Journeys in Non-Classical Computation

A Grand Challenge for Computing Research

18 May 2004

Susan Stepney, Samuel L. Braunstein , John A. Clark, Andy Tyrrell : University of York

Andrew Adamatzky, Robert E. Smith : University of the West of England

Tom Addis : University of Portsmouth

Colin Johnson, Jonathan Timmis, Peter Welch : University of Kent

Robin Milner, University of Cambridge

Derek Partridge : University of Exeter

The Challenge

A *gateway event* [34] is a change to a system that leads to the possibility of huge increases in kinds and levels of complexity. It opens up a whole new kind of phase space to the system's dynamics. Gateway events during evolution of life on earth include the appearance of eukaryotes (organisms with a cell nucleus), an oxygen atmosphere, multi-cellular organisms, and grass. Gateway events during the development of mathematics include each invention of a new class of numbers (negative, irrational, imaginary, …), and dropping Euclid's parallel postulate.

A gateway event produces a profound and fundamental change to the system: once through the gateway, life is never the same again. We are currently poised on the threshold of a significant gateway event in computation: that of breaking free from many of our current "classical computational" assumptions. The Grand Challenge for computer science is

to journey through the gateway event obtained by breaking our current classical computational assumptions, and thereby develop a mature science of Non-Classical Computation

Journeys versus Goals

To travel hopefully is a better thing than to arrive.
– Robert Louis Stevenson, "El Dorado", 1878.

Many Grand Challenges are cast in terms of *goals*, of end points: "achieving the goal, before this decade is out, of landing a man on the moon and returning him safely to earth" [49], mapping the human genome, proving whether **P** = **NP** or not. We believe that a goal is not the best metaphor to use for this particular Grand Challenge, however, and prefer that of a *journey*.

The metaphor of a journey emphasises the importance of the entire process, rather than emphasising the end point. In the 17th and 18th centuries it was traditional for certain sections of "polite society" to go on "a Grand Tour of Europe", spending several years broadening their horizons: the experience of the entire journey was important. And in the Journey of Life, death is certainly not the goal! Indeed, an open journey, passing through gateway events, exploring new lands with ever expanding horizons, need not have an end point.

A journey of a thousand miles begins with a single step.
– Lao Tzu, *Tao Te Ching*, Chapter 64, ~600 B.C.

Journeys and goals have rather different properties. A goal is a fixed target, and influences the route taken to it. With an open journey of exploration, however, it is not possible to predict what will happen: the purpose of the journey is discovery, and the discoveries along the journey suggest new directions to take. One can suggest starting steps, and some intermediate *way points*, but not the detailed progress, and certainly not the end result.

Thinking of the Non-Classical Computation Challenge in terms of a journey, or rather several journeys, of exploration, we suggest some early way points that appear sensible to aim for. But we emphasise that these *are* early points, that we spy

today as we peer through the gateway. As the community's journey progresses, new way points will heave into view, and we can alter our course to encounter these as appropriate.

The Road goes ever on and on.
– J. R. R. Tolkien, *The Lord of the Rings*, 1954.

Six classical paradigms to disbelieve before breakfast

Classical computing is an extraordinary success story. However, there is a growing appreciation that it encompasses an extremely small subset of all computational possibilities.

In many avenues of life, we create unnecessary limitations. Perhaps the most invidious of these are the implicit assumptions we make. We need to distinguish *this has to be the case* from the merely *this has always been the case*. Discoveries may emerge when what was considered an instance of the former is found to be an instance of the latter. For example, dropping Euclid's parallel postulate gave rise to the whole field of non-Euclidean geometry, arguably paving the way for General Relativity. We wish to encourage similar revolts against the assumptions of classical computing. So below we identify several paradigms that seem to define classical computing, but that may not necessarily be true in all computing paradigms, and we encourage the community to drop, invert, or otherwise perturb these paradigms in whatever ways seem interesting. Our brochure of reality-based journeys is a start.

Many computational approaches seek inspiration in reality (mainly biology and physics), or seek to exploit features of reality. These *reality-based computing* approaches hold great promise. Often, nature does it better, or at the very least differently and interestingly. Examining how the real world solves its computational problems provides inspirations for novel algorithms (such as genetic algorithms or artificial immune systems), for novel views of what constitutes a computation (such as complex adaptive systems, and self-organising networks), and for novel computational paradigms (such as quantum computing).

There is a gulf between the maturity of classical computing and that of the emerging non-classical paradigms. For classical computing, intellectual investment over many years is turning craft into science. To fully exploit emerging non-classical computational approaches we must seek for them such rigour and engineering discipline as is possible. What that science will look like is currently unclear, and the Grand Challenge encourages exploration.

Here we outline some assumptions of classical computation, and ways researchers in different fields are challenging them. In later sections we discuss alternatives in more detail. (Some of the categories arguably overlap.)

It ain't necessarily so.
– George Gershwin, *Porgy and Bess*, 1934

1: The Turing paradigm

classical physics: information can be can be freely copied, information is local, states have particular values. *Rather*, at the quantum level information cannot be cloned, entanglement implies non-locality, and states may exist in superpositions.

atomicity: computation is discrete in time and space; there is a before state, an after state and an operation that transforms the former into the latter. *Rather*, the underlying implementation substrate realises intermediate physical states.

infinite resources: Turing machines have infinite tape state, and zero power consumption. *Rather*, resources are always constrained.

substrate as implementation detail: the machine is logical, not physical. *Rather*, a physical implementation of one form or another is always required, and the particular choice has consequences.

universality is a good thing: one size of digital computer, one size of algorithm, fits all problems. *Rather*, a choice of implementation to match the problem, or hybrid solutions, can give more effective results.

closed and ergodic systems: the state space can be pre-determined. *Rather*, the progress of the computation opens up new regions of state space in a contingent manner.

2: The von Neumann paradigm

sequential program execution. *Rather*, parallel implementations already exist.

fetch-execute-store model of program execution. *Rather*, other architectures already exist, for example, neural nets, FPGAs.

the static program: the program stays put and the data comes to it. *Rather*, the data could stay put and the processing rove over it.

3: The output paradigm

a program is a black box: it is an oracle abstracted away from any internal structure. *Rather*, the trajectory taken by a computation can be as interesting, or more interesting, than the final result.

a program has a single well-defined output channel. *Rather*, we can chose to observe other aspects of the physical system as it executes.

a program is a mathematical function: logically equivalent systems are indistinguishable. *Rather*, correlations of multiple outputs from different executions, or different systems, may be of interest.

4: The algorithmic paradigm

a program maps the initial input to the final output, ignoring the external world while it executes. *Rather*, many systems are ongoing adaptive processes, with inputs provided over time, whose values depend on interaction with the open unpredictable environment; identical inputs may provide different outputs, as the system learns and adapts to its history of interactions; there is no prespecified endpoint.

randomness is noise is bad: most computer science is deterministic. *Rather*, nature-inspired processes, in which randomness or chaos is essential, are known to work well.

the computer can be switched on and off: computations are bounded in time, outside which the computer does not need to be active. *Rather*, the computer may engage in a continuous interactive dialogue, with users and other computers.

5: The refinement paradigm

incremental transformational steps move a specification to an implementation that realises that specification. *Rather*, there may be a discontinuity between specification and implementation, for example, bio-inspired recognisers.

binary is good: answers are crisp yes/no, true/false, and provably correct. *Rather*, probabilistic, approximate, and fuzzy solutions can be just as useful, and more efficient.

a specification exists, either before the development and forms its basis, or at least after the development. *Rather*, the specification may be an emergent and changing property of the system, as the history of interaction with the environment grows.

emergence is undesired, because the specification captures everything required, and the refinement process is top-down. *Rather*, as systems grow more complex, this refinement paradigm is infeasible, and emergent properties become an important means of engineering desired behaviour.

6: The "computer as artefact" paradigm

computation is performed by artefacts: computation is not part of the real world. *Rather*, in some cases, nature "just does it", for example, optical Fourier transforms.

the hardware exists unchanged throughout the computation. *Rather*, new hardware can appear as the computation proceeds, for example, by the addition of new resources. Also, hardware can be "consumed", for example, a chemical computer consuming its initial reagents. In the extreme, nanites will construct the computer as part of the computation, and disassemble it at the end.

the computer must be on to work. *Rather*, recent quantum computation results [46] suggest that you don't even need to "run" the computer to get a result!

