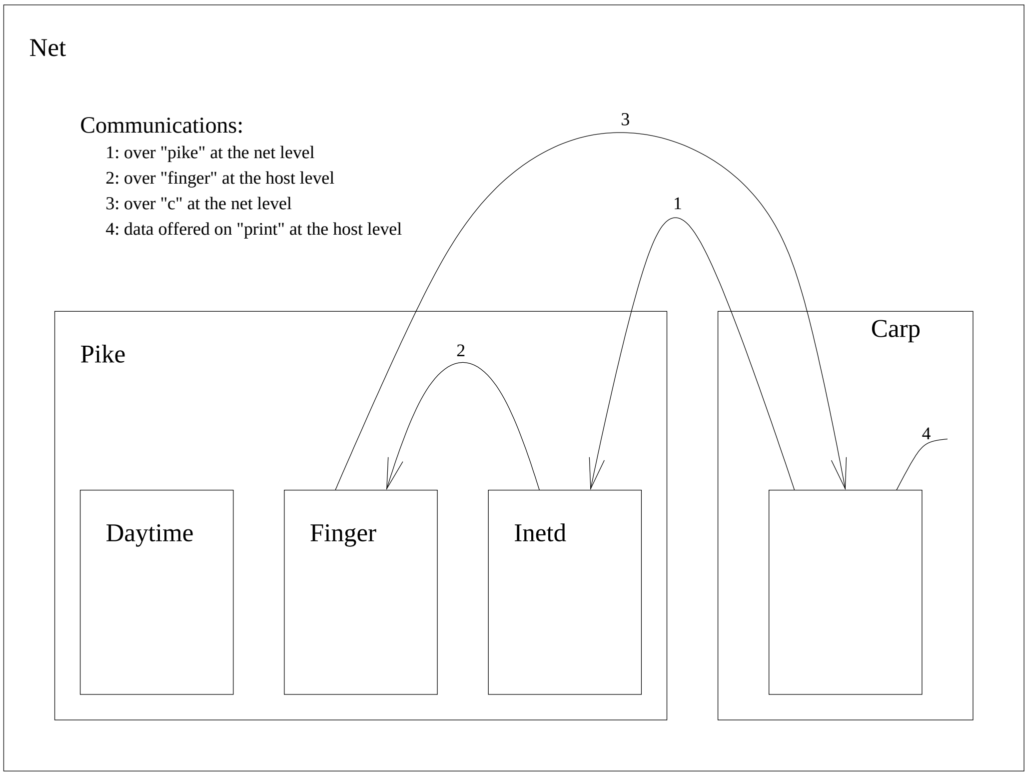

Fig. 1. Operating a remote `finger` service through an `inet` dæmon

Channel names are drawn from a countably infinite supply, $Chan$. Syntactically, they behave exactly as in the π-calculus. Levels are rather more constrained: we assume prior choice of some finite and totally ordered set $Level$. Throughout the paper we use $app < host < net$, and take ℓ and m as metavariables for levels.

Processes are given by the following syntax, based on the asynchronous polyadic π-calculus.

Process $P, Q ::= 0$	inactive process	$\mid\ \bar{a}\langle\vec{b}\rangle$	output tuple
$\mid\ P \mid Q$	parallel composition	$\mid\ a(\vec{b}).P$	input
$\mid\ \ell[P]$	local area at level ℓ	$\mid\ !a(\vec{b}).P$	replicated input
$\mid\ \nu a{:}\sigma.P$	fresh channel a of type σ		

The only novelty here is $\ell[P]$, which represents a process P running in a local area at level ℓ; we refer to a process of this form as an *agent*. Areas, like processes, are anonymous; this is in contrast to systems for locations, which are usually tagged with identifiers.

Channel names may be bound or free in any process. The binding prefixes are as usual the input prefixes $a(\vec{b})$, $!a(\vec{b})$ and restriction $\nu a{:}\sigma$; the type σ gives information about the level of operation of a and the tuples it carries. We write $fn(P)$ for the set of free names of process P.

We identify process terms up to *structural congruence* '$\equiv$', the smallest congruence relation containing the following equations:

$$
\begin{array}{ccc}
P \,|\, 0 \equiv P & a(\vec{b}).P \equiv a(\vec{c}).P\{\vec{c}/\vec{b}\} & \vec{c} \cap fn(P) = \emptyset \\
P \,|\, Q \equiv Q \,|\, P & !a(\vec{b}).P \equiv !a(\vec{c}).P\{\vec{c}/\vec{b}\} & \vec{c} \cap fn(P) = \emptyset \\
(P \,|\, Q) \,|\, R \equiv P \,|(Q \,|\, R) & \nu a{:}\sigma.P \equiv \nu b{:}\sigma.P\{b/a\} & b \notin fn(P) \\
\nu a{:}\sigma.0 \equiv 0 & \nu a{:}\sigma.\nu b{:}\tau.P \equiv \nu b{:}\tau.\nu a{:}\sigma.P & a \neq b \\
\ell[\nu a{:}\sigma.P] \equiv \nu a{:}\sigma.(\ell[P]) & (\nu a{:}\sigma.P) \,|\, Q \equiv \nu a{:}\sigma.(P \,|\, Q) & a \notin fn(Q)
\end{array}
$$

2.2 Scope and areas

The interesting equation in this structural congruence is $\ell[\nu a{:}\sigma.P] \equiv \nu a{:}\sigma.(\ell[P])$, commuting name binding and area boundaries. A consequence of this is that the scope of a channel name, determined by ν-binding, is quite independent from the layout of areas, given by $\ell[-]$. Scope determines where a name is known, and this will change as a process evolves: areas determine how a name can be used, and these have a fixed structure.

For a process description to be meaningful, this fixed structure of nested areas must accord with the predetermined ordering of levels. For example, a *net* may contain a *host*, but not vice versa; similarly a *host* cannot contain another *host*. Writing $<_1$ for the one-step relation in the total order of levels, we require that in a well-formed process every nested area must be $<_1$-below the one above.

Consider some occurrence of a bound channel name a in a well-formed process P, as the subject of some action: $\bar{a}\langle - \rangle$, $a(-)$, or $!a(-)$. The *scope* of a is the enclosing ν-binding $\nu a{:}\sigma.(-)$. The *local area* of this occurrence of a is the enclosing level ℓ area $\ell[-]$. A single name may have several disjoint local areas within its scope. It is also possible for a name to occur outside any local area of the right level; in this case it may only be treated as data, not used for communication.

