Objects and Mobility

Benjamin C. Pierce

University of Pennsylvania

Global computing

Some Questions for FOOLs

Benjamin C. Pierce

University of Pennsylvania

More questions than answers...

... in fact, almost all questions!

- Some familiar (but worthy)
- Hopefully a couple of new ones

What is “the global computer”?

- “the network”?
- the GRID?
- the cell phone network?
- thousands of cars and airplanes running collision-avoidance protocols?
- millions of web browsers running VMs?
- zillions of “informational artifacts”? computational sand? Armies of nano-robots?
- all things to all people?

let's try that again

What is the global computer like?

really **really** **BIG**

What follows from this?

- geographically distributed $\Rightarrow$ locality/latency key issues
- many nodes $\Rightarrow$ constant failures
  temporary failures
  partial/intermittent resource availability
- frequent reconfiguration $\Rightarrow$ uncertainty
- no central coordination $\Rightarrow$ autonomy
  local coordination
- many players $\Rightarrow$ limited trust

What's New

ubiquity of (partial) failures

limited trust

What's FOOLish about this?

First...

What is FOOL?

$$\text{FOOL} \quad \sim \quad \text{FOOL}_\lambda + \text{FOOL}_\forall + \text{FOOL}_\pi$$

- FOOL_λ: semantics of "functional objects"

  pretty well understood

- $\text{FOOL}_\forall$: object types

  active, but its vitality depends on a steady stream of new problems from λ and π

- FOOL_π: foundations of concurrent / distributed objects

  hot...

What is FOOL? (once again)

FOOL $\sim$ foundations of interaction

...focusing on:

- "programmable" models
- compositionality
- reasoning
- static analysis

...Just what GC needs!

Global Computing: Where's the Beef?

Global Applications

- directory/naming services (DNS, LDAP, ...)
- communication (email, SMS, chat, ...)
- the Web
- peer-2-peer services (FreeNet, ...)
- business-2-business (BizTalk?)
- groupware
- [soon] broadcast media distribution (TV, ...)
- ... what else?? (We should not just leave this for other people to figure out!)

Getting down to work...

For the rest of the talk,
let's look at some areas where
global computing raises issues that
might be fun to formlize...

I get around...

Mobile Agents

A long-standing topic for FOOLs and their friends:

many papers, creative, thoughtful proposals,
deep insights, and... few applications

Why?

1. Difficult to find applications where full-blown mobile agents are really useful
2. (Inherently?) hard security problems with sending arbitrary programs to run in enviroments that are neither trusting nor trusted

Mobile Code Today

Still, plenty of code does get shipped across the network. What is it used for?

Mobile Code Today

plug-ins

ActiveX, Netscape, PhotoShop, ...

- arbitrary code
- full control over local machine
- standard (legal) mechanisms used to address security concerns

⇒ "mobile code as a software distribution model"

Mobile Code Today

custom user-interface extensions

Java, JavaScript, ...

- arbitrary code
- very restricted access to local resources
- “call home” access to remote resources
- minimal (but still annoying) security problems

Mobile Code Today

“active data”

Postscript, ...

- specialized programming languages
- highly restricted access to local resources (e.g., just the screen)
- no access to remote resources
- no significant security problems

Mobile Code Today

"rich requests"

SQL

- specialized programming languages
- restricted access to local resources
- "just send back the answer" access to remote resources
- minor security concerns

Mobile Code Tomorrow

Prediction:

- Mobile code will be an increasingly hot application area for domain-specific languages.
- Full-blown mobile agents (“migratory applications”) will remain rare.

Getting the message across...

Communication Primitives

Process-calculi have mostly focused on one case:

- reliable
- location-transparent
- point-to-point

In the real world, reliability and location-transparency are costly (and many applications need less). Can we drop them?

Also, broadcast and multicast are critical in some application domains. Have we forgotten about them?

When bad things happen to good programs...

Failures

Most distributed process calculi ignore failures.

The main exception, DJoin, addresses just "fail-stop" (atomic, permanent, detectable) failures:

But on a "global computer," failures will be

- intermittent ("failed" nodes can reappear)
- partial (services degrade instead of disappearing)

What are the right abstractions for addressing these?
[cf. Luca]

The shape

of things to come...