Doubtless there are other classical paradigms that we accept almost without question. They too can be fruitfully disbelieved.

The Real World : breaking the Turing paradigm

Real World as its own computer

The universe doesn't need to calculate, it just does it. We can take the *computational stance*, and view many physical, chemical and biological processes *as if* they were computations: the Principle of Least Action "computes" the shortest path for light and bodies in free fall; water "computes" its own level; evolution "computes" fitter organisms; DNA and morphogenesis "computes" phenotypes; the immune system "computes" antigen recognition.

This natural computation can be more effective than a digital simulation. Gravitational stellar clusters do not "slow down" if more stars are added, despite the problem appearing to us to be $O(n^2)$. And as Feynman noted [30], the real world performs quantum mechanical computations exponentially faster than can classical simulations.

Real World as our computer

Taking the computational stance, we may exploit the way the world works to perform "computations" for us. We set up the situation so that the natural behaviour of the real world gives the desired result.

There are various forms of real world sorting and searching, for example. Centrifuges exploit differences in density to separate mixtures of substances, a form of *gravitational sorting*. Vapours of a boiling mixture are richer in the components that have lower boiling points (and the residual mixture is richer in those that have higher boiling points); distillation exploits this to give a form of *thermal sorting*. Chromatography provides chemical means of separation. Ferromagnetic objects can be separated out from other junk by using industrial-strength magnets. Optics can be exploited to determine Fourier transforms.

Maggots perform the "computation" of eating dead flesh: historically, maggots were used to clean wounds, that is, to perform their computation in a context to benefit us. More recently, bacterial metabolisms have been altered to perform the "computation" of cleaning up pollution.

Access control computations abound. Suitably constructed shape is used to calculate whether the key inserted in a tumbler lock is the correct one. Physical interlocks are exploited for safety and practical reasons across many industries: for example, it is impossible to insert a nozzle from a leaded petrol pump into the fuel tank of a unleaded petrol car.

Real World as analogue computer

We may exploit the real world in more indirect ways. The "computations" of the "real world as our computer" are very direct. Often we are concerned with more abstract questions. Sometimes the physical world can be harnessed to provide results that we need: we may be able to set up the situation so that there is an *analogy* between the computation performed by the real world, and the result we want.

There is an age-old mechanism for finding the longest stick of spaghetti in an unruly pile, exploiting the physics of gravity and rigidity: we can use this to sort by setting up an analogy between spaghetti strand length and the quantity of interest. Mercury and alcohol thermometers use a physical means of computing temperature by fluid expansion: the analogy is between the length of the fluid column and the temperature. Millikan's calculation of the charge on an electron exploits relationships between velocity of falling oil drops, viscosity of air, the charge on those drops, and the strength of surrounding electric fields.

Classical computing already exploits physics at the level of electron movements. But there are other ways of exploiting nature.

Analogue computing itself exploits the properties of electrical circuits as analogues of differential equations.

DNA computing [4] encodes problems and solution as sequences of bases (strands) and seeks to exploit mechanisms such as strand splitting, recombination and reproduction to perform calculations of interest. This can result in vast parallelism, of the order of 10^{20} strands.

Quantum computing [70] presents one of the most exciting developments for computer science in recent times, breaking out of the classical Turing paradigm. As its name suggests, it is based on quantum physics, and can perform computations that cannot be *effectively* implemented on a classical Turing machine.[1] It exploits interference, many worlds, entanglement and non-locality. Newer work still is further breaking out of the binary mind-set, with multiple-valued "qudits", and continuous variables. Research in quantum computing is mushrooming, and it is apparent that we are not yet in position to fully exploit the possibilities it offers. If only small quantum computers were to prove practical then uses could still be found for simulating various quantum phenomena. However, if larger computers prove possible we will find ourselves unprepared.

- Why are there so few distinct quantum algorithms? How can new ones be found?
- How do we discover new a quantum algorithms to solve a given problem? How do we use existing algorithms to solve new problems? How can we find the best algorithms to use given limited computational resources? More generally….
  - What would a discipline of quantum software engineering look like? (See later for more detail.)

---

1 Analogue (as in continuous) computing also breaks the Turing paradigm. But the real world is neither analogue nor classically discrete; it is quantum. So analogue computing might be dismissed as of theoretical interest only. However, the same dismissal might then be made of classically discrete (classical) computation! (The real world is also relativistic, but that paradigm has not been embraced by computation theory, yet.)

- How can quantum computers be harnessed most effectively as part of a hybrid computational approach?

Real World as Inspiration

Many important techniques in computer science have resulted from observing the real world. *Meta-heuristic search* techniques have drawn inspiration from physics (simulated annealing), evolution (genetic algorithms [35] [67], genetic programming [7] [52]), neurology (artificial neural networks [11] [51] [66] [82]), immunology (artificial immune systems [24]), plant growth (L-systems [80]), social networks (ant colony optimisation [12]), and other domains.

These have all proved remarkably successful, or look highly promising, yet the science underpinning their use comes nowhere near matching the science of classical computing. Given a raft of nature-inspired techniques we would like to get from problem to solution efficiently and effectively, and we would like to reason about the performance of the resulting systems. But this falls outside the classical refinement paradigm.

- What would a science of non-classical refinement look like? A science would allow us, for example, to reason confidently about the behaviour of neural networks in critical applications, to derive highly effective systems targeted at highly limited resources.

In the virtual worlds inside the computer, we are no longer constrained by the laws of nature. Our simulations can go beyond the precise way the real world works. For example, we can introduce novel evolutionary operators to our genetic algorithms, novel kinds of neurons to our neural nets, and even, as we come to understand the embracing concepts, novel kinds of complex adaptive systems themselves. The real world is our inspiration, not a restriction.

- How can we use nature inspired computation to build "better than reality" systems? What are the computational limits to what we can simulate?
- What is the best you can do given many components, each with highly restricted memory and processing ability?

Massive parallelism : breaking the von Neumann paradigm

Parallel processing (Cellular Automata [93], etc) and other non-classical architectures break out of the sequential, von Neumann, paradigm. (The fact that the sequential paradigm is named after von Neumann should not be taken to imply that von Neumann himself was an advocate of purely sequential computation; indeed, he was also one of the early pioneers of CAs [69].)

Under the classical paradigm assumptions, any parallel computation can be serialised, yet parallelism has its advantages.

Real-time response to the environment. The environment evolves at its own speed, and a single processor might not be able to keep pace. (Possibly the ultimate example of this will be the use of vast numbers of nanotechnological assemblers (*nanites*) to build macroscopic artefacts. A single nanite would take too long, by very many orders of magnitude.)

Better mapping of the computation to the problem structure. The real world is intrinsically parallel, and serialisation of its interactions to map the computational structure can be hard. Parallelism also permits colocation of each processor and the part of the environment with which it interacts most. It then permits colocation of the software: software agents can roam around the distributed system looking for the data of interest, and meeting other agents in a context dependent manner.

And once the classical paradigm assumptions are challenged, we can see that serialisation is not necessarily equivalent.

Fault tolerance. Computation requires physical implementation, and that implementation might fail. A parallel implementation can be engineered to continue working even though some subset of its processors have failed. A sequential implementation has only the one processor.

Interference/interaction between devices. Computation requires physical implementation, and those implementations have extra-logical properties, such as power consumption, or electromagnetic emissions, which may be interpreted as computations in their own right (see later). These properties may interfere when the devices are running in parallel, leading to effects not present in a serialised implementation. (Possibly the ultimate example of this is the exponentially large state space provided by the superposed parallel qubits in a quantum computer.)

The use of massive parallelism introduces new problems. The main one is the requirement for *decentralised control*. It is just not possible to have a single centralised source exercising precise control over vast numbers of heterogeneous devices (this is merely a covert attempt to serialise the system). Part of this problem is tackled by the sister *Grand Challenges in Ubiquitous Systems*, and part is addressed in the later section on *open processes*.

In the eye of the beholder : breaking the output paradigm

The classical paradigm of program execution is that an abstract computation processes an input to produce an output. This input-output mapping is a logical property of the computation, and is all that is important: no intermediate states are of interest, the computation is independent of physical realisation, and different instances of the computation yield precisely the same results.

