To: *'GC Non-Classical Computation Workshop'*, York, 18-19 April, 2005
Title: *'Mobile Processes: very tiny, very fast and very many'*
From: Peter Welch, University of Kent Computing Laboratory

Through the Concurrency Gateway:

The real world exhibits concurrency at all levels of scale – from atomic, through human, to astronomic. This concurrency is endemic. Central points of control do not remain stable for long. Most of the novel paradigms identified in the GC7 description paper hint at something stronger – namely that central points of control actively work against the logic and efficiency of whatever it is that is we are trying to control/model/understand. To overcome this, we expect concurrency to play a fundamental rôle in the design and implementation of systems, reflecting the reality of the environment in which they are embedded. This does not currently seem to be the case, where concurrency appears hard to deploy safely and effectively and is, therefore, turned to only when absolutely necessary (e.g. to meet some performance requirement). We need to break through this mind block.

Unlike systems developed for traditional embedded and parallel supercomputing applications, our ambitions are for fine-grained, highly structured and highly dynamic networks — with elements (such as channels, barriers and processes) growing and decaying in reaction to environmental and internal pressures. We want millions (later, billions) of such elements. We want continually evolving system topology as higher level organisms (whether designed or emergent) are born, combine, decay and die. Then, we will be free.

Process Algebra and occam-π:

The analysis, certification, and verification of concurrent programs must depend on a sound theory of their behaviour. For programs expressed in the notations of a formal process algebra, much of the theory is well developed, and some of it is already incorporated in verification tools, for example, the Concurrency Workbench[1] or FDR[2]. But these need to be integrated with analysis of sequential processes, as for example in the old programming language occam[3] founded on the CSP[4, 5] process algebra. Some of the necessary unifying theories have been developed and tested, but without the benefit of verification tools.

The original occam language and its implementation on the transputer gained its strength, its efficiency and its simplicity by limiting its expressions to sets of statically determined networks of processes and communication channels. Rejection of this restriction was the guiding motive in the design of the π-calculus[6], which used dynamic communication of channel names to model mobile processes and the evolution of their network topology. The π-calculus by design is fully expressive of many of the things that can go wrong in real networks, as it has to be if it is to provide the means of proving the absence of anomalies. Much research has been done on type systems that permit such proofs to be done cheaply by a verifying compiler.

Over the last ten years, our group[7] at Kent has been engaged in research aimed at providing languages, libraries and environments which might be suitable for highly reactive and dynamic systems on the scale envisaged. Overviews of some of this work (on JCSP and occam-π) can be found at [8] and below. It is based on ideas from process algebra [4, 5, 6] that the concept of a *process* – and of *communication*, *concurrency* and *mobility* – are uniformly applicable at any level of granularity; and that they are logically independent of the actual distribution of the program over different computer architectures and network configurations. This uniformity of concept could contribute to simplicity of structure and understanding of multi-level simulation programs applied in biology and elsewhere; and it would make the programs independent of changes and improvements to the underlying architecture of the computer network.

Twenty years ago, improved understanding and architecture independence were the goals of the design by Inmos of the occam programming language and the transputer. The goals were achieved by implementation of the abstract primitives and operators of process algebra, and with an efficiency which is today almost unimaginable and certainly unmatchable. We have been extending the classical occam language with ideas of mobility and dynamic network reconfiguration [9-13], which are taken from the π-calculus. We have found ways of implementing these extensions that involve significantly less resource overhead than that imposed by the rather less structured concurrency primitives of existing languages like Java. As a result, we can run simulations of the order of millions of processes on modern PCs (3 GHz). We have plans to extend the system without sacrifice of too much efficiency (and none of logic) to networks such as the Grid. The new language is christened occam-π.

In the interests of proveability, we have been careful to preserve the distinction between the original simple static point-to-point synchronised communication of occam and the dynamic asynchronous multiplexed communication of the π-calculus; in this we have been prepared to sacrifice the elegant sparsity of the π-calculus. We conjecture that the extra complexity (and discipline) introduced will make the task of proving concurrent and distributed programs easier. Nevertheless, our practice has a tendency to race ahead of the necessary formal semantics – we are always receptive to criticism and/or help in ensuring that the theory *can* and does catch up (i.e. that we have not made any bad mistakes).

