UCAM overview

PEPITO Workshop 1 at Rocquencourt, Jan 31 - Feb 1

Peter Sewell

University of Cambridge

January 29, 2002

This talk: a map of our recent and current interests.

(some work of less relevance omitted)

Sketchy overview only – more details on request...

People

Us here: Gavin Bierman, Peter Sewell, Keith Wansbrough

PhD students: Moritz Becker, Matthew Parkinson, Andrei Serjantov, Gareth Stoyle

ex students/pdocs: Pawel Wojciechowski (EPFL), Asis Unyapoth

(and also involved: Michael Norrish, Andy Pitts)

Nomadic Pi (Sewell, Wojciechowski, Unyapoth, Pierce)

Aim: Understand Distributed Infrastructure for mobile systems

algs are becoming complex, esp. with mobility - to understand them we need languages that have clean semantics (not just psuedocode)

focus on one problem - infrastructure for...

Location Independence

Focus on one problem – the design of communication primitives for agents to interact.

Communication primitives (e.g. async. messages):

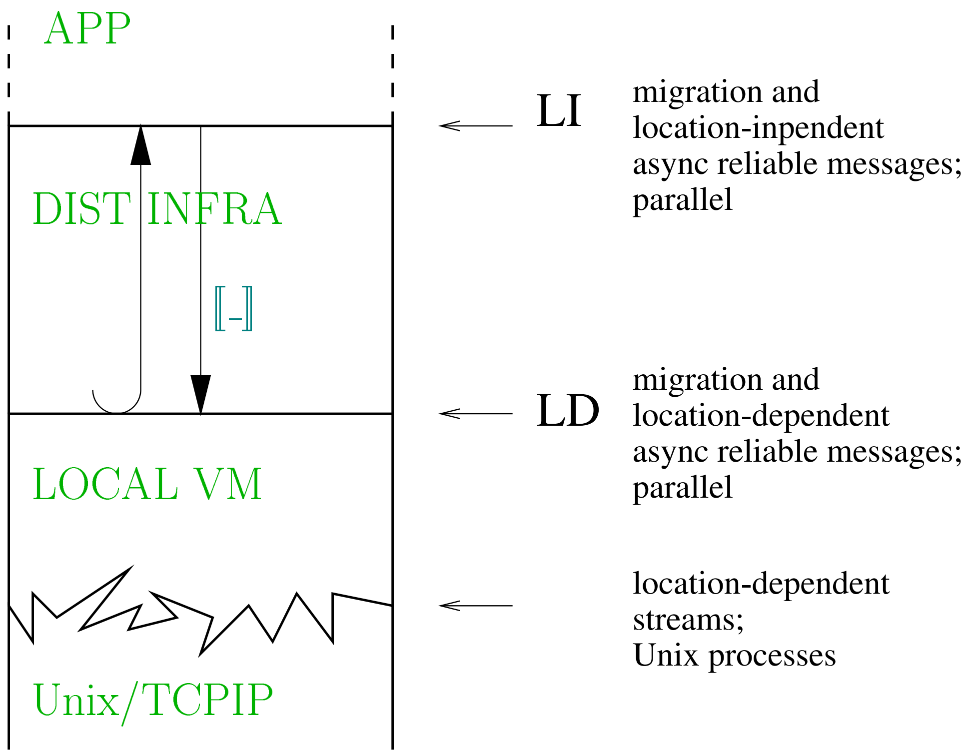
- Location Independent
- Location Dependent
- Same site

LI primitives potentially simplify application development

...BUT...

they need delicate distributed infrastructure

Nomadic Pi: Modular Design (Two Levels of Abstraction)



define levels by giving process calculi (with reduction semantics)

- Many infrastructures
- Understanding of the algorithms
- Experimental implementation and prototyping
- Correctness and robustness results

Nomadic Pi: Low- and High-Level Calculi

Names of sites s, of agents a, b, and of channels c

Low-Level: Agents

$\textbf{agent}\ a = P\ \textbf{in}\ Q$	agent creation
$\textbf{migrate to}\ s \rightarrow P$	agent migration
$P \mid Q$	parallel composition
0	nil

π-calculus style communication within agents

$\textbf{new}\ c\ \textbf{in}\ P$	new channel name creation
c!v	output v in the current agent
$c?w \rightarrow P$	input from channel c
$*c?w \rightarrow P$	replicated input from channel c
$\textbf{if}\ a = b\ \textbf{then}\ P\ \textbf{else}\ Q$	name equality testing

Inter-agent communication

$\langle a@s \rangle c!v$	LD output to agent a on site s
$\textbf{iflocal}\ \langle a \rangle c!v \rightarrow P\ \textbf{else}\ Q$	test-and-send

High-Level:

$\langle a@? \rangle c!v$	LI output to agent a

Nomadic Pi: Example Encoding – Central Daemon

A Daemon at the initiating site, maintaining a map with the current site of every agent.