Computation, however, is in the eye of the beholder. Algorithms are implemented by physical devices; intermediate states exist, physical changes happen in the world, different devices are distinguishable. Any information that can be observed in this physical world may be used to enrich the perceived computation [19].

Logical Trajectory Observations

An executing algorithm follows a *trajectory* through the logical state space. (Caveat: this is a classical argument: intermediate *quantum* computational states may be in principle unobservable.) Typically, this trajectory is not observed (except possibly during debugging). This is shockingly wasteful: such logical information can be a computational resource in its own right. For example, during certain types of heuristic search the trajectory followed can give more information about a sought solution than the final "result" of the search itself.

- How can logical observations made during execution be used to give useful information?

Physical Trajectory Observations

An executing algorithm is accompanied by physical changes to the world: for example, it consumes trajectory-dependent power as it progresses, and can take trajectory-dependent time to complete. Such physical resource consumption can be observed and exploited as a computational resource, for example, to deduce features of the logical trajectory. (For example, some recent attacks on smart cards have observed the power consumption profile and data-dependent timing of internal operations to deduce secret key information [17].) Such physical observations provide a very powerful source of information, currently exploited mainly by attackers, but available for more general computational use.

- What physical observations are feasible, and correlated with logical trajectories?
- What new uses can be found for such physical observations?

Differential Observations

An executing algorithm is realised in a physical device. Physical devices have physical characteristics that can change depending on environmental conditions such as temperature, and that differ subtly across *logically* identical devices. (Indeed, much of the rationale for digitisation is the removal of these differences.) So one can make observations not merely of the output of a single execution, but of set of outputs from a family of executions, of multiple systems, of different but related systems. For example, if repeated executions of a search each get 90% of elements of a sought solution correct then repeated executions might be combined to give an overall solution.

- How can diversity of multiple computations be exploited?
- How should diversity be engineered? By repeated mutation of a source program? By embracing technologically diverse solution paradigms?

Higher-order Observations

These are observations not of the program execution itself, but of the execution of the program used to design (the program used to design…) the program.

Open processes : breaking the algorithmic paradigm

In the classical paradigm, the ultimate goal of a computation is reaching a fixed point: the final output, the "result" of the computation, after which we may switch off the computer. The majority of classical science is also based around the notion of *fixed-point equilibrium* and *ergodicity* (ergodicity

is the property that the system has well defined spatial and temporal averages, because any state of the system will recur with non-zero probability).

Modern theories of physics consider systems that lack repetition and stability: they are *far from equilibrium* and *non-ergodic*. Perhaps the most obvious non-ergodic, far from equilibrium system is that of life itself, characterised by perpetual evolution (change). Most human problems are also best described in such terms; since computation is ultimately in service of such problems, the implications of non-ergodic, far from equilibrium physics must be considered in relationship to computing's future.

Consider the most basic of chaotic systems: the logistic process, parameterised by *R*.

$$x_{t+1} = Rx_t(1 - x_t)$$

The behaviours of various logistic processes as a function of *R* are shown in Figure 1, where each point on the plot is a point on the attractor.

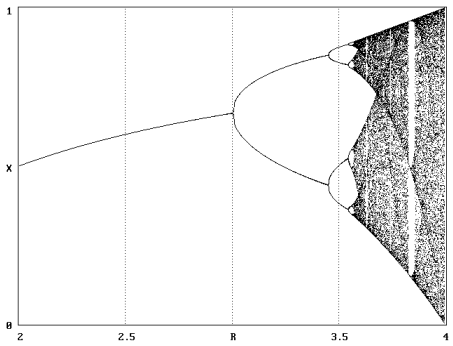

Figure 1: Points on the attractors of various logistic processes, versus the parameter *R*

For values of $1 < R < 3$, these logistic processes have a fixed point attractor. For $R = 3$ they an attractor of period two. As we raise *R*, the attractor becomes period four, period eight, etc. This *period doubling* continues as we raise *R*, and the values of *R* where each doubling occurs get closer together. For $R > 3.569945671\ldots$ the logistic process's attractor goes through an infinite number of values (except for a few "islands" or order, of attractors with multiples of odd periods). There is a *phase transition* from order (the region of period doubling) to chaos ("random" behaviour). The phase transition point at $R = 3.569945671\ldots$ is the so-called *edge of chaos* [60].

Consider a discretised process whose underlying (continuous) dynamics are those of the logistic equation. Imagine taking measurements from this process. Take very coarse measurements: say the process outputs 1 if $x > 0.5$, and 0 otherwise; and take samples of length *L* bits. For a given *L*, construct an automaton that represents the process. So now the logistic processes generated by various values of *R* are being interpreted as a variety of automata: *logistic machines*. It turns out that there is a clear phase transition (a peak in the machine size versus the entropy of the bit sequence) as we move from the period doubling region to the chaotic region.

At the phase transition, the machine size versus the length of the sequence *L*, *expands without bound*. That is, at the edge of chaos, the logistic machine requires an infinite memory machine for accurate representation. There is a leap in the level of intrinsic computation going on in the logistic machine at the edge of chaos. (In terms of the Chomsky hierarchy, the machine has gone from the level of regular grammars to the level of context-free grammars.)

At the edge of chaos, we can add new resources (computational or physical) to get results that are neither redundant (as they are in the structured period doubling regime) nor random (as in the chaotic regime). Within the classical paradigm, such conditions would be anathema, indicating unceasing variety that never yields "the solution". But in life-like systems, there is simultaneously sustained order, and useful innovation. In this setting, emergence of the unforeseen is a desirable property, rather disruptive noise.

Some computational approaches attempt to exploit the biological paradigm: cellular automata, evolutionary computation, recurrent networks (autocatalytic, neural, genomic, immune system, ecological webs, …), social insect and agent-based systems, DNA-computing, and nanite-systems that build themselves. However, in most of these cases, the implementations of such systems have been locked into themselves, *closed*, unable to take on new matter or information, thus unable to truly exploit emergence.

We should consider *open systems*, systems where new resources, and new kinds of resources can be added at any time, either by external agency, or by the actions of the system itself. These new resources can provide gateway events, that fundamentally alter the character of the system dynamics, by opening up new kinds of regions of phase space, and so allowing new possibilities. Computational systems are beginning to open themselves, to unceasing flows of information (if not so much to new matter). The openness arises, for example, through human interactivity as a continuing dialogue between user and machine [89], through unbounded networks, through robotic systems with energy autonomy. As computers become ubiquitous, the importance of *open systems physics* to understanding computation becomes critical. The solutions we expect from people are

ongoing processes, and this should be our expectation from computers too.

A coherent revolutionary Challenge, that also respects the past

Classical physics did not disappear when modern physics came along: rather its restrictions and domains of applicability were made explicit.

Similarly, the various forms of non-classical computation will not supersede classical computation: they will augment and enrich it. And when a wide range of tools is available, we can pick the best one, or the best combination, for each job. For example, it might be that using a quantum algorithm to reduce a search space, and then a meta-heuristic search to explore that, is more effective than using either algorithm alone.

We would like

to create a general flexible conceptual framework that allows effective and efficient exploitation of hybrid approaches, including classical and non-classical components

The journey is the important thing. At various points in journey-space researches will alight to mark their way, leaving behind diary entries to which they may return at a later date. In common parlance these intermediate recordings may be regarded as "achievements". Opportunities are manifold. We expect journeys relevant to the sub-disciplines to be articulated separately; several have already been prepared. These are given in the appendixes. Also relevant are the sister Ubiquitous Systems challenges.

It is important these separate journeys are not seen as independent explorations. Rather, their results and insights should provide valuable groundwork for the overarching challenge

to produce a fully mature science of all forms of computation, that unifies the classical and non-classical paradigms

The Grand Challenge Criteria

It arises from scientific curiosity about the foundation, the nature or the limits of a scientific discipline. It arises from questioning the assumptions of the classical paradigms, and aims at the creation of a new science.

It gives scope for engineering ambition to build something that has never been seen before. It aims to build a new science; the engineering opportunities will follow.