2.3 Type system

Channel types have the following rather simple grammar.

$$\text{Type } \sigma \ ::= \ \vec{\sigma}@\ell$$

A type declaration of the form $a : \vec{\sigma}@\ell$ states that a is a level ℓ channel carrying tuples of values whose types are given by the vector $\vec{\sigma}$. The base types are those with empty tuples: a channel of type $()@\ell$ is for synchronization within an ℓ-area. Additional base datatypes like *int* or *string* can be incorporated without difficulty.

Figure 2 presents the rules for deriving type assertions of the form $\Gamma \vdash_\ell P$, where Γ is a finite map from channel names to types. This states that

$$\Gamma \vdash_\ell 0 \qquad \frac{\Gamma \vdash_\ell P \quad \Gamma \vdash_\ell Q}{\Gamma \vdash_\ell P \,|\, Q} \qquad \frac{\Gamma, \vec{b} : \vec{\sigma} \vdash_\ell P}{\Gamma \vdash_\ell a(\vec{b}).P} \begin{array}{l}\Gamma(a) = \vec{\sigma}@m \\ \text{with } \ell \leq m\end{array}$$

$$\frac{\Gamma \vdash_\ell P}{\Gamma \vdash_m \ell[P]} \ell <_1 m \qquad \frac{\Gamma, a : \sigma \vdash_\ell P}{\Gamma \vdash_\ell \nu a{:}\sigma.P} \qquad \frac{\Gamma, \vec{b} : \vec{\sigma} \vdash_\ell P}{\Gamma \vdash_\ell !a(\vec{b}).P} \begin{array}{l}\Gamma(a) = \vec{\sigma}@m \\ \text{with } \ell \leq m\end{array}$$

$$\Gamma \vdash_\ell \bar{a}\langle\vec{b}\rangle \quad \text{if } \Gamma(a) = \vec{\sigma}@m,\ \Gamma(\vec{b}) = \vec{\sigma} \text{ and } \ell \leq m$$

Fig. 2. Types for processes in the local area calculus

$$\begin{array}{lcl}
\text{OUT} & \Gamma \vdash_\ell \ \bar{a}\langle\vec{b}\rangle \xrightarrow{\bar{a}\langle\vec{b}\rangle} 0 & \\[2ex]
\text{IN} & \Gamma \vdash_\ell \ a(\vec{b}).P \xrightarrow{a(\vec{b})} P & \vec{b} \cap dom(\Gamma) = \emptyset \\[2ex]
\text{IN!} & \Gamma \vdash_\ell \ !a(\vec{b}).P \xrightarrow{a(\vec{b})} P \,|\, !a(\vec{b}).P & \vec{b} \cap dom(\Gamma) = \emptyset \\[2ex]
\text{PAR} & \dfrac{\Gamma \vdash_\ell \ P \xrightarrow{\alpha} P'}{\Gamma \vdash_\ell \ P \,|\, Q \xrightarrow{\alpha} P' \,|\, Q} & \\[3ex]
\text{COMM} & \dfrac{\Gamma \vdash_\ell \ P \xrightarrow{\bar{a}\langle\vec{c}\rangle} P' \quad \Gamma \vdash_\ell \ Q \xrightarrow{a(\vec{b})} Q'}{\Gamma \vdash_\ell \ P \,|\, Q \xrightarrow{\tau} P' \,|\, Q'\{\vec{c}/\vec{b}\}} & \\[3ex]
\text{BIND} & \dfrac{\Gamma, a : \sigma \vdash_\ell \ P \xrightarrow{\alpha} P'}{\Gamma \vdash_\ell \ \nu a{:}\sigma.P \xrightarrow{\alpha} \nu a{:}\sigma.P'} & a \notin fn(\alpha) \\[3ex]
\text{AREA} & \dfrac{\Gamma \vdash_\ell \ P \xrightarrow{\alpha} P'}{\Gamma \vdash_m \ \ell[P] \xrightarrow{\alpha} \ell[P']} & \begin{array}{l}\text{if } \alpha \text{ is } \bar{a}\langle\vec{b}\rangle \text{ or } a(\vec{b}) \\ \text{then } \Gamma(a) = \vec{\sigma}@m' \\ \text{with } \ell <_1 m \leq m'\end{array}
\end{array}$$

Fig. 3. Operational semantics for the local area calculus

process P is well-typed at level ℓ in context Γ. The static checking provided by the type system makes two assurances: tuples sent over channels will always be the right size, and a well-typed process will not attempt to communicate on a name above its level of operation.

2.4 Operational semantics

We give the calculus a late-binding, small-step transition semantics, following the regular π-calculus. There is only one addition: although the static type system guarantees that a process will not initiate communication on a name above its operating level, we still need a dynamic check to make sure that no active communication escapes from its local area.

The operational semantics is given as an inductively defined relation on well-typed processes, indexed by their level ℓ and context Γ. Figure 3 gives rules for deriving transitions of the form

$$\Gamma \vdash_\ell \; P \xrightarrow{\alpha} Q$$

where $\Gamma \vdash_\ell P$ and α is one of the following.

$$\begin{array}{rcll} \text{Transition } \alpha & ::= & \bar{a}\langle\vec{b}\rangle & \text{output} \\ & | & a(\vec{b}) & \text{input} \\ & | & \tau & \text{silent internal action} \end{array}$$

We make a few observations of these rules and the side-conditions attached to them.

- Active use of the structural congruence '$\equiv$' is essential to make full use of the rules: a process term may need to be rearranged or α-converted before it can make progress. For example, there is no symmetric form for the PAR rule (and no need for one).
- In order to apply the COMM rule it may be necessary to use structural congruence to expand the scope of communicated names to cover both sender and recipient.
- Late binding is enforced by the side-condition $\vec{b} \cap dom(\Gamma) = \emptyset$ on the input rules; this ensures that input names are chosen fresh, ready for substitution $Q\{\vec{c}/\vec{b}\}$ in the COMM rule. Again, we can always α-convert our processes to achieve this.
- The side-condition $m \leq m'$ on AREA is the dynamic check that prevents communications escaping from their local area.