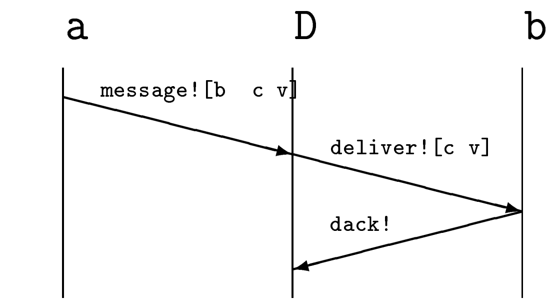
Agents synchronize with the Daemon on creation and before & after migration.

An LI message for an agent is sent to the Daemon, which delivers it.

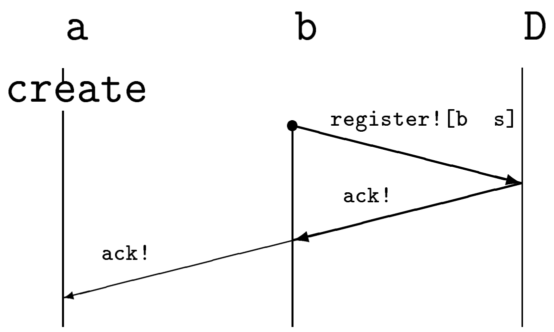
(this is one of the simplest possible algorithms. It has poor performance and robustness – but fits on a slide...)

Nomadic Pi: Example Encoding – Central Daemon

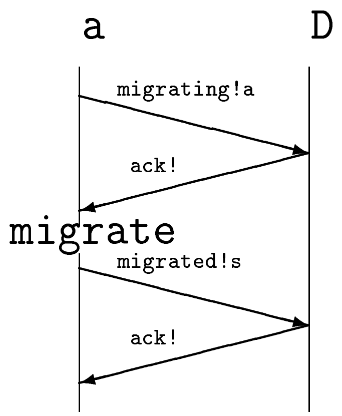
Location-independent output



Creation



Migration



Nomadic Pi: Example Encoding – Central Daemon

$[\![P]\!]$ is roughly $[\![P]\!]_a | DAEMON$

$$
\begin{array}{lcl}
[\![\langle b@?\rangle c!v]\!]_a & = & \langle D@Dsite\rangle message![bcv] \\[1em]
[\![\textbf{agent}\ b = P\ \textbf{in}\ Q]\!]_a & = & currentloc?s \rightarrow \\
 & & \quad \textbf{agent}\ b = \\
 & & \qquad\quad * deliver?cv \rightarrow (\langle D@Dsite\rangle dack![] | \mathtt{c!v}) \\
 & & \qquad | \langle D@Dsite\rangle register![bs] \\
 & & \qquad | ack? \rightarrow (\langle a@s\rangle ack![] | \mathtt{currentloc!s} | [\![P]\!]_b) \\
 & & \quad \textbf{in} \\
 & & \qquad ack? \rightarrow (\mathtt{currentloc!s} | [\![Q]\!]_a)
\end{array}
$$

$$
\begin{array}{lcl}
[\![\langle b@?\rangle c!v]\!]_a & = & \langle D@Dsite\rangle message![bcv] \\
[\![\textbf{agent}\ b = P\ \textbf{in}\ Q]\!]_a & = & currentloc?s \rightarrow \\
 & & \quad \textbf{agent}\ b = \\
 & & \qquad\quad * deliver?cv \rightarrow (\langle D@Dsite\rangle dack![] | \mathtt{c!v}) \\
 & & \qquad | \langle D@Dsite\rangle register![bs] \\
 & & \qquad | ack? \rightarrow (\langle a@s\rangle ack![] | \mathtt{currentloc!s} | [\![P]\!]_b) \\
 & & \quad \textbf{in} \\
 & & \qquad ack? \rightarrow (\mathtt{currentloc!s} | [\![Q]\!]_a) \\
[\![\textbf{migrate to}\ u \rightarrow P]\!]_a & = & currentloc?\_ \rightarrow \\
 & & \qquad \langle D@Dsite\rangle migrating!a \\
 & & \quad | ack? \rightarrow \\
 & & \qquad\quad \textbf{migrate to}\ u \rightarrow \\
 & & \qquad\qquad \langle D@Dsite\rangle register![au] \\
 & & \qquad\quad | ack? \rightarrow (\mathtt{currentloc!u} | [\![P]\!]_a) \\
[\![0]\!]_a & = & \\
[\![P|Q]\!]_a & = & \\
[\![c?w \rightarrow P]\!]_a & = & \\
[\![*c?w \rightarrow P]\!]_a & = & \text{ALL HOMOMORPHIC} \\
[\![\textbf{iflocal}\ \langle b\rangle c!v \rightarrow P\ \textbf{else}\ Q]\!]_a & = & \\
[\![\textbf{new}\ c\ \textbf{in}\ P]\!]_a & = & \\
[\![\textbf{if}\ a = b\ \textbf{then}\ P\ \textbf{else}\ Q]\!]_a & = &
\end{array}
$$