It will be obvious how far and when the challenge has been met (or not). It will never be met fully: it is an open journey, not a closed goal. The science will continue to mature, until itself overtaken by the next paradigm shift.

It has enthusiastic support from (almost) the entire research community, even those who do not participate and do not benefit from it. No. However, in the best tradition of paradigm shifts, the change will occur.

An important scientific innovation rarely makes its way by gradually winning over and converting its opponents: it rarely happens that Saul becomes Paul. What does happen is that the opponents gradually die out, and that the growing generation is familiarised with the ideas from the beginning.
– Max Planck, *Scientific Autobiography*, 1949

It has international scope: participation would increase the research profile of a nation. This is a new fundamental area of computer science.

It is generally comprehensible, and captures the imagination of the general public, as well as the esteem of scientists in other disciplines. Much popular literature already exists in several of these areas, written by scientists in other disciplines (quantum computing, complexity, nanotech, …), and so they and the general public are arguably already ahead of the CS community!

It was formulated long ago, and still stands. Its seeds have been around for a long time, but it has only recently become of obvious importance.

It promises to go beyond what is initially possible, and requires development of understanding, techniques and tools unknown at the start of the project. The structure of the Challenge mirrors the journey suggested by this criterion.

It calls for planned co-operation among identified research teams and communities. It is a multi-disciplinary Challenge, with contributions needed from a range of research specialities.

It encourages and benefits from competition among individuals and teams, with clear criteria on who is

winning, or who has won. There need not be a single "winner". Diversity of solutions should be encouraged to be applicable to a range of application domains. Winners may emerge in particular application domains, as the strengths of the various techniques become clear.

It decomposes into identified intermediate research goals, whose achievement brings scientific or economic benefit, even if the project as a whole fails. There are several components to the Challenge that can be explored in parallel.

It will lead to radical paradigm shift, breaking free from the dead hand of legacy. Non-classical computing *is* a radical paradigm shift!

It is not likely to be met simply from commercially motivated evolutionary advance. Applications might be supported by industry, but it is unlikely that the development of the underlying science would be.

References and Further Reading

[1] Editorial article. *Nature Immunology* **3**(10) 883, October 2002

[2] Andrew Adamatzky. *Computing in Nonlinear Media and Automata Collectives*. IoP, 2001

[3] Andrew Adamatzky, ed. *Collision-Based Computing*. Springer, 2002.

[4] Leonard M. Adleman. Molecular Computation of Solutions to Combinatorial Problems. *Science*, **266**:1021-1024, Nov 1994.

[5] Thomas Back, David B. Fogel, Zbigniew Michalewicz, eds. *Evolutionary Computation 1: basic algorithms and operators*. IoP, 2000

[6] Per Bak. *How Nature Works: the science of self-organized criticality*. OUP, 1997

[7] Wolfgang Banzhaf, Peter Nordin, Robert E. Keller, Frank D. Francone. *Genetic Programming, An Introduction: on the automatic evolution of computer programs and its applications*. Morgan Kaufmann, 1998

[8] Albert-Laszlo Barabasi. *Linked: the new science of networks*. Perseus, 2002

[9] G. Berry, G. Boudol. The chemical abstract machine. *Theoretical Computer Science* **96** 217-248, 1992

[10] Hugues Bersini, Francisco J. Varela. Hints for Adaptive Problem Solving Gleaned from Immune Networks. In H. P. Schwefel, H. Mühlenbein, eds, *Parallel Problem Solving from Nature*. Springer, 1991

[11] Christopher M. Bishop. *Neural Networks for Pattern Recognition*. OUP, 1995

[12] Eric W. Bonabeau, Marco Dorigo, Guy Theraulaz. *Swarm Intelligence: from natural to artificial systems*. OUP, 1999

[13] Daryl W. Bradley, Andy M. Tyrrell. Hardware Fault Tolerance: an immunological approach. In *Proc IEEE Conf on System, Man, and Cybernetics*. 2000

[14] S. D. Brookes, C. A. R. Hoare, A. W. Roscoe. A theory of communicating sequential processes. *Journal of the ACM* **31** 560-699, 1984

[15] Cristian S. Calude, Gheorghe Paun. *Computing with Cells and Atoms.* Taylor & Francis, 2001

[16] L. Cardelli, A. Gordon. Mobile ambients. *Theoretical Computer Science* **240** 177-213, 2000

[17] Suresh Chari, Charanjit S. Jutla, Josyula R. Rao, Pankaj Rohatgi. *Power analysis: attacks and countermeasures*. In Annabelle McIver, Carroll Morgan, eds. *Programming Methodology*. Springer, 2003.

[18] Bastine Chopard, Michel Droz. *Cellular Automata Modeling of Physical Systems*. CUP, 1998

[19] John A. Clark, Susan Stepney, Howard Chivers. *Breaking the model: finalisation and a taxonomy of security attacks*. Technical Report YCS-2004-371, University of York. 2004

[20] E. M. Clarke, E. A. Emerson, A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM ToPLaS* **8**(2) 244-263, 1986

[21] R. Cleaveland, J. Parrow, B. Steffen. The Concurrency Workbench: A Semantics Based Tool for the Verification of Concurrent Systems. *ACM ToPLaS* **15** 36-72, 1993

[22] David W. Corne, Marco Dorigo, Fred Glover, eds. *New Ideas in Optimization*. McGraw Hill, 1999.

[23] Dipankar Dasgupta, ed. *Artificial Immune Systems and Their Applications*. Springer, 1999.

[24] Leandro N. de Castro, Jonathan Timmis. *Artificial Immune Systems: a new computational intelligence approach*. Springer, 2002.

[25] Leandro N. de Castro, Fernando J. von Zuben. An Evolutionary Immune Network for Data Clustering. *SBRN '00, Brazil*, 84-89. IEEE, 2000.

[26] Marianne Delorme, Jacques Mazoyer, eds. *Cellular Automata: a parallel model*. Kluwer, 1999

[27] K. Eric. Drexler. *Engines of Creation: the coming era of nanotechnology*. Doubleday, 1986

[28] K. Eric Drexler. *Nanosystems: molecular machinery, manufacturing and computation*. Wiley, 1992

[29] J. Doyne Farmer, Norman H. Packard, Alan S. Perelson. The Immune System, Adaptation, and Machine Learning. *Physica D* **22** 187-204, 1986.

[30] Richard P. Feynman. Simulating Physics with Computers. *Int. J. Theor. Phys*. **21**(6/7). 1982.

[31] R. W. Floyd. Assigning meanings to programs. *Mathematical Aspects of Computer Science*, *Proc. Symp. in Applied Mathematics 19*, 19-32, AMS, 1967

[32] Stephanie Forrest, ed. *Emergent Computation: self-organizing, collective, and cooperative phenomena in natural and computing networks*. MIT Press, 1991.

[33] Stephanie Forrest, Alan S. Perelson, Lawrence Allen, Rajesh Cherukuri. Self-Nonself Discrimination in a Computer. *Symposium on Research in Security and Privacy*, 202-212. IEEE, 1994

[34] Murray Gell-Mann. *The Quark and the Jaguar*. Abacus, 1994.