In previous work [4] we demonstrated that reduction preserves types, and as a consequence the semantics does successfully capture the intuition behind areas and levels: areas retain their structure over transitions, and actions on a channel are never observed above the correct operating level.

2.5 Example: Internet daemon

Recall the internet service example given earlier: a host *Carp* wishes to contact a *Finger* dæmon running on host *Pike*, through a general *Inet* dæmon. Figure 4 fills out the details of this, including type definitions.

The type *service* for *finger* and *daytime* expands to $(string@net)@host$. This means that these channels can be used only for *host*-level communication, but the values carried will themselves be *net*-level names. The *host*-level communication is between *Inet* and *Finger* or *Daytime*; the *net*-level communication is the response sent out to the original enquirer, in this case over channel c to machine *Carp*. Channel *pike* has a *net*-level type that acts as a gateway to this, reading the name of a service and a channel where that service should send its reply.

$$
\begin{aligned}
Carp &= host[\, \nu c{:}response.(\overline{pike}\langle finger, c\rangle \mid c(x).\overline{print}\langle x\rangle)\,] \\
Pike &= host[\, Inet \mid Finger \mid Daytime\,]
\end{aligned}
$$

$$
\begin{aligned}
Inet &= !pike(s,r).\bar{s}\langle r\rangle \\
Finger &= !finger(y).\bar{y}\langle PikeUsers\rangle \\
Daytime &= !daytime(z).\bar{z}\langle PikeDate\rangle
\end{aligned}
$$

$$
\begin{array}{ll}
\Gamma = \{finger, daytime : service & service = response@host \\
\quad pike : (service, response)@net & response = string@net \\
\quad print : string@host\} &
\end{array}
$$

$$\Gamma \ \vdash_{net} \ (Carp \mid Pike)$$

Fig. 4. Example of processes using local areas: an Internet server dæmon

We can now apply our operational semantics to see this in action.

$$
\begin{array}{rl}
\Gamma \ \vdash_{net} \ (Carp \mid Pike) \equiv & (\ host[\, \nu c{:}response.(\overline{pike}\langle finger, c\rangle \mid c(x).\overline{print}\langle x\rangle)\,] \\
& \mid host[\, Inet \mid Finger \mid Daytime\,]\ ) \\
\text{extend scope of } c \quad & \equiv \nu c{:}response.(\ host[\, \overline{pike}\langle finger, c\rangle \mid c(x).\overline{print}\langle x\rangle\,] \\
& \qquad \mid host[\, Inet \mid Finger \mid Daytime\,]\ ) \\
\text{expand } Inet \quad & \equiv \nu c{:}response.(\ host[\, \overline{pike}\langle finger, c\rangle \mid c(x).\overline{print}\langle x\rangle\,] \\
& \qquad \mid host[\, !pike(s,r).\bar{s}\langle r\rangle \\
& \qquad\qquad \mid Finger \mid Daytime\,]\ ) \\
\text{communication} \quad & \xrightarrow{\tau} \nu c{:}response.(\ host[\, c(x).\overline{print}\langle x\rangle\,] \\
\text{on } pike@net \quad & \qquad \mid host[\, \overline{finger}\langle c\rangle \mid Inet \\
& \qquad\qquad \mid Finger \mid Daytime\,]\ ) \\
\text{expand } Finger \quad & \equiv \nu c{:}response.(\ host[\, c(x).\overline{print}\langle x\rangle\,] \\
& \qquad \mid host[\, \overline{finger}\langle c\rangle \mid Inet \\
& \qquad\qquad \mid !finger(y).\bar{y}\langle PikeUsers\rangle \mid Daytime\,]\ ) \\
\text{communication} \quad & \xrightarrow{\tau} \nu c{:}response.(\ host[\, c(x).\overline{print}\langle x\rangle\,] \\
\text{on } finger@host \quad & \qquad \mid host[\, Inet \mid \bar{c}\langle PikeUsers\rangle \\
& \qquad\qquad \mid Finger \mid Daytime\,]\ ) \\
\text{communication} \quad & \xrightarrow{\tau} \nu c{:}response.(\ host[\, \overline{print}\langle PikeUsers\rangle\,] \\
\text{on } c@net \quad & \qquad \mid host[Inet \mid Finger \mid Daytime\,]\ )
\end{array}
$$

After a sequence of internal communications at the *net* and *host* level, the first host *Carp* is ready to print the information *PikeUsers*, and host *Pike* is restored to its original configuration.

3 The target π-calculus.

The target calculus is an asynchronous π-calculus [?,1,?], with guarded recursion and name testing.

$$P \ ::= \ a(\boldsymbol{b}).P \ \mid \ \bar{a}\langle \boldsymbol{b} \rangle \ \mid \ 0 \ \mid \ (P|Q) \ \mid \ \mu X.P \ \mid \ \nu x.P \ \mid \ \mathit{if}\ x = y\ \mathit{then}\ P\ \mathit{else}\ Q$$

The absence of output prefixing and choice reflects the local area calculus. Other aspects are tuned to make the encoding as simple as possible by reducing internal transitions and avoiding inert processes. For example, we could easily use replication $!P$ rather than recursion, but this needs an extra trigger channel.

Unusually, our channels carry non-empty lists of values rather than tuples; we write $\boldsymbol{b}$ for this, with ; for list concatenation. This is because the translation multiplexes the action of several polyadic $la\pi$-channels onto a single ether, and hence a single π-channel may carry packets of different sizes. We use head/tail pattern matching on these lists to unwrap packets; in fact, testing on the head element of a list is always enough to determine its length. An alternative would be to further encode lists using standard π-calculus techniques [?], or possibly type packets with some polymorphic datatype [?].

We need to test names for both equality and inequality, and so combine these into a conditional with operational rules derived from those for matching [?].

$$\frac{P \xrightarrow{\alpha} P'}{\mathit{if}\ x = x\ \mathit{then}\ P\ \mathit{else}\ Q \xrightarrow{\alpha} P'} \qquad \frac{Q \xrightarrow{\alpha} Q'}{\mathit{if}\ x = y\ \mathit{then}\ P\ \mathit{else}\ Q \xrightarrow{\alpha} Q'}\ x \neq y$$

With these rules we can consistently add the following convenient structural congruence:

$$(\mathit{if}\ x = x\ \mathit{then}\ P\ \mathit{else}\ Q) \ \equiv \ P\,.$$

The operational semantics of the calculus is otherwise quite standard, and we omit the details.

