# What's in a name?

(in honour of Roger Needham)

Robin Milner

January 2003

In the late eighties Roger Needham wrote a paper called 'Naming', which is now a chapter in a leading text on distributed systems.[1] The paper highlights some subtleties of naming, and points out how these can either illuminate or confuse system design. Around the same time colleagues and I worked out the $\pi$-calculus[2], a calculus for mobile systems intended for modelling and analysis. Names are the most prominent feature in the $\pi$-calculus, and in this essay I explain in simple terms how it deploys them.

Some things about names are so buried in our linguistic habits that we hardly ever talk about them. Roger talked about one of them: the difference between pure and impure names. To paraphrase him, a *pure* name is nothing but an identifier or pointer; you can follow the pointer, but otherwise you can only test it for equality with another one. A name is *impure* to the extent that you can do other things with it. You can resolve it into parts, or you can take advantage of your knowledge about the thing that it designates; an email address like `Robin.Milner@cl.cam.ac.uk` illustrates both of these.

We also habitually assume that a name designates something with persistent identity. This assumption works well for us in sequential programming: a pointer designates a storage cell, and a procedure identifier designates a piece of code. It doesn't work reliably in distributed systems. Consider a call-centre; on each call you get someone different. Consider an e-mail message to `Robin.Milner@cl.cam.ac.uk`; it may go to me, or to an agent to which I (on holiday) have delegated the power to respond.

The $\pi$-calculus is built upon the idea that the respondent to (or referent of) a name exists no more persistently than a caller of the name. In other words, the notions of *calling* and *responding* are more basic than the notions of *caller* and *respondent*; every activity contains calls and responses, but to have a *persistent* respondent to $x$ – one that responds similarly to every call on $x$ – is a design choice that may be sensible but is not forced.

What follows is a taxonomy of the small range of things you can do with names in the $\pi$-calculus. At the end I speculate on whether these are enough.

**Using and mentioning names** The logician W.V.Quine discussed the distinction between the use and mention of names. In natural language, a name is *used* when something is intended of the referent, *mentioned* when intended of the name itself; further, a *use* can be imperative (an invocation), or indicative (an assertion). In the $\pi$-calculus we

[1] **Distributed Systems** (second edition), ed. Sape Mullender, Addison-Wesley (1993)

[2] A calculus of mobile processes, R.Milner, J.Parrow, and D.Walker; Information and Computation 100 (1) 1992, pp1–77.

only have imperative use, and what it intends is an act. But we distinguish between a *call* act and a *response* act, even though one cannot occur without the other. The reason to distinguish them is that, in describing any agent, we define its *potential* behaviour: what calls/responses it can make, provided that its environment makes homonymous responses/calls. Here is a *call* on $x$, mentioning $y$:

$$\overline{x}\langle y\rangle.P$$

which could be pronounced '$x$, here is $y$; now I'll do $P$'. Superficially it is like 'John, here is Stephen'; actually, it corresponds to 'John, here is (the name) "Stephen" '. It is just a message with address $x$ and content $y$; we can call this *quoting* $y$.

Here is a *response* on $x$, mentioning $z$:

$$x(z).Q$$

which could be pronounced '$x$, thanks for $z$; now I'll do $Q$ with it'. We can see how calls and responses are dual. Following the mathematical convention of 'co-' for a dual, we can say that the response is *co-quoting* $z$, because $z$ acts as a place-holder in $Q$ for a name quoted by a call.

In fact the only rule of action in the $\pi$-calculus is that, when a call may concur with a homonymous response, as in

$$\overline{x}\langle y\rangle.P \mid x(z).Q\ ,$$

then they are fused together; thereafter $P$ and $Q$ happen concurrently, with $y$ occupying the place in $Q$ held by $z$.

It is better to think of 'response on $x$' rather than 'respondent designated by $x$', because there need be no agent identifiable as respondent. The power to *respond* on a name can be delegated or duplicated (consider the call-centre), just as the power to *call* on a name can be so. For example, in the above rule of action, if $Q$ happens to contain a response on the place-holder $z$, then the call that quoted $y$ has delegated to $Q$ the power to respond on $y$.

**Creating names** So far we have only talked about use and mention. But where do all the names come from? How can we represent the very specific mechanisms (e.g. time-stamps) that allow a system to create names which it can safely assume to differ from all other names?

The $\pi$-calculus does this by fiat. It has a name-creator new that is *assumed* to create a globally distinct name. In some eyes this is cheating; in other eyes it isolates the implementer's problem of creating new names in practice from the analyst's task of explaining how a system works *assuming* generated names are unique. Here is an example of unique name creation:

$$(\mathsf{new}\ z\ P) \mid Q\ ;$$

it creates $z$ local to $P$. Whatever $P$ does, this name remains different from any name occurring in $Q$ –or in the wider environment– even if such a name is textually identical with $z$, and even if $P$ mentions its new $z$ to $Q$.