[35] David E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[36] M. J. C. Gordon. HOL: A proof generating system for higher-order logic. *VLSI Specification, Verification and Synthesis*, Kluwer 1987

[37] Prabhat Hajela, Jun Sun Yoo. Immune Network Modelling in Design Optimization. In [22]

[38] Emma Hart, Peter Ross. The Evolution and Analysis of a Potential Antibody Library for Use in Job Shop Scheduling. In [22]

[39] C. A. R. Hoare. An axiomatic basis for computer programming. *CACM* **14**(1) 39-45, 1971

[40] C. A. R. Hoare. *Communicating Sequential Processes*. Prentice Hall, 1985

[41] John H. Holland. *Hidden Order: how adaptation builds complexity*. Addison-Wesley, 1995

[42] John H. Holland. *Emergence: from chaos to order*. OUP, 1998

[43] Yoshitero Ishida. Distributed and Autonomous Sensing Based on Immune Network. In *Proc Artificial Life and Robotics, Beppu*. AAAI Press. 1996

[44] Henrik Jeldtoft Jensen. *Self-Organized Criticality: emergent complex behaviour in physical and biological systems*. CUP, 1998

[45] Niels K. Jerne. Towards a Network Theory of the Immune System. *Annals of Immunology* **125C** 373-389, 1974

[46] Richard Jozsa: Characterising Classes of Functions Computable by Quantum Parallelism. *Proc. R. Soc. Lond.* A 435. 1991

[47] Stuart A. Kauffman. *The Origins of Order: self-organization and selection in evolution*. OUP, 1993

[48] J. A. Scott Kelso. *Dynamic Patterns: the self-organization of brain and behavior*. MIT Press, 1995

[49] John F. Kennedy. Announcement to the US Congress. 25 May, 1961

[50] Jeffrey O. Kephart. A Biologically Inspired Immune System for Computers. In Rodney A. Brooks, Pattie Maes, eds, *Artificial Life IV*. MIT Press , 1994

[51] Teuvo Kohonen. *Self-Organization and Associative Memory*. Springer, 1988

[52] John R. Koza. *Genetic Programming: on the programming of computers by means of natural selection*. MIT Press, 1992

[53] John R. Koza. *Genetic Programming II: automatic discovery of reusable programs*. MIT Press, 1994

[54] John R. Koza, Forrest H. Bennett III, David Andre, Martin A. Keane. *Genetic Programming III: Darwinian invention and problem solving*. Morgan Kaufmann, 1999

[55] D. Kozen. Results on the propositional mu-calculus. *Theoretical Computer Science* **27** 333-354, 1983

[56] George Lakoff. *Women, Fire, and Dangerous Things*. University of Chicago Press, 1986

[57] George Lakoff, Mark Johnson. *Metaphors We Live By*. University of Chicago Press, 1980

[58] Leslie Lamport. The Temporal Logic of Actions. *ACM ToPLaS* **16**(3) 872-923, 1994

[59] L. F. Landweber, E. Winfree, eds. *Evolution as Computation*. Springer, 2002

[60] Christopher G. Langton. Computation at the Edge of Chaos: phase transitions and emergent computation. In [32]

[61] Christopher G. Langton, ed. *Artificial Life: an Overview*. MIT Press, 1995

[62] Benoit B. Mandelbrot. *The Fractal Geometry of Nature*. Freeman, 1977

[63] Robin Milner. *A calculus of communicating systems*. LNCS 92, Springer, 1980

[64] Robin Milner. *Communicating and Mobile Systems: the π-Calculus*. CUP, 1999

[65] Robin Milner, J. Parrow, D. Walker. A calculus of mobile processes. *Information and Computation* **100**(1) 1-77, 1992

[66] Marvin L. Minsky, Seymour A. Papert. *Perceptrons*. MIT Press, 1988

[67] Melanie Mitchell. *An Introduction to Genetic Algorithms*. MIT Press, 1996

[68] Mark Neal, Jonathan Timmis. Timidity: A Useful Emotional Mechanism for Robot Control? *Informatica: Special Issue on Perception and Emotion based Reasoning*, 2003

[69] John von Neumann. *Theory of Self-Reproducing Automata*, edited by A. W. Burks. University of Illinois Press, 1966

[70] Michael A. Nielsen, Isaac L. Chuang. *Quantum Computation and Quantum Information*. CUP, 2000

[71] Mihaela Oprea, Stephanie Forrest. Simulated evolution of antibody gene libraries under pathogen selection. In *Systems, Man and Cybernetics*. IEEE, 1998

[72] Derek Partridge. On the difficulty of really considering a radical novelty. *Minds and Machines* **5**(3) 391-410, 1995

[73] Derek Partridge. Non-Programmed Computation. *CACM* **43** 293-301, 2000.

[74] Derek Partridge, T. C. Bailey, R. M. Everson, A. Hernandes, W. J. Krzanowski, J. E. Fieldsend, V.Schetinin. A Bayesian computer. 2004. http://www.cs.york.ac.uk/nature/gc7/partridge.pdf

[75] Gheorghe Paun. *Membrane Computing: an introduction*. Springer, 2002.

[76] Heinz-Otto Peitgen, Peter H. Richter. *The Beauty of Fractals: images of complex dynamical systems*. Springer, 1986

[77] C. A. Petri. *Kommunikation mit automaten*. PhD Thesis, Technical Report 2, Institut fur Instrumentelle Mathemematik, Bonn, 1962

[78] A. Pnueli. The temporal logic of programs. *Proceedings of FOCS*, 46-77. IEEE, 1977

[79] V. R. Pratt. Semantical considerations on Floyd-Hoare logic. *Proc. 17th Symp. Foundations of Computer Science*, 109-121. IEEE, 1976

[80] Przemyslaw Prusinkiewicz, Aristid Lindenmayer. *The Algorithmic Beauty of Plants*. Springer, 1990

[81] J. C. Reynolds. Towards a theory of type structure. *Proc. Paris Symposium on Programming*, LNCS 16, 408-425, Springer, 1974

[82] David E. Rumelhart, James L. McClelland. *Parallel Distributed Processing*. MIT Press, 1986
[83] D. S. Scott, C. Strachey. Towards a mathematical semantics for computer languages. *Proc. Symposia on Computers and Automata, Microwave Research institute Symposia 21*, 19-46, 1971
[84] Tanya Sienko, Andrew Adamatzky, Nicholas G. Rambidi, Michael Conrad, eds. *Molecular Computing*. MIT Press, 2003
[85] Derek J. Smith, Stephanie Forrest, David H. Ackley, Alan S. Perelson. Modeling the effects of prior infection on vaccine efficacy. In [23]
[86] Susan Stepney. Critical Critical Systems. In *Formal Aspects of Security, FASeC'02*. LNCS 2629. Springer, 2003
[87] Tommaso Toffoli, Norman H. Margolus. *Cellular Automata Machines*. MIT Press, 1985
[88] Duncan J. Watts. *Small Worlds: the dynamics of networks between order and randomness*. Princeton University Press, 1999
[89] Peter Wegner. Why interaction is more powerful than algorithms. *CACM* **40**(5) 1997
[90] P. H. Welch *et al*. Concurrency Research Group. www.cs.kent.ac.uk/research/groups/crg/ 2004
[91] L. Wittgenstein. *Tractatus Logico-Philosophicus*. 1921
[92] L. Wittgenstein. *Philosophical Investigations*. Blackwells, 1953
[93] Stephen Wolfram. *Cellular Automata and Complexity: collected papers*. Addison-Wesley, 1994
[94] Andrew Wuensche, Mike Lesser. *The Global Dynamics of Cellular Automata*. Addison-Wesley, 1992

Initial Journeys and Waypoints

The appendixes that follow comprise a collection of suggested journeys that could be brought under the umbrella of Non-Classical Computation. It is assumed that these journeys would be conducted not in isolation, but in the context of the overall challenge, informing it, and being informed by it. The currently identified journeys are:

- Non-Classical Philosophy – Socially Sensitive Computing
- Non-Classical Physics – Quantum Software Engineering
- Non-Classical Refinement – Approximate Computation
- Computing in non-linear media – reaction-diffusion and excitable processors
- Artificial Immune Systems
- Non-Classical Interactivity – Open Dynamical Networks
- Non-Classical Architectures – Evolving Hardware
- Non-Classical Architectures – Molecular Nano-technology
- Non-von Architectures – Through the Concurrency Gateway

Journey: Non-Classical Philosophy – Socially Sensitive Computing

Wittgenstein produced two major works on the philosophy of language: the 1921 *Tractatus* [91], and the 1953 *Philosophical Investigations* [92]. We can use the *Tractatus* and its relationship to the world, as a model of classical computation. However, Wittgenstein found flaws in his initial work, and he explored these in his later *Philosophical Investigations*. Can we use these later ideas as a model of post-classical computation?

A Philosophical Paradigm and Computing

Wittgenstein's *Tractatus* encapsulates a formal and logical representational of language into a descriptive form based upon denotational (or referential) semantics. Given the Church-Turing Thesis we can take the *Tractatus* as a paradigmatic description of classical computer science.

A major result of the *Tractatus* stance is that every object is potentially unambiguously describable. Let us define a ***'rational' set*** to be *a set where there is a finite set of rules that can include unambiguously any member of that set and unambiguously excludes any non-member of that set*. All the sets referenced by the Tractatus are rational and context independent, or have an explicit context that is also rational. The *Tractatus* provides an extensive model of computer languages.