4 Encoding local areas

In this section we present a compositional encoding of $la\pi$-terms into the π-calculus, following the scheme outlined in the introduction. All communication is mapped into packets passing over designated ether channels; thus tuple output $\bar{a}\langle\vec{b}\rangle$ becomes list output $\bar{e}\langle a; \boldsymbol{b}\rangle$ where ether e varies according to the level of a. To keep track of which ether to use, we maintain an environment Δ mapping levels to ether names. The encoding is parameterized over this, and takes the form

$$[\![\Gamma \vdash_\ell P]\!]_\Delta$$

Structure:

$$
\begin{aligned}
[\![\Gamma \vdash_\ell 0]\!]_\Delta &= 0 \\
[\![\Gamma \vdash_\ell P \mid Q]\!]_\Delta &= [\![\Gamma \vdash_\ell P]\!]_\Delta \ \mid\ [\![\Gamma \vdash_\ell Q]\!]_\Delta \\
[\![\Gamma \vdash_\ell m[P]]\!]_\Delta &= \nu e.[\![\Gamma \vdash_m P]\!]_{\Delta, m\mapsto e} && e \notin \mathit{fn}(P) \cup \mathit{cod}(\Delta) \\
[\![\Gamma \vdash_\ell \nu a{:}\sigma.P]\!]_\Delta &= \nu a.[\![\Gamma, a{:}\sigma \vdash_\ell P]\!]_\Delta
\end{aligned}
$$

Actions, all with $e = \Delta(m)$ where $\Gamma(a) = \vec{\sigma}@m$:

$$
\begin{aligned}
[\![\Gamma \vdash_\ell \bar{a}\langle\vec{b}\rangle]\!]_\Delta &= \bar{e}\langle a; \boldsymbol{b}\rangle \\
[\![\Gamma \vdash_\ell a(\vec{b}).P]\!]_\Delta &= \mu X.e(x;\boldsymbol{b}).\mathit{if}\ x = a\ \mathit{then}\ [\![\Gamma, \boldsymbol{b}{:}\vec{\sigma} \vdash_\ell P]\!]_\Delta\ \mathit{else}\ (\bar{e}\langle x;\boldsymbol{b}\rangle \mid X) \\
[\![\Gamma \vdash_\ell !a(\vec{b}).P]\!]_\Delta &= \mu X.e(x;\boldsymbol{b}).(X \mid \mathit{if}\ x = a\ \mathit{then}\ [\![\Gamma, \boldsymbol{b}{:}\vec{\sigma} \vdash_\ell P]\!]_\Delta\ \mathit{else}\ \bar{e}\langle x;\boldsymbol{b}\rangle)
\end{aligned}
$$

Fig. 5. Rules for encoding $la\pi$ into the plain π-calculus

where P is a well-typed term of level ℓ in context Γ, and Δ assigns ethers to levels ℓ and above.

Figure 5 presents the full encoding, with one clause for each constructor; here we go through each one individually. The null process, parallel composition, and name restriction are unchanged.

$$
\begin{aligned}
[\![\Gamma \vdash_\ell 0]\!]_\Delta &= 0 \\
[\![\Gamma \vdash_\ell P \mid Q]\!]_\Delta &= [\![\Gamma \vdash_\ell P]\!]_\Delta \mid [\![\Gamma \vdash_\ell Q]\!]_\Delta \\
[\![\Gamma \vdash_\ell \nu a{:}\sigma.P]\!]_\Delta &= \nu a.[\![\Gamma, a{:}\sigma \vdash_\ell P]\!]_\Delta
\end{aligned}
$$

To place a process in a local area, as an agent, we create a new ether name and assign it a level in the environment Δ. A side condition ensures that we do not accidentally capture any existing names when introducing the new ether.

$$
[\![\Gamma \vdash_\ell m[P]]\!]_\Delta = \nu e.[\![\Gamma \vdash_m P]\!]_{\Delta, m\mapsto e} \qquad e \notin \mathit{fn}(P) \cup \mathit{cod}(\Delta)
$$

Translating an output action uses the environment and the assignment of levels to ethers to find the correct ether for the output channel. It then sends both the output channel and the data for transmission over this ether name as a list.

$$
[\![\Gamma \vdash_\ell \bar{a}\langle\vec{b}\rangle]\!]_\Delta = \bar{e}\langle a; \boldsymbol{b}\rangle \qquad \text{with } e = \Delta(m) \text{ where } \Gamma(a) = \vec{\sigma}@m
$$

An encoded input also uses the environment and the assignment of levels to ether names to find which ether it should listen on. When it receives a packet over this ether, it tests the head of the list to see if it matches the input channel name. If it does, then the packet is meant for this input and execution continues as appropriate. If the names do not match, then this packet is meant for some other channel in the same area. The packet is resent and the process restarts.

$$[\![\Gamma \vdash_\ell a(\vec{b}).P]\!]_\Delta = \mu X.e(x;\boldsymbol{b}).\mathit{if}\ x = a\ \mathit{then}\ [\![\Gamma, \boldsymbol{b}{:}\vec{\sigma} \vdash_\ell P]\!]_\Delta\ \mathit{else}\ (\bar{e}\langle x;\boldsymbol{b}\rangle \mid X)$$
$$\text{with } e = \Delta(m) \text{ where } \Gamma(a) = \vec{\sigma}@m.$$

Replicated input is the same, except that the process restarts whether or not the input key is correctly matched.

$$[\![\Gamma \vdash_\ell !a(\vec{b}).P]\!]_\Delta = \mu X.e(x;\boldsymbol{b}).(X \mid \mathit{if}\ x = a\ \mathit{then}\ [\![\Gamma, \boldsymbol{b}{:}\vec{\sigma} \vdash_\ell P]\!]_\Delta\ \mathit{else}\ \bar{e}\langle x;\boldsymbol{b}\rangle)$$
$$\text{with } e = \Delta(m) \text{ where } \Gamma(a) = \vec{\sigma}@m.$$

The encoding is well-defined up to structural congruence.