Brief Overview of occam-π:

occam-π is a sufficiently small language to allow experimental modification and extension, whilst being built on a language of proven industrial strength. It integrates the best features of CSP and the π-calculus, focussing them into a form whose semantics is intuitive and amenable to everyday engineering by people who are not specialised mathematicians – the mathematics is built into the language design, its compiler, run-time system and tools (so that users benefit automatically from that foundation). The new dynamics broadens its area of direct application to a wide field of industrial, commercial and scientific practice.

occam-π runs on modern computing platforms and has much of the flexibility of Java and C, whilst at the same time retaining all the safety guarantees of *classical* occam (e.g. against aliasing and parallel usage errors) and the lightness of its concurrency mechanisms. It supports the dynamic construction of processes, data, channels and barriers, their *movement* across channels and their automatic de-allocation (without the need for garbage collection) [9-13]. We have extended the range of static safety checks so that aliasing errors and race hazards are not possible in occam-π systems, despite the new dynamics. This means that subtle side-effects between component processes cannot exist, which impacts (positively) on the general scalability and dependability of systems. The *mobility* and *dynamic construction* of processes, channels etc. opens up a wealth of new design options that will let us follow nature more closely – with network structures *evolving* at run-time. Apart from the logical benefits derived from such directness and flexibility, there will be numerous gains for application efficiency.

Multiway barrier synchronisation is a recent addition to occam-π. Barriers correspond to primitive CSP events, though some higher level design patterns (such as *resignation*) have been built in. Barriers can also be *mobile* so that regions (or layers) of mobile processes coordinated by the barriers can grow, as well as shrink. Basic CSP semantics apply: when a process *synchronises* on a barrier, it is blocked until all other processes enrolled on the barrier have also *synchronised*. Once the barrier is completed, all blocked processes are rescheduled. Run-time costs, including cache-miss penalties from context switching, are extremely low – around 15 nanoseconds per process per synchronisation (for a 3.2 GHz Pentium 4). Barriers have many uses at many levels of parallel granularity. On high, there is Valiant's BSP[14] (*Bulk Synchronous Parallelism*) for distributed supercomputing. We are particularly interested in experimenting with multiple barriers operating at finer levels (for the coordination of specialist regions / layers of behaviour).

Performance overheads for all occam-π concurrency mechanisms are mostly unit time, with the order of between 10 and 100 *nanoseconds* on modern PCs. Memory overheads are also very light: no more than 8 words per process. This means that dynamic systems evolving millions of processes are already practical on single processors. Those processes can be implementing complex behaviour with time and space overheads for managing the concurrency minimal (less than 10%). Further, occam-π networks can naturally span many machines – the concurrency model does not change between internal and external concurrency. Application networks up to tens of millions of processes then become viable (e.g. on modest PC clusters). The continuing progress of Moore's Law likely over the next few years means that networks of hundreds of millions of processes will become possible.

The TUNA Project:

TUNA (*Theories Underpinning Nanite Assemblies*) is an EPSRC funded 2-year project, started in December 2004. Collaborators are York, Surrey and Kent universities. The TUNA project aims to investigate the emergent properties of systems containing millions of interacting agents — nanites (artificial mechanisms of nanometre scale) or biological organelles (such as blood platelets). Here, goals are achieved by emergent behaviour from force of numbers, not by complicated programming or external direction. Such systems are complex, but not complicated.

The plan is to develop design rules, backed up by formal theory spanning all levels of scale, for the safe management of such systems, with a view to the safe deployment of nanite technologies in human medicine.

To model more directly (and, hence, simply) the underlying biological/mechanical processes, extremely fine-grained concurrency will be used. Complex behaviour will be obtained not by direct programming of individual process types, but by allowing maximum flexibility for self-organisation following randomised encounters between mobile processes — modulo physical constraints imposed by their modeled environments. We will need to develop location awareness for the lowest level processes, so they may discover other processes in their neighbourhood and what they have to offer. We will need to synchronise the development of organisms to maintain a common awareness of time.

Our ambitions in the TUNA project call for scaling of these models up to tens of millions of processes and hard-to-quantify orders of complexity. We will need to model (and visualise) multiple dimensions, factor in a mass of environmental stimulators and inhibitors and distribute the simulation efficiently over many machines (to provide sufficient memory and processor power).

Simple cellular automata will not be sufficient. Lazy versions will be needed, in which cells that are inactive make no demands on the processor. We also need to concentrate our modeling on processes that directly represent nanites/organelles and environmental factors – all of which are mobile and may attach themselves to particular cells in space (which can be modeled as passive server processes that, probably, do not need to be time-synchronised). Barrier *resignation* will be crucial to manage this laziness; but care will need to be applied to finding design patterns that overcome the *non-determinism* that arises from unconstrained use.

Achieving this will be a strong testing ground for the dynamic capabilities (e.g. mobile processes, channels and barriers) built into the experimental language, its compiler and runtime kernel. occam-π offers support for our required scale of parallelism and relevant concurrency primitives (backed up with compiler-checked rules against their misuse). We need the very high level of concurrency to give a chance for interesting complex behaviour to emerge that is not pre-programmed. We need to be able to capture rich emergent behaviour to investigate and develop the necessary theories to underpin the safe deployment of *nanite* technology in medicine and elsewhere.