There are social consequences of the view adopted by the *Tractatus* in that it is assumed that rules can be created for all situations and as such these rule can bypass human judgement. It also assumes that

there is only one correct way of seeing the world and so human existence can be governed by some finite set of laws.

Dual Semantics

Computer languages have a *dual semantics*. The names given to data items, procedures and sub-routines at the highest level have referents in the world. The analysis of the problem domain identifies constructs in the world that are meant to be stable and unchanging (as per *Tractatus* referents) to which names can be given and meaning assigned. Yet the ultimate referent is the bit, the mechanical equivalent of Wittgenstein's referent objects. At the bit level the program links to the world and has meaning, which allows the program to have "sense" with respect to the computer.

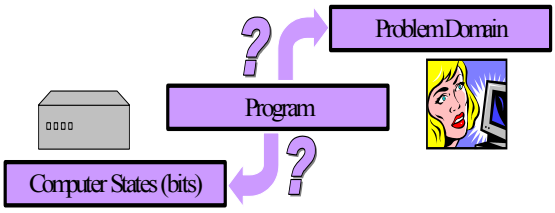

Figure 2. The dual semantics

But according to the *Tractatus,* a proposition can have *one and only one complete analysis*. Such an analysis is dependent upon only the essential features of the proposition (here, program) that link it to the referent objects (here, the bit). So the alternative high-level interpretation of a program depends upon its *accidental features*. This develops a peculiar tension in program design that is hard to keep stable, particularly with respect to the informal, and often undefined, mechanism which links the program names with the user's domain. Further, the 'objects' that are usually chosen to be referenced in the informal analysis of the problem domain do not normally have all the features required of Wittgenstein's objects.

The Paradigm Leap

The *Tractatus* is a magnificent piece of work and is an effective description of how programming languages should be linked to a computer through 'sense' (as with meaning) assignment. There is no problem with the engineering necessity of this approach to sense and meaning. On a broader scale it sidesteps many of the paradoxes of the linguistic philosophy of the day. However, it has *one fatal flaw* when applied to the human use of language and Wittgenstein eventually exposed this flaw. He noted that *it is not possible to unambiguously describe everything within the propositional paradigm*. He found that the normal use of language is riddled with example concepts that cannot be bounded by logical statements that depend upon a pure notion of referential objects. One of his illustrations is an attempt to define a "game". Such a definition cannot be achieved that either excludes all examples that are not games or include all examples that are. Most things are not potentially unambiguously describable. This lack of boundaries for concepts is the *family resemblance* effect. It is through such considerations that Wittgenstein proposed his new linguistic philosophy.

We call the basis of this new philosophy *inferential semantics*. Let us define an ***'irrational' set*** to be *a set where no finite set of rules can be constructed that can include unambiguously any member of that set and, at the same time, unambiguously exclude any non-member of that set.*[2]

Even though there are irrational sets we still have rational sets, and so denotation remains one mechanism for relating meaning to a name. For irrational sets there is an additional and more important mechanism for meaning assignment based on human usage and context. It is this latter mechanism that provides the link between the program and the world it is designed to represent and is the other half of the dual semantics.

Some Predictions

So we have computer programs with a semantics based upon computer bits, but we create programs that cannot rationally be assigned meaning to the very problem domain for which they have been written. Programs must remain in the domain of rational sets if they are to be implemented on a machine. However, we do have the freedom to use the program's accidental properties without affecting the program's meaning with respect to the computer. We can chose the names we use and select the computer organisation from the possibilities bounded by the essential program.

A proposition, and hence a program, can adopt many equivalent forms. It is the job of a compiler to make a transformation of a program in order that it is acceptable for a particular computer to run it. Apart from some limitations, the choice of form is in the hands of the programmer.

This means that:

- reverse engineering requires domain information
- formal 'objects' (e.g. operating systems) will be stable but informal 'objects' (e.g. persons, chairs,

[2] Note that we are not talking about such things as fuzzy sets, or probabilistic sets. These sets are *rational* in that a membership number is assigned by a finite set of rules.

games) will never be fully captured or be stable because they are irrational sets

- it will not be possible to completely represent certain human functionality such as natural language understanding on any machine that is not adaptable
- increasing a training set for machine-learning algorithms will eventually cause degradation in recognition performance if the set includes irrational distinctions

Inferential Semantics

The tension caused by the dual semantics that pivots on the essential and accidental meaning of the signs used in programs has been recognised as can be seen by the continued search for new languages, program structuring, and systems design methods (e.g. conceptual modelling, object orientation). The central problem of the human context has also been addressed through the pursuit of natural language understanding, naïve physics, case-based reasoning and adaptive interfaces. There is a belief that given sufficient power or moving beyond the Turing machine would somehow solve the problem. However, none of the approaches tried so far have really succeeded, not with many-fold increases in computer power, or parallel mechanisms such as neural nets. Many of the pursuits have been constrained by the formal bounds represented by the *Tractatus* and those approaches that have broken away have not bridged the gap identified here.

The Challenge

An alternative to Wittgenstein's family resemblance is Lakoff's [56][57] use of prototypes (paradigms) and metaphor instead of reference. With either route we have a more acceptable approach to human relationships in that there will always be a need for human judgement because what is acceptable behaviour or performance is a time sensitive and socially dependent notion. The requirement to encapsulate a wide range and ever changing perceptions of a problem domain will be the need for a continuous link with human activity. Such perceptions cannot be predicted and hence planed for in advance. So many of the current principles of design will have to be shelved and two distinct design paths will need to be forged that involve the two independent elements of a program; the formal rational and the informal irrational (figure 3).

The challenge is

to construct computing based upon family resemblance rather than sets, paradigms rather than concepts, and metaphor rather than deduction, and to devise systems that make judgement rather than take decisions

One possibility is that we might be able to write dynamic, socially sensitive interfacing-compilers that can match any program to any user (figure 3).

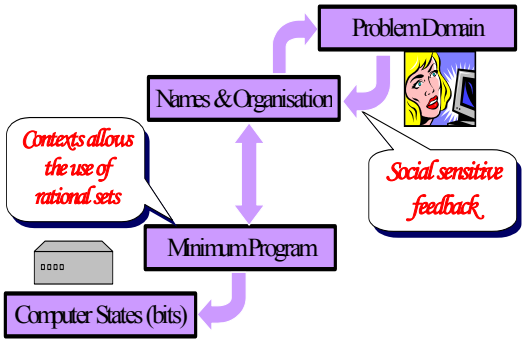

Figure 3. Showing where change can occur to solve the dual semantic problem

Such a compiler would be in 'conversation' with its user, other users and machines via (say) the Internet absorbing the human cultures and language so that its generated semantic and semiotic mappings make a program usable by a person. This might provide a more natural communication between people and machines; it may identify what is really meant by common sense.

The overall Challenge

GC7 provides a series of challenges in non-classical computing. It is hoped that such explorations will produce computational engines that rise beyond some of the limitations found in classical computation.

Many of these limitations are caused by the existence of irrational sets, or are created by the mismatch between the computer and its system with the problem domains.

The challenge can be expressed as

to develop a Science of Mechanisms

The science would develop a way of arranging mechanisms into family organisations, and in particular identify such mechanisms by their organisational features; features that are relevant to a counter family organisation of problem domains. A result would be a way of reducing complexity of implementation by construction mechanisms that match the problem. Flexibility to change (as required for irrational sets) would be provided by a change in mechanism definition. Mechanism definition would also include the soft variants in

terms of program organisation and the possibility of combining distinct physical implementations.

Journey: Non-Classical Physics – Quantum Software Engineering

This journey of Non-Classical Computation is

to develop a mature discipline of Quantum Software Engineering

We wish to be ready to exploit the full potential of commercial quantum computer hardware, once it arrives, projected to be around 2020 (or, less optimistically, "20 years from now", no matter when "now" is).

We might have to wait a while for commercial quantum computers, but when they arrive, Moore's law suggests they will grow in power *very quickly*. Doubling a classical computer's register length (roughly) doubles classical computing power, but adding just *one bit* to a quantum computer's register doubles quantum computing power. We need to be ready to exploit these devices once they appear. However, the majority of today's theory of computation, algorithms, programming languages, specification models, refinement calculi, and so on, is purely classical. The challenge is to build the corresponding languages, tools and techniques for quantum software engineering.