We can illustrate new with a simple example: simulating a function call. The $\pi$-calculus has no built in notion of call-and-return, but if a process calls on $x$ quoting $y$, then it can simultaneously create a private channel $res$ and pack it up with $y$ in the call; thereafter it can respond on $res$ to receive the result that comes back. This call-and-return action is defined by:

$$\mathsf{new}\ res\ (\ \overline{x}\langle y, res\rangle \mid res(z).Q\ )\ .$$

(A multiple quotation, such as $\langle y, res\rangle$, can easily be coded in the $\pi$-calculus.) The creation of $res$ ensures its distinction from every other return-address. This little sequence is very commonly used, so we shall abbreviate it to

$$\overline{x}\langle y\rangle \Rightarrow (z).Q\ .$$

**Matching names** So far we have seen only one way to mention a name: quoting it in a call (or co-quoting in a response). Surprisingly, with a few control mechanisms this is enough to model all computation! Nonetheless, it does not give the direct facility to 'test a name for equality with another name'. So there is a second way to mention names: *matching*. With (only) these two kinds of mention, the $\pi$-calculus can much more directly model the handling of names in real systems.[3]

Matching in the $\pi$-calculus can done by the construction

$$[x = y]\, P\, /\, Q$$

meaning 'if $x$ and $y$ are the same name then do $P$, else do $Q$'. (It matches names, *not* their referents, because referents need not exist.)

In the context of $\pi$-calculus we can illustrate how directory lookup can be handled, following closely how Roger Needham illustrates it. A hierarchical directory – say the one containing the graduate students at Wolfson College, Cambridge – typically has a composite name like `Wolfson/Grads`. It is not a unique designator; there will be a directory with this name at Oxford too, because both Oxford and Cambridge have a Wolfson College. However, usable systems will ensure that each directory and subdirectory will also have a unique *directory identifier* (DI) which is a pure name.

If I know the DI of Cambridge University, I can access the University's main directory and then use a composite name like `Wolfson/Grads` –or extensions of it– to get to all its subdirectories, even if I don't have *their* DIs. For example, suppose I want to get hold of (the DI of) `Smith-J` at Wolfson College Cambridge. If the DI of the Cambridge University directory is #312, then I can get to where I want (without knowing any other DIs) by a composite call as follows:

$$\overline{\#312}\ \langle \mathtt{Wolfson}, \mathtt{Grads}, \mathtt{Smith\text{-}J}\rangle \Rightarrow (di).Q.$$

This call-and-return will cause the required DI to occupy the place held in $Q$ by $di$. To make this happen, the directory itself can be defined with matching like this:

[3] In applied languages built upon the $\pi$-calculus, there can of course be impure names like 23 which designate known entities, operations on them like $+$ and $\times$, and variables or place-holders $a, b, c \ldots$ for them. With appropriate type discipline, this doesn't impair the rigorous handling of pure names.

$$
\begin{array}{l}
\mathbf{!}\,\#312\,(college, group, person). \\
\qquad [college = \texttt{Trinity}]\ \overline{\#427}\,\langle group, person\rangle \\
\qquad / [college = \texttt{Wolfson}]\ \overline{\#203}\,\langle group, person\rangle \\
\qquad / \ldots / \ldots
\end{array}
$$

where #427 and #203 are the DIs of `Trinity` and `Wolfson`. Thus a matching occurs at each level. Notice that there is only one kind of pure name. We chose to write *college*, `Trinity` and #427 differently because we treat them differently; for example, we never *use* the first two, but only *mention* them.

Finally, you may have noticed the new operator '**!**' in the above code for a directory. It is a *replicator*; it gives persistent identity to the respondent that it qualifies, making it a re-usable resource. So in this case the pure name #312 *does* designate a persistent agent: the Cambridge University Directory.

**What else is in a name?** We have illustrated *use* (*call*, *response*), *mention* (*quote*, *co-quote*, *match*) and *creation* of names. That is all the $\pi$-calculus can do with them. Are there other things it might do?

I have not said anything so far about computer security, which has in fact been a main application of process calculi that use names. Another influential paper by Roger Needham and his co-authors[4] has inspired much of the recent logical work on security, authentication and associated topics, and under this heading come many approaches using process calculi that use names whose scope may be controlled (for example by new in the $\pi$-calculus). A leading example is the *spi calculus* of Abadi and Gordon[5] which is largely based upon the $\pi$-calculus, but uses extra features for encryption and decryption.

These extra features allow the spi calculus to represent security protocols very directly, and have led to powerful analytical studies. But there is a theoretical question that hasn't been fully answered as far as I know: in what rigorous sense do they extend ther expressive power of the $\pi$-calculus? It would be illuminating to prove that the extra features *can*, or that they *cannot*, be mimicked in the $\pi$-calculus in some exact sense.

More generally, if we suspect that the $\pi$-calculus *can't* do something that *can* be done with pure names, then where could we look for the weakness? A more powerful form of *use* of names might have something to do with synchronisation. The $\pi$-calculus only ever synchronises a pair of actions, one call and one response. What about synchronising two (or more) calls with a single response? The calls could be on two distinct names $x_1$ and $x_2$, and the response on both of these names simultaneously. So our rule of action would be strengthened to synchronise these three actions:

$$\overline{x_1}\langle y_1\rangle.P_1 \mid \overline{x_2}\langle y_2\rangle.P_2 \mid x_1x_2(z_1z_2).Q$$

[4] Burrows, M., Abadi, M. and Needham, R., A logic of authentication. Proc. Royal Society of London A, 426:233–271, 1989.

[5] Abadi, M. and Gordon, A.G., A calculus for cryptographic protocols: The spi calculus. Information and Compuitation 148:1–70, 1999.

– causing $y_1$ and $y_2$ simultaneously to occupy the places held in $Q$ by $z_1$ and $z_2$, and then $P_1$, $P_2$ and $Q$ to proceed concurrently. Can this be mimicked in the $\pi$-calculus? What exact meaning would 'mimicked' have here?

Such theoretical questions may seem arcane. They certainly should not distract us from applying process calculi to security (or to anything else). But they have their own charm, and the better we can answer them, the more confident we can be of finding good primitives for expressing and analysing mobile communication.