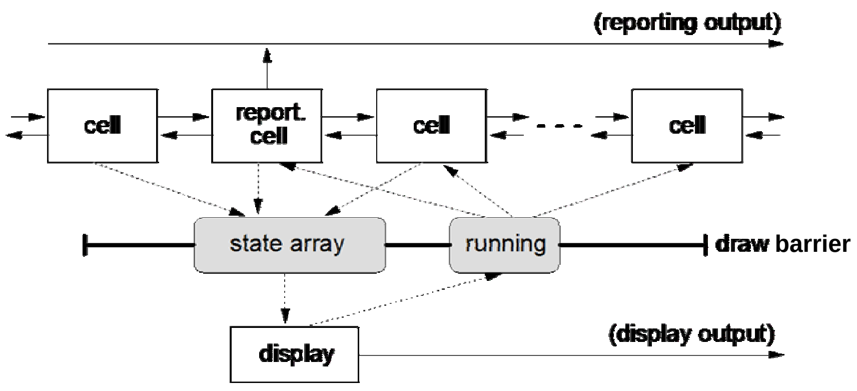
Developing formal theories of emergent behaviour and relating them to the process algebrae underlying occam-π semantics (i.e. Hoare's CSP and Milner's π-calculus) is a very interesting and very open question. This work will contribute to the (UK) 'Grand Challenges for Computer Science' areas 1 (*In Vivo – In Silico*) and 7 (*Non-Standard Computation*).

References and URLs:

[1] *The Edinburgh Concurrency Workbench.* www.dcs.ed.ac.uk/home/cwb/. 2004.
[2] Formal Systems (Europe) Ltd. *FDR Home Page*. www.fsel.com. 2004.
[3] Inmos Ltd. *occam2.1 Language Reference Manual*. wotug.kent.ac.uk/parallel/occam/documentation/
[4] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985.
[5] A. Roscoe, *The Theory and Practice of Concurrency*. Prentice Hall, 1997, ISBN: 0-13-674409-5.
[6] R. Milner. *Communicating and Mobile Systems: the π-Calculus*. Cambridge University Press, 1999.
[7] P.H. Welch et al. *Concurrency Research Group*. www.cs.kent.ac.uk/research/groups/crg/. 2004.
[8] P.H. Welch. *IFIP WG2.4 (Peter Welch's page)*. www.cs.kent.ac.uk/projects/ofa/ifip/. 2004.
[9] P.H. Welch and F.R.M. Barnes. *KRoC Home Page*. www.cs.kent.ac.uk/projects/ofa/kroc/. 2004. See also http://frmb.org/kroc.html (latest pre-releases) and http://frmb.org/occ21-extensions.html (summary of new language elements).
[10] P.H. Welch and F.R.M. Barnes. *Prioritised Dynamic Communicating and Mobile Processes*. IEE Proceedings Software, 150(2), April 2003. www.iee.org/Publish/Journals/ProfJourn/Proc/SEN/Preprints.cfm
[11] P.H. Welch and F.R.M. Barnes. *Mobile Data, Dynamic Allocation, and Zero Aliasing: an occam Experiment*. CPA 2001, *Concurrent Systems Engineering Series* (59) pp. 243-- 264. IOS Press, 2001.
[12] M. Schweigler, F.R.M. Barnes and P.H. Welch. *Flexible, Transparent, and Dynamic occam Networking with KRoC.net*. Communicating Process Architectures 2003, *Concurrent Systems Engineering Series* (61) pp. 199-- 224. IOS Press, 2003.
[13] P. H. Welch and F.R.M. Barnes, *Communicating mobile processes: introducing occam-pi,* in '*25 Years of CSP',* A. Abdallah, C. Jones, and J. Sanders, Eds., *Lecture Notes in Computer Science*, vol. 3525. Springer Verlag, Apr. 2005, pp. 175–210, to appear.
[14] L. Valiant, *A Bridging Model for Parallel Computation*, Communications of the ACM, vol. 33, no. 8, pp. 103–111, Aug. 1990.

Footnote (network diagrams):

Designing/implementing occam-π systems is about diagrams and mathematics. The diagram below is *part* of a system for modeling and visualizing the circulation of platelets in a blood stream. Shown is a pipeline of 'cell' processes, representing the stream, with some reporting cells logging statistics to a shared channel. The cells also coordinate with a 'display' process through barrier synchronization on the 'draw' barrier, which also orchestrates the safe sharing of state between 'display' and the 'cell's. 'Platelet' processes (not shown) drift through the cell pipeline, where they become influenced by environmental factors (such as oxygen partial pressures, chemical stimulators and inhibitors) and bump into each other. Environmental factors are updated by other mobiles also wandering around.


(reporting output)
cell
report. cell
cell
- - -
cell
state array
running
draw barrier
display
(display output)