We need to raise the level of thinking about quantum programs. Today we reason about quantum programs predominantly at the level of quantum *gates*: imagine how far classical computing would have progressed if the only language we had to describe programs was that of AND and OR gates! Most importantly, we need a new paradigm (or paradigms) for thinking about quantum computations, to augment the existing classical declarative, functional, and imperative paradigms.

The whole of classical software engineering needs to be reworked and extended into the quantum domain.

Foundations

Much foundational work is still needed. We need further developments of the fundamentals of quantum computability. We need to investigate **quantum algorithmic complexity**: time, space, "parallel universe space", and any other parameters of interest.

We have models of classical computation – von Neumann machines with fetch-execute-store, imperative, functional and logic languages, etc – that let us write and reason about classical programs without worrying about logic levels, transistors, gates, etc. In much the same way we need **metaphors and models of quantum computation**, that enable us design and reason about quantum algorithms without recourse to QM, unitary matrices, etc. Does Deutsch's many-worlds description provide the best programming metaphor, or are there better ones? Whatever the actual metaphors chosen, they must be formalised into new computational models.

We need theories and models of that weirdest quantum process of all: that of **quantum entanglement**. Two qubit entanglement is relatively well understood – but multi qubit entanglement, and qudit entanglement, are barely understood.

Quantum Computational Models

There are many models of classical computation, such as Turing machines, functional combinators, logic circuits, fixed point approaches, and so on. Within the context of classical computation these are essentially equivalent, yielding identical results but from vastly differing underlying formalisms. Within the quantum computational world this unity is less clear. For example, a fixed-point algorithm on a quantum computer could include a superposition of all fixed points, not just the stable one obtained by repeated substitution.

This suggests that the various classical formalisms may generalise to the quantum realm in different ways. Currently, the most extensively studied quantum computational model is the circuit model. But designing high-level algorithms, reasoning about complexity, or other such important tasks, are very hard in this formalism.

Additionally, this model may not be the most appropriate quantum generalisation. The underlying structure of Quantum Information may be so radically different from anything that we currently understand that we need a whole new approach. Quantum mechanical versions of classical models may simply be insufficiently powerful to encompass the new properties offered by the quantum domain.

However, before we attempt to resolve such a daunting issue, there is much to be gained from examining the various classical models, to see if, and how, and how far, they might provide us with

new insights into computation within the quantum domain.

We need to thoroughly investigate the various classical computational models in terms of their generalisability to cover quantum properties. This will either provide powerful new generalised quantum computational models, or potentially demonstrate that a truly novel, fundamentally quantum, paradigm is indeed required.

Additionally, this work will feed back into classical computation (one example of this interrelationship between the necessity of quantum reversibility and the possibility of designing efficient classical "reversible compilers").

Languages and Compilers

We need to determine the **fundamental building blocks** of quantum programming: is there a simple extension of GCL? of classical logic languages? of classical assembly languages? is an entirely new paradigm needed?

We need to design suitable **assembly level** and **high level Q-languages** (analogues of classical imperative, declarative, and functional languages, at 3rd, 4th, 5th generation, and beyond). We need to design and build the corresponding **Q-compilers** for these languages.

We need to design and implement (initially, simulate) **new Q-algorithms** (beyond the current ones of Min Max, Shor's period finding algorithm used for factorization, and Grover's algorithm for DB searching). What classes of algorithms may be quantised? How may certain well-known classical algorithms be quantised?

We need to develop suitable **reasoning systems** and **refinement calculi** for these languages. (Even sequential composition is different in the quantum regime, due to the fundamental unobservability of the intermediate state.) Although higher level specifications may well abstract away from details of any underlying classical *versus* quantum implementation, there may be certain application-specific **quantum specification languages**, for example, for **quantum protocols**.

Methods and Tools

Before commercial quantum computers are available, we have to make do with simulations on classical machines. We need to implement powerful **quantum computer simulators**, in order to perform computational experiments and validate language and algorithm designs. (Computational resources for simulating quantum algorithms can be exponentially large. Something like a simulation engine of multiple FPGAs might be appropriate, to get the required massive parallelism.)

We need to discover what **higher level structuring techniques and architectures** are suitable for quantum software. In particular, can classical structuring (such as object-orientation, or component based software), be extended to incorporate Q-software? How can classical and quantum paradigms co-exist? (It seems likely that, at least to start with, most software will remain classical, with a "call-out" to quantum power as needed. But the development process needs to be able to handle such hybrid developments seamlessly.)

Given that quantum execution is *in principle* unobservable, we need to discover new **debugging and testing techniques** for these Q-languages.

We need to design ways of **visualising** Q-algorithm execution, as an aid to understanding, design, and implementation.

Novel Quantum possibilities

Quantum information processing can do some things that cannot even be simulated by discrete deterministic classical computers. We need to extend quantum software engineering to encompass these new domains.

Quantum devices can produce **genuine random numbers**; classical digital simulations can produce only *pseudo*-random numbers. We need to investigate the differences this causes, if any. In the short term, will a quantum computer simulator need to be hooked up to a genuinely random number source? In the longer term, what new power, what new difficulties, might emerge as a result of genuine randomness?

Quantum **entanglement** offers many new possibilities, such as information teleportation. We need to understand how entanglement can be applied to produce genuinely new algorithms, and new kinds of protocols.

Journey: Non-Classical Refinement – Approximate Computation

This journey of Non-Classical Computation is

to develop a science of approximate computation, and to derive from it a well-founded discipline for engineering approximate software

A radical departure from discrete correct/incorrect computation is required, a shift away from logics towards statistical foundations, such that meaningful estimates of 'confidence' emerge with each approximate result. This implies that probabilities play an integral part in computation throughout the process. The component probabilities and the eventual confidence estimates, if secured by large numbers (e.g. repeated sampling from a proposed distribution), imply a computational effort that is becoming increasingly feasible as a result of hardware advances as well as innovative developments in statistical modelling theory (e.g. reversible-jump Markov Chain Monte Carlo methods).

Classical computation versus approximations

The classical, discrete, view of computation has each step as either correct or incorrect, and the middle ground excluded. This naturally leads to formal logics as the dominant underpinning framework. The programmer devises the "formula", which is intended to be an exact solution to the problem; this symbol structure is translated into a machine executable form and the manipulations that the programmer envisaged are performed automatically, at high speed and with complete accuracy.

Consider the symbol structures being manipulated by a trained a multilayer perceptron (MLP), for example. These are not formulae composed of operators and variables that admit a ready mapping to the operations and parameters of the human conception of the problem. One consequence is that any adjustment to the function to be computed by an MLP involves complete retraining, because code-fixing is not an option. The "formulae" cannot reasonably be devised by a programmer; they must be automatically generated from data samples.

Typically, the output of an MLP classifier, a real-value, is arbitrarily thresholded to obtain a class label. This and other inherent weaknesses of an approximate classifier constructed with empirically determined (suboptimal) values for its parameters are widely acknowledged. *Ad hoc*-ery is rife in neural computing, but work on error-bars already points the way towards a well-founded science.

These innovative developments to move beyond the constraint of correct/incorrect results from hand-crafted formulae are but piecemeal strategies; they need to be woven into the basic fabric of a comprehensive model for approximate computation, not stitched-on to classical computation as useful extras, or mere curiosities.

How would the classical paradigm be shifted?

Taking the viewpoint that the computational task is an unknown (or intractable, see later) function, the computational goal is to approximate it in a way that holds the promise of reasonable optimality, but crucially associates a *meaningful estimate of confidence* with every output computed. In general terms, data-driven software development supplants specification-driven; computational tasks are viewed as data-defined rather than (abstract) specification-defined.

In detail, the approach might be through a survey, sampling by, say, Markov Chain Monte Carlo methods across a continuum of potentially viable models. By doing this within a Bayesian framework, rationalisable probabilities are attached to various elements throughout the computational process. The outcome is a weighted average across a range of modelling possibilities. It is a well-founded approximation whose validity emerges as a secure estimate from the computational processes employed. The infrastructure of the new paradigm seeks to avoid searching, comparing and selecting from amongst a discrete set of alternative models (and hence commitment to a specific model, or even discrete set of alternative models) by maintaining the range of potential models as a set of continuous parameters; probability theories, secured by large-number sampling, provide the over-arching framework.