Types in the Global Computer

XML

- a standard representation for structured data
- values are self-describing
  - actual type can be read off from tags in structure
  - expected type can be
    - included in structure, or
    - known in advance (but must be checked)

⇒ Mix of static and dynamic checking

XML Types

Many "schema languages" (type systems) for XML

- DTDs, XML-Schema
- Regular expression types [Hosoya/Pierce/Vouillon]
- Relax [Murata]
- type-indexed products [Meijer/Shields]
- etc.

All are (more or less fancy) regular tree languages over simple base types.

Is this everything we need?

What about Polymorphism?

Polymorphism doesn't work very well in this setting: we cannot rely on the fact that a remote "polymorphic function" is unable to inspect the concrete values that we send it.

Polymorphism via Encryption

$$\frac{\begin{array}{c}\text{polymorphism} = \text{information hiding} \\ \text{encryption} = \text{information hiding}\end{array}}{\text{polymorphism} = \text{encryption ??}}$$

Recent work by Sumii explores this connection...

Example

pack (1,

$\lambda x.\ x$ mod 2)

as $\exists$X. X $\times$ (X $\rightarrow$ Int)

$\Rightarrow$

let k = ⟨new key⟩ in

(⟨encrypt 1 under k⟩,

$\lambda y.$ let x = ⟨decrypt y using k⟩ in

x mod 2)

More generally...

This example can be extended to an encoding from full System F into a simply typed lambda-calculus with cryptographic primitives.

The translation is appealing, but its properties are still being studied.

"Polymorphism in the large"?

Life in the Jungle...

Security

Mechanisms for establishment of trust, verification of authenticity, protection of privacy, etc., are clearly a precondition for global programming.

(Unclear what is a good level of abstraction [cf. Jim].)

Low-level concepts (crypto) beginning to be understood in the PL community

Higher-level concepts (identity, trust, authenticity, ...) not well understood or integrated with other PL ideas.

(To be fair, it may simply be too early for this)

An Ugly Little Secret...

Resource Management

Current distributed process calculi create global resources (e.g., names) with wild abandon and don't worry **at all** about garbage collection.

On the global computer, this is not going to fly.

For some hints at solutions, look at emerging web-scripting technologies — e.g., Microsoft's ASP.net, in which "session state" can be transparently

- kept in an active server thread
- stored in a server database
- threaded back through the client

To thine own self be true...

Competitive agents

 small concurrent system

$\Longrightarrow$ single trust domain

$\Longrightarrow$ cooperative organization

 big distributed system

$\Longrightarrow$ multiple trust domains

$\stackrel{??}{\Longrightarrow}$ competitive organization

Is competition (markets, etc.) a natural way of organizing robust global computations?

How about a "Calculus of Competing Agents"?

Let's call the whole thing off...

[Thanks to Peter Sewell, Cédric Fournet, and many others...]

Transactions

In the real world, transactions are what make it possible for ordinary people to write distributed applications that work.

"The transaction concept is the computer equivalent to contract law."

[Gray and Reuter]

Claim:

A successful global programming language **must** provide (some mechanism like) transactions.

The standard story

"Informally, an atomic transaction appears to occur indivisibly (with respect to failures and to other concurrent transactions). This means that the programmer can view an atomic transaction as a sequential program, even though it runs concurrently with other transactions and failures can occur while it is running.

— W. Weihl

Another Perspective

Atomic transactions are a structuring technique for fault-tolerant concurrent systems that comes with a proof technique in which the proof obligations for "client code" are extremely straightforward.

Another Perspective

If

1. the database implementation obeys some hideous property [typically hard to show]
2. any series of completely successful "client transactions," each running with sole access to the DB, yields a good behavior [typically obvious]

then the whole system (many clients concurrently interacting with DB, with possible failures) always behaves well.

Challenge for FOOLs

What would it mean to "program transactionally" in, say, the pi-calculus?

One “Answer”

The pi-calculus is a general-purpose programming language.

$\Rightarrow$ We can use it to implement a transactional database.

Duh.

Better(?) Questions

- What does one transactional object look like to another?
- Is serializability related in some way to causality?
- What would it be to offer primitives for general purpose transactional programming in the pi-calculus...
- By the way, what is "general-purpose transactional programming"??

Needed: a really crisp question!

Need to look to the distributed systems community for insights!

Transactions vs. Locations

A transaction is (among other things) a unit of failure.

A location is (among other things) a unit of failure.

Hmmm.

Conclusion

A FOOL's work is never done.