Proposition 4.1 *For any $la\pi$-terms P and Q, if $P \equiv Q$ then $[\![P]\!]_\Delta \equiv [\![Q]\!]_\Delta$.*

Proof. Because the encoding is compositional, it is enough to check that all of the structural axioms for $la\pi$ given at the end of §2.1 translate to valid π-calculus equivalences. All of these are immediate; the only significant case is that $\ell[\nu a{:}\sigma.P] \equiv \nu a{:}\sigma.(\ell[P])$ becomes exchange of name binders $\nu e.\nu a.[\![P]\!]_\Delta \equiv \nu a.\nu e.[\![P]\!]_\Delta$ for some ether e. □

It is worthwhile noting that the encoding uses π-calculus channels in a highly stereotyped manner. Names fall into two distinct classes: data names, like a and b, which correspond directly to $la\pi$ channels, and ether names like e. All communication involves sending data over ethers. Data names are never used as channels, while ether names are never transmitted, nor do they appear in match tests. One consequence of this is that all our terms happen to lie in the subset studied by Merro [8] as the *local* π-calculus, where names sent over channels may not be used for further communication.

In the introduction we mentioned that in general π-calculus names have a dual rôle, for identity and for communication; what happens in the translation is that each rôle is mapped to a different name.

5 Correctness of the encoding

A $la\pi$-process and its encoding behave in very similar ways, and this is preserved under reduction. Our main result is that they enjoy a form of bisimilarity on outputs, up to the translation between direct and ether-based communication.

Theorem 5.1 *For any well-typed process $\Gamma \vdash_\ell P$ in the local area π-calculus and transition $\alpha = \bar{a}\langle\vec{b}\rangle$ or $\alpha = \tau$, the following hold.*

(i) *If $\Gamma \vdash_\ell P \stackrel{\alpha}{\Longrightarrow} P'$ then $[\![\Gamma \vdash_\ell P]\!]_\Delta \stackrel{[\![\alpha]\!]_\Delta}{\Longrightarrow} [\![\Gamma \vdash_\ell P']\!]_\Delta$.*

(ii) *If $[\![\Gamma \vdash_\ell P]\!]_\Delta \stackrel{[\![\alpha]\!]_\Delta}{\Longrightarrow} Q$ then there is P' such that $\Gamma \vdash_\ell P \stackrel{\alpha}{\Longrightarrow} P'$ and $[\![\Gamma \vdash_\ell P']\!]_\Delta \equiv Q$.*

Here $[\![\tau]\!]_\Delta = \tau$ *and* $[\![\bar{a}\langle\vec{b}\rangle]\!]_\Delta = \bar{e}\langle a, \boldsymbol{b}\rangle$ *where* $e = \Delta(m)$ *for* $\Gamma(a) = \vec{\sigma}@m$.

This result says nothing about inputs: in fact, a term and its encoding generally have different input behaviour, with the translated terms being more receptive than the original. It is conventional though in asynchronous calculi to regard only output as observable, as there is no way in principle to know when an input has been received.

In the terminology of Nestmann and Pierce [?], this is an *operational correspondence* between the calculi. Although expressed in terms of weak transitions, the correspondence is in fact rather close. Transitions match exactly, except that a single internal π-transition may map to zero $la\pi$-transitions. Unfortunately this does introduce the possibility of divergence: most translated terms can perform an unbounded sequence of τ steps as they collect and return ether packets. Divergence also arises in Nestmann and Pierce's choice encoding, except that there it is inserted by design, to give a more convenient full abstraction result; their initial encoding is divergence free. In our system, divergence arises rather naturally from mechanism of ethers. We expect that replacing this with more pragmatic lists of readers and writers would lead to a divergence-free encoding, but at a cost of considerable complexity.

Theorem 5.1 follows without difficulty from the following more precise results, which characterise exactly the possible actions of encoded processes.

Lemma 5.2 *For any well-typed process* $\Gamma \vdash_\ell P$ *in the local area* π*-calculus the following hold. In each case* $Q = [\![\Gamma \vdash_\ell P]\!]_\Delta$ *and* $e = \Delta(m)$ *where* $\Gamma(a) = \vec{\sigma}@m$.

(i) *If* $\Gamma \vdash_\ell P \xrightarrow{\bar{a}\langle\vec{b}\rangle} P'$ *then* $Q \xrightarrow{\bar{e}\langle a,\boldsymbol{b}\rangle} Q'$ *where* $Q' \equiv [\![\Gamma \vdash_\ell P']\!]_\Delta$.

(ii) *If* $\Gamma \vdash_\ell P \xrightarrow{a(\vec{b})} P'$ *then* $Q \xrightarrow{e(x,\boldsymbol{b})} Q'$ *such that for any vector of names* $\vec{c}$ *we have* $Q'\{a;\boldsymbol{c}/x;\boldsymbol{b}\} \equiv [\![\Gamma' \vdash_\ell P'\{\vec{c}/\vec{b}\}]\!]_\Delta$.

(iii) *If* $\Gamma \vdash_\ell P \xrightarrow{\tau} P'$ *then* $Q \xrightarrow{\tau} Q'$ *where* $Q' = [\![\Gamma \vdash_\ell P']\!]_\Delta$.

(iv) *If* $Q \xrightarrow{\bar{e}\langle a;\boldsymbol{b}\rangle} Q'$ *then* $\Gamma \vdash_\ell P \xrightarrow{\bar{a}\langle\vec{b}\rangle} P'$ *with* $[\![\Gamma \vdash_\ell P']\!]_\Delta \equiv Q'$.

(v) *If* $Q \xrightarrow{e(x;\boldsymbol{b})} Q'$ *then either* $Q' \equiv \bar{e}\langle x;\boldsymbol{b}\rangle \,|\, Q$ *or* $\Gamma \vdash_\ell P \xrightarrow{a(\vec{b})} P'$ *where for any vector of names* $\vec{c}$ *we have* $Q'\{a;\boldsymbol{c}/x;\boldsymbol{b}\} \equiv [\![\Gamma' \vdash_\ell P'\{\vec{c}/\vec{b}\}]\!]_\Delta$.

(vi) *If* $Q \xrightarrow{\tau} Q'$ *then either* $Q \equiv Q'$ *or* $\Gamma \vdash_\ell P \xrightarrow{\tau} P'$ *with* $[\![\Gamma \vdash_\ell P']\!]_\Delta \equiv Q'$.