$$
\begin{array}{l}
DAEMON = \textbf{new}\ lock\ \textbf{in} \\
\qquad \mathtt{lock!}\textbf{emptymap} \\
\quad | * register?as \rightarrow \\
\qquad\quad lock?m \rightarrow \\
\qquad\quad \textbf{let}\ m' = (m\ \textbf{with}\ a \mapsto s)\ \textbf{in} \\
\qquad\qquad \mathtt{lock!m'} | \langle a@s\rangle ack![] \\
\quad | * migrating?a \rightarrow \\
\qquad\quad lock?m \rightarrow \\
\qquad\quad \textbf{lookup}\ a\ \textbf{in}\ m\ \textbf{with} \\
\qquad\qquad \textbf{found}(s) \rightarrow \\
\qquad\qquad\qquad \langle a@s\rangle ack![] \\
\qquad\qquad\quad | migrated?s \rightarrow \\
\qquad\qquad\qquad \textbf{let}\ m' = (m\ \textbf{with}\ a \mapsto s')\ \textbf{in} \\
\qquad\qquad\qquad \mathtt{lock!m'} | \langle a@s'\rangle ack![] \\
\qquad\qquad \textbf{notfound} \rightarrow 0 \\
\quad | * message?acv \rightarrow \\
\qquad\quad lock?m \rightarrow \\
\qquad\quad \textbf{lookup}\ a\ \textbf{in}\ m\ \textbf{with} \\
\qquad\qquad \textbf{found}(s) \rightarrow \\
\qquad\qquad\qquad \langle a@s\rangle deliver![cv] \\
\qquad\qquad\quad | dack? \rightarrow \mathtt{lock!m} \\
\qquad\qquad \textbf{notfound} \rightarrow 0
\end{array}
$$

$$
\begin{array}{lcl}
[\![@_a P]\!]_s & = & @_a \textbf{new}\ register, migrating, message, dack, deliver, ack, currentloc\ \textbf{in} \\
 & & \quad \textbf{agent}\ D = DAEMON\ \textbf{in} \\
 & & \qquad \textbf{let}\ Dsite = s\ \textbf{in} \\
 & & \qquad\qquad * deliver?cv \rightarrow (\langle D@Dsite\rangle dack![] | \mathtt{c!v}) \\
 & & \qquad\quad | \langle D@Dsite\rangle register![as] \\
 & & \qquad\quad | ack? \rightarrow (\mathtt{currentloc!s} | [\![P]\!]_a)
\end{array}
$$

Nomadic Pi: Implementation (Wojciechowski)

NOMADIC PICT compiler and distributed runtime, based on Pict, written in OCaml. Takes an LI program and an encoding; produces and executes LD byte-code. [PhD thesis, ASAMA99, IEEE Concurrency 00.]

Nomadic Pi: Algorithms (Wojciechowski)

Broad exploration of the design space; many algorithms prototyped. [WIPL98, PhD thesis, AISB01].

Nomadic Pi: Semantics (Unyapoth)

Typing, Reduction and LTS semantics.

Proof techniques dealing with mobility (translocating bisimulation; temporary immobility).

Correctness result for a 'simple' algorithm: for all high level P we have $P \approx [\![P]\!]$ [PhD thesis, POPL01].

Secure Encapsulation (Sewell, Vitek)

Question: how to control interaction between untrusted components?

Answer: encapsulate them in wrappers.

Harder question: how to state and prove the properties such wrappers guarantee?
Our answer:

- design a *box-π* calculus of components and wrappers, with constrained interaction
- program some simple examples in box-π
- express various security properties (with coloured semantics)
- develop a type system for causal flow, to prove them with

[CSFW99,CSFW00,JCS]

Versioning (Sewell)

Problem

Consider a wide-area distributed program:

- many machines, many admin domains, long-lived...
  ...so can't synchronise software updates, so have many interacting versions

  (eg Nomadic Pict communication infrastructures for mobile computation)
- may have communication at abstract types
- hard to debug, so want to detect errors at build time...
  ...so want to use static typing as far as possible

(Partial!) Solution

Consider an ML-style module system, with abstract and transparent functors (expressed with singleton kinds).

Develop an operational semantics, introducing new-binders during reduction of abstract modules.

Allow a type-safe versioning coercion of abstract types with compatible representations [POPL01].