This makes clear the close connection between $la\pi$-transitions and ether packets. For output, the correspondence is exact: a process can perform an output if and only if its translation can. For input, recall that when an encoded process reads a packet, it tests it and if unsuitable it retransmits the packet again and continues as before. This means that an input in the encoded system may be matched by a similar input in the system it encodes or it may perform an output and revert to the original process.

This choice of two possible responses to any input action is carried over to

$$\llbracket\, \Gamma \vdash_{net} \mathit{Carp} \mid \mathit{Pike} \,\rrbracket_{\{net \mapsto n\}} \;=\; \mathit{Carp}' \mid \mathit{Pike}'$$

$$\begin{aligned}
\mathit{Carp}' &= \nu q.\nu c.\,(\; \bar{n}\langle \mathit{pike}, \mathit{finger}, c\rangle \\
&\qquad\qquad \mid \mu X.n(x;\boldsymbol{y}).\mathit{if}\ x = c\ \mathit{then}\ \bar{q}\langle \mathit{print};\boldsymbol{y}\rangle\ \mathit{else}\ (\bar{n}\langle x;\boldsymbol{y}\rangle \mid X)\;) \\
\mathit{Pike}' &= \nu p.(\mathit{Inet}' \mid \mathit{Finger}' \mid \mathit{Daytime}')
\end{aligned}$$

$$\begin{aligned}
\mathit{Inet}' &= \mu X.n(x;\boldsymbol{y}).(X \mid \mathit{if}\ x = \mathit{pike}\ \mathit{then}\ \bar{p}\langle \boldsymbol{y}\rangle\ \mathit{else}\ \bar{n}\langle x;\boldsymbol{y}\rangle) \\
\mathit{Finger}' &= \mu X.p(s,r).(X \mid \mathit{if}\ s = \mathit{finger}\ \mathit{then}\ \bar{n}\langle r, \mathit{PikeUsers}\rangle\ \mathit{else}\ \bar{p}\langle s,r\rangle) \\
\mathit{Daytime}' &= \mu X.p(s,r).(X \mid \mathit{if}\ s = \mathit{daytime}\ \mathit{then}\ \bar{n}\langle r, \mathit{PikeDate}\rangle\ \mathit{else}\ \bar{p}\langle s,r\rangle)
\end{aligned}$$

Ether names: n, p, q

Data names: $\mathit{pike}, \mathit{finger}, \mathit{daytime}, \mathit{print}, c, r, s$

Fig. 6. Example of processes using local areas: an Internet server dæmon

the case of an encoded process performing a τ. This may either reflect a τ in the system it encodes or it may be a rejected communication, in which case the process it reduces to is congruent to the original.

The proofs for each clause in the lemma follow a similar pattern. For clauses (i)–(iii), we break down a process into the part that performs the action and a surrounding context. Next we use the encoding rules to encode these parts. Then we show how the encoding of the part of the $la\pi$-process that performs the action can perform a matching π-action. Finally, we show that the encoding of the context allows this similar action to escape. There is a dependency, in that we must prove parts (i) and (ii) before (iii).

Clauses (iv)–(vi) are proved similarly, but in the reverse direction. First we break down Q, into the parts that perform the action and a context, then using this decomposition we characterize P, finally we show that this P can perform the required action and reduce to a process matching Q'.

The direct relationship between the behaviour of a process and its encoding make the proof much easier. In particular, there are no intermediate forms on the π-calculus side to be analysed. If there were such additional "housekeeping" steps, then we would need to enlarge the lemma to cover a one-to-many relation R_Δ between $la\pi$-processes and π-terms.

6 Encoding of the internet daemon

To illustrate how this works we encode the `inetd` example from Section 2.5. Figure 6 shows the result, which can be compared with Figure 4. The translation uses three ethers, for which we take names n, p and q to cover the network, server host *Pike* and client host *Carp* respectively. All $la\pi$-channel names like *finger* map to themselves.

Figure 7 represents graphically the behaviour of the translated system.

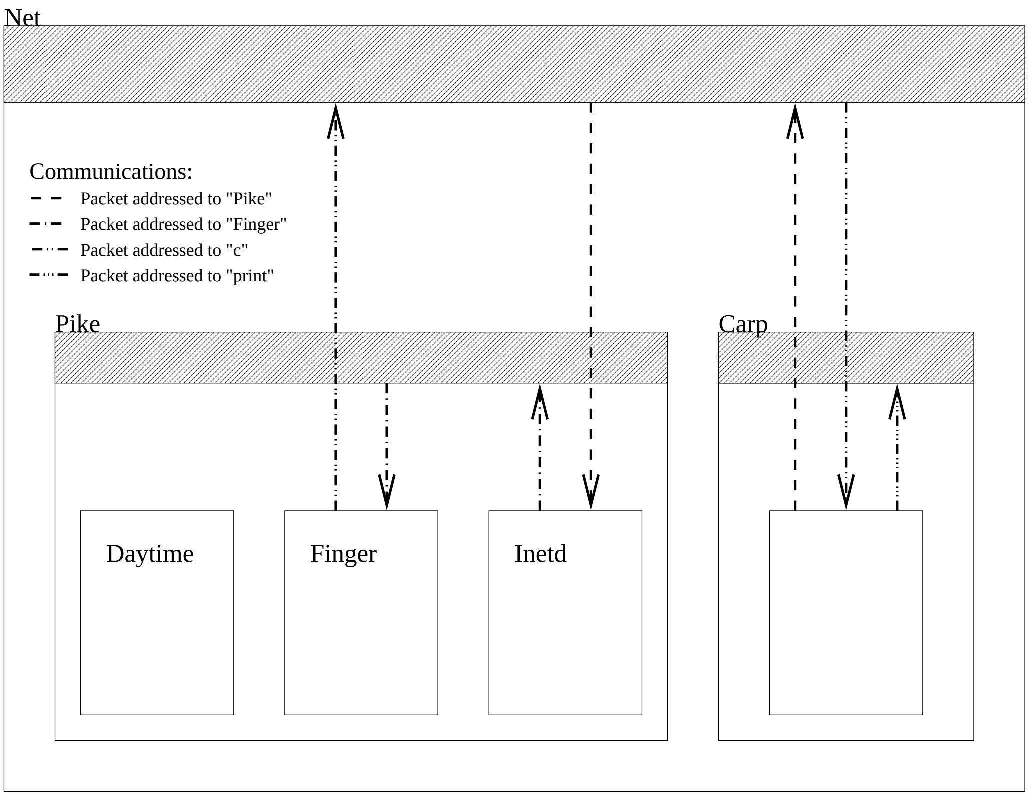

Fig. 7. Ether-based encoding of `inet` dæmon relaying `finger` service

Grey bars indicate the local ethers; compare this to the direct links of Figure 1. As we expect from Lemma 5.2, reductions of the translated process closely match those of the original given earlier.

$$
\begin{aligned}
\mathit{Carp}' \mid \mathit{Pike}' &\equiv \; (\ \nu q.\nu c.(\bar{n}\langle \mathit{pike}, \mathit{finger}, c\rangle \mid \mu X.n(x;\boldsymbol{y})\ldots \\
&\qquad \mid \nu p.(\mathit{Inet}' \mid \mathit{Finger}' \mid \mathit{Daytime}')\ ) \\
\text{extend scope} &\equiv \nu p, q, c.\,(\ \bar{n}\langle \mathit{pike}, \mathit{finger}, c\rangle \mid \mu X.n(x;\boldsymbol{y})\ldots \\
\text{of } p,\ q \text{ and } c &\qquad\qquad \mid \mathit{Inet}' \mid \mathit{Finger}' \mid \mathit{Daytime}'\ ) \\
\text{unroll } \mathit{Inet}' &\equiv \nu p, q, c.\,(\ \bar{n}\langle \mathit{pike}, \mathit{finger}, c\rangle \mid \mu X.n(x;\boldsymbol{y})\ldots \\
&\qquad\qquad \mid n(x;\boldsymbol{y}).(\mathit{Inet}' \mid \mathit{if}\ x = \mathit{pike}\ \mathit{then}\ \bar{p}\langle \boldsymbol{y}\rangle\ \mathit{else}\ \bar{n}\langle x;\boldsymbol{y}\rangle) \\
&\qquad\qquad \mid \mathit{Finger}' \mid \mathit{Daytime}'\ ) \\
\text{communication} &\xrightarrow{\tau} \nu p, q, c.\,(\ \mu X.n(x;\boldsymbol{y})\ldots \\
\text{of } \mathit{pike} \text{ over } n &\qquad\qquad \mid \mathit{if}\ \mathit{pike} = \mathit{pike}\ \mathit{then}\ \bar{p}\langle \mathit{finger}, c\rangle\ \mathit{else}\ldots \\
&\qquad\qquad \mid \mathit{Inet}' \mid \mathit{Finger}' \mid \mathit{Daytime}'\ )
\end{aligned}
$$

$$
\begin{array}{rll}
\text{apply test and} & \equiv \nu p, q, c.\, (\ \mu X.n(x; \boldsymbol{y}) \ldots & \\
\text{unroll } \mathit{Finger}' & \quad\mid \bar{p}\langle \mathit{finger}, c\rangle & \\
& \quad\mid p(s, r).(\mathit{Finger}' \mid \mathit{if}\ s = \mathit{finger} & \\
& \qquad\qquad\qquad \mathit{then}\ \bar{n}\langle r, \mathit{PikeUsers}\rangle\ \mathit{else}\ \ldots) & \\
& \quad\mid \mathit{Inet}' \mid \mathit{Daytime}'\ ) & \\
\text{communication} & \xrightarrow{\tau} \nu p, q, c.\, (\ \mu X.n(x; \boldsymbol{y}) \ldots & \\
\text{of } \mathit{finger} \text{ over } p & \qquad \mid \mathit{if}\ \mathit{finger} = \mathit{finger}\ \mathit{then}\ \bar{n}\langle c, \mathit{PikeUsers}\rangle\ \mathit{else} \ldots & \\
& \qquad \mid \mathit{Inet}' \mid \mathit{Finger}' \mid \mathit{Daytime}'\ ) & \\
\text{apply test} & \equiv \nu p, q, c.\, (\ \mu X.n(x; \boldsymbol{y}).\mathit{if}\ x = c\ \mathit{then}\ \bar{q}\langle \mathit{print}; \boldsymbol{y}\rangle\ \mathit{else} \ldots & \\
& \quad\mid \bar{n}\langle c, \mathit{PikeUsers}\rangle & \\
& \quad\mid \mathit{Inet}' \mid \mathit{Finger}' \mid \mathit{Daytime}'\ ) & \\
\text{communication} & \xrightarrow{\tau} \nu p, q, c.\, (\ \mathit{if}\ c = c\ \mathit{then}\ \bar{q}\langle \mathit{print}, \mathit{PikeUsers}\rangle\ \mathit{else} \ldots & \\
\text{of } c \text{ over } n & \qquad \mid \mathit{Inet}' \mid \mathit{Finger}' \mid \mathit{Daytime}'\ ) & \\
\text{apply test} & \equiv \nu p, q, c.\, (\ \bar{q}\langle \mathit{print}, \mathit{PikeUsers}\rangle & \\
& \quad\mid \mathit{Inet}' \mid \mathit{Finger}' \mid \mathit{Daytime}'\ ) &
\end{array}
$$

Comparing the reduction in given in Section 2.5, notice how communication restricted to a local area ("communication on *finger*@*host*") is replaced by communication on a local ether ("communication of *finger* over p").

Unlike the original $la\pi$-term, other reduction sequences are possible, though they will only add extra τ-transitions. For example, the $\mathit{Daytime}'$ server may mistakenly pick up the *finger* request, but will always immediately rebroadcast it.

7 Conclusion and further work

We have encoded a notion of distributed areas and local communication into the π-calculus, by giving a translation of the local area π-calculus. At the core of this encoding is the technique of replacing communication on a channel name with communication over an ether associated with the appropriate local area.

The operational correspondence of Section 5 says that there is a close relation between the actions of processes and their translations. The next step is to build on this to investigate the degree to which the encoding preserves and reflects equivalences between processes. We would expect adequacy, but not full abstraction, as encoding local areas by ethers exposes them to probing by general π-calculus terms. For example, it is possible to eavesdrop on all top-level communications, even ones involving private names ("packet-snooping"),

Well-known names that mean different things in different places are reminiscent of dynamic binding in programming languages; that slippery concept whereby the meaning of a local variable at a program point depends on how we got there. While there seems to be no direct connection, it would be interesting to know how local areas affect the classic encoding of functions as π-calculus processes [**?**].

Limiting communication to local areas can be seen as a form of "security". The $la\pi$-calculus does not itself prove processes to be secure, but instead can show how particular protocols operate under imposed security constraints. (In this sense it is about liveness, rather than safety.) We hope to use this to model aspects of Network Address Translation (NAT), a standard method for shared internet access, which is known to interact poorly with certain kinds of application.

The fixed arrangement of local areas in $la\pi$ does not lend itself to a dynamic runtime structure. There is however some flexibility: where areas appear under replication, they will be freshly created during execution; and empty areas are indistinguishable from the null process. For more general mobility, we are working on an extension of $la\pi$ with primitives for relocating areas, and an associated type system. The encoding given in the present paper does not extend to handle mobility, because it assumes that each process has direct access to the ethers for every containing level. We can suggest a solution though, using an encoding with a network of controllers. Within an area, each process communicates only through its immediate local area controller. Packets are routed by controllers to their destination, and mobile areas can be represented by reprogramming the controllers.

References

[1] Roberto Amadio, Ilaria Castellani, and Davide Sangiorgi. On bisimulations for the asynchronous π-calculus. *Theoretical Computer Science*, 195:291–324, 1998.

[2] Gérard Boudol. Asynchrony and the π-calculus. Rapport de recherche 1702, INRIA, Sophia Antipolis, 1992.

[3] Gérard Boudol, Ilaria Castellani, and Davide Sangiorgi. Observing localities. *Theoretical Computer Science*, 114:31–61, 1993.

[4] Gérard Boudol, Ilaria Castellani, and Davide Sangiorgi. A theory of processes with localities. *Formal Aspects of Computing*, 6:165–200, 1994.

[5] Luca Cardelli, Giorgio Ghelli, and Andrew D. Gordon. Secrecy and group creation. In *CONCUR '2000: Concurrency Theory. Proceedings of the 11th International Conference*, Lecture Notes in Computer Science 1877, pages 365–379. Springer-Verlag, 2000.

[6] Luca Cardelli and Andrew D. Gordon. Mobile ambients. In *Foundations of Software Science and Computation Structure: Proceedings of FoSSaCS '98*, Lecture Notes in Computer Science 1378, pages 140–155. Springer-Verlag, 1998.

[7] Giuseppe Castagna and Jan Vitek. Seal: A framework for secure mobile computations. In *Internet Programming Languages*, Lecture Notes in Computer Science 1686, pages 47–77. Springer-Verlag, 1999.

[8] Tom Chothia and Ian Stark. A distributed π-calculus with local areas of communication. In *Proceedings of HLCL '00: High-Level Concurrent Languages*, Electronic Notes in Theoretical Computer Science 41.2. Elsevier, 2001.

[9] Cédric Fournet, Jean-Jacques Lévy, and Alain Schmitt. An asynchronous distributed implementation for mobile ambients. In *Theoretical Computer Science: Proceedings of TCS 2000*, Lecture Notes in Computer Science 1872, pages 348–364. Springer-Verlag, August 2000.

[10] Matthew Hennessy and James Riely. Resource access control in systems of mobile agents. In *Proceedings of HLCL '98: High-Level Concurrent Languages*, Electronic Notes in Theoretical Computer Science 16.3, pages 3–17. Elsevier, 1998.

[11] Kohei Honda and Mario Tokoro. An object calculus for asynchronous communication. In *ECOOP '91: Proceedings of the European Conference on Object-Oriented Programming*, Lecture Notes in Computer Science 512, pages 133–147. Springer-Verlag, July 1991.

[12] IANA, the Internet Assigned Numbers Authority. Protocol numbers and assignment services: Port numbers. `http://www.iana.org/numbers.html\#P`.

[13] Massimo Merro. *Locality in the π-calculus and applications to distributed objects*. PhD thesis, Ecole des Mines, France, October 2000.

[14] Robin Milner. The polyadic π-calculus — a tutorial. Technical Report ECS-LFCS-91-180, Laboratory for Foundations of Computer Science, University of Edinburgh, 1991.

[15] Robin Milner, Joachim Parrow, and David Walker. A calculus of mobile processes, parts I and II. *Information and Computation*, 100:1–77, 1992.

[16] Uwe Nestmann and Benjamin C. Pierce. Decoding choice encodings. BRICS report RS-99-42, Department of Computer Science, University of Aarhus, December 1999. To appear in *Information and Computation*; a version appeared as a paper at CONCUR '96.

[17] Benjamin Pierce and Davide Sangiorgi. Behavioral equivalence in the polymorphic pi-calculus. *Journal of the ACM*, 47(5):531–584, September 2000.

[18] James Riely and Matthew Hennessy. Distributed processes and location failures. In *Automata, Languages and Programming: Proceedings of the 24th International Colloquium ICALP '97*, Lecture Notes in Computer Science 1256, pages 471–481. Springer-Verlag, 1997.

[19] Davide Sangiorgi. Lazy functions and mobile processes. Rapport de recherche 2515, INRIA, Sophia Antipolis, 1995.

[20] Davide Sangiorgi. Locality and non-interleaving semantics in calculi for mobile processes. *Theoretical Computer Science*, 155:39–83, 1996.

[21] Peter Sewell. Global/local subtyping and capability inference for a distributed π-calculus. In *Automata, Languages and Programming: Proceedings of the 25th International Colloquium ICALP 98*, Lecture Notes in Computer Science 1442. Springer-Verlag, 1998.

[22] David Turner. *The Polymorphic Pi-Calculus: Theory and Implementation*. PhD thesis, Laboratory for Foundations of Computer Science, Edinburgh University, 1996. Also published as LFCS Technical Report 345.

[23] José-Luis Vivas and Mads Dam. From higher-order π-calculus to π-calculus in the presence of static operators. In *CONCUR '98: Concurrency Theory. Proceedings of the 9th International Conference*, Lecture Notes in Computer Science 1466. Springer-Verlag, 1998.