A fundamental basis of continuity avoids the brittleness inherent in discrete, classical computation. Notice, for example, that the necessary discretisation of the real numbers that plagues classical computation is not similarly problematic for MLPs, despite their fundamental dependence upon the real continuum.

Initially at least, classical computation will provide the virtual machine upon which the approximate computations will run, but hardware innovations coupled with the establishment of generally applicable approximation algorithms could change that dramatically. However, building the required confidence in a classically programmed virtual machine is not the same scale of problem as doing it individually for every piece of application software.

The initial challenge is to begin to establish the limits and the potential infrastructure of such a science of approximate computation. This includes

major subdomains, such as a discipline of engineering approximate software. It also involves the identification and integration into a coherent framework of many activities that are currently pursued under a variety of labels, for example, statistical pattern recognition, some varieties of data mining, statistical data modelling, some technologies of inductive generalization or data-driven computation.

A science of approximate computation: when and where?

The new science of approximate computation will not oust the classical one; it will sit alongside it as a new weapon in an armoury of well-founded alternative computational techniques to be used when appropriate.

It will be appropriate to use whenever a computational task is defined more by samples of desired or observed behaviour than by an abstract specification. It will also be appropriate to use whenever the problem is well defined but computationally intractable, where the particular task is appropriate for approximate solutions, albeit with a 'confidence' measure attached; there is no prohibition on certainty emerging as an extreme of approximation.

Consider an illuminating extreme – safety-critical software. Such systems would seem to absolutely require the classical strategy: they must be correct. However, the practical impossibility of this requirement leads to a slight relaxation: it is typically couched in terms of a very low failure/error rate, and the major component of the required assurances is extensive testing. The bulwark of statistical reasoning, as an integral part of the testing, is thus dragged in by the back door (as it were) – how much better to integrate it into the fabric of the computation from beginning to end, instead of tagging in on the end as a stopgap for verification failure?

Will 'programming' an approximation computer be more difficult than conventional programming? All we can say is it will be fundamentally different [74] – for example, data analysis, selecting sampling strategies, rather than formula derivation. The 'programming' difficulties that confront the user of this new paradigm will be directly determined by how successful we are in formulating the fundamental model(s) of approximate computation.

Nothing in the above *requires* a new paradigm: any of the innovations envisaged could be realised within the scope of classical computation, as some already are. However, although a screwdriver can be used to open a tin, it is quicker, neater and generally preferable to use a well-designed tin opener for the task.

Journey: Computing in non-linear media – reaction-diffusion and excitable processors

Nature diffuses, reacts and excites. Does it compute? Everyday life gives us instances of propagating structures: dynamics of excitation in heart and neural tissue, calcium waves in cell cytoplasm, the spreading of genes in population dynamics, forest fires. Could we use the travelling structures — emerging in reaction-diffusion and excitable systems — to do computation? This journey of Non-Classical Computation is

to develop a science of computation using spatio-temporal dynamics and propagating phenomena, in many-dimensional amorphous non-linear media

What is so particular about reaction-diffusion systems? A non-linear chemical medium is *bistable*: each micro-volume of the medium has at least two steady stable states, and the micro-volume switches between these states. In the chemical medium, fronts of diffusing reactants propagate with constant velocity and wave-form; the reagents of the wave front convert reagents ahead of the front into products left behind. In an excitable chemical medium the wave propagation occurs because of coupling between diffusion and auto-catalytic reactions. Auto-catalytic species produced in one micro-volume of the medium diffuse to neighbouring micro-volumes, and thus trigger an auto-catalytic reaction there. So an excitable medium responds to perturbations that exceed the excitation threshold, by generating excitation waves.

Why are excitation waves so good for computing? Unlike mechanical waves, excitation waves do not conserve energy, rather, they conserve waveform and amplitude, do not interfere, and generally do not reflect. So excitation waves can play an essential role of information transmission in active non-linear media processors.

Specialised non-linear media processors

A problem's *spatial representation* is a key feature of reaction-diffusion processors. Data and results

are represented through concentration profiles of the reagents, or spatial configurations of activity patterns. A computation is also defined in a physical space. The computation is realised by spreading and interacting waves of the reagents, or excitation patterns. A computational code, or program, is interpreted in a list of possible reactions between the diffusing components, and in a form of diffusive or excitation coupling between micro-volumes of the computing medium. Usually, such properties could not be changed online. However they can be determined and adjusted to work towards the solution of a particular problem. Therefore most reaction-diffusion processors are intentionally designed to solve a few particular problems — they are specialised. Examples of working laboratory prototypes of specialised reaction-diffusion computing devices include Belousov-Zhabotinsky chemical medium image processors implemented (Kuhnert-Agladze-Krinsky and Rambidi designs), chemical processors for computation of a skeleton of planar shape, plane sub-division (Voronoi diagram), shortest path (Adamatzky-Tolmachev-De Lacy Costello designs), chemical processors for robot navigation and taxis (De Lacy Costello-Adamatzky implementations).

Experimental chemical computers are very rare species — there are just a handful of chemical processors in the world — why? Because the design of every chemical processor requires at least a chemist and a computer scientist. So one of the actual tasks deals with the make-up of non-classical scientists is

to dissolve boundaries between chemical-physical sciences and theoretical computer science, training a new generation of scientists who tackle theory and experiments with ease

Computational (logical) universality of non-linear media: dynamical vs. static architectures

A natural process is called *computationally universal*, by analogy with mathematical machines, if it potentially can calculate any computable logical function, so realises a functionally complete set of logical gates in its spatio-temporal dynamics. So in the state of the given system, we need to represent information quanta (usually TRUE and FALSE values of a Boolean variable), logic gates (where information quanta are processes), and routes of information transmission or connections between the gates. There are two ways to build a logical circuit in a non-linear system: structural, or *stationary*, and structureless, or *dynamic*, designs.

In a *stationary* design, which underlines an architecture-based universality, a logical circuit is embedded into a system in such a manner that all elements of the circuit are represented by the system's stationary states; the architecture is static and its topology is essential for a computation. The architecture-based universality allows for applying conventional solutions to unconventional materials: we could fabricate varieties of traditional computers made of non-standard materials (glass tubes filled with chemicals, molecular arrays, excitable tissues). Examples of stationary architectures of non-classical computers include Hjelmfelt-Weinberger-Ross mass-transfer or kinetic-based logical gates (a logical gate is constructed by adjusting flow rates of chemical species between several reactors), Tóth-Showalter circuits (a logical circuit comprises several narrow tubes filled with Belousov-Zhabotinsky excitable chemical medium, the tubes are connected via expansion chambers where logical functions are implemented by interacting wave fronts), and Adamatzky-de Lacy Costello palladium gate (implemented in simple non-excitable reaction-diffusion chemical system).

Most biological systems are "architecture-based computationally universal". Could they compute better if they lose this comparmentalisation? If all neurons in our brain were to dissolve their membranes and fuse into an amorphous mass, could this "gooware" be computationally more powerful?

Dynamic, or collision-based, computers employ mobile self-localisations, which travel in space and execute computation when they collide with each other. Truth-values of logical variables are represented by absence or presence of the travelling information quanta. There are no pre-determined wires: patterns can travel anywhere in the medium, a trajectory of a pattern motion is analogous to a momentarily wire. A typical interaction gate has two input 'wires' (trajectories of the colliding mobile localisations) and, typically, three output 'wires' (two representing the localisations' trajectories when they continue their motion undisturbed, the third giving the trajectory of a new localisation formed as the result of the collision of two incoming localisations). The travelling is analogue to information transfer, while collision is an act of computation, thus we call the set up *collision-based computing*. There are three sources of collision-based computing: Berlekamp-Conway-Guy proof of universality of Conway's Game of Life via collisions of glider streams, Fredkin-Toffoli conservative logic and cellular automaton implementation of the billiard ball model (Margolus block cellular automata), and the Steiglitz-Kamal-Watson particle machine (a concept of computation in cellular automata with soliton-like patterns). A wide range of physical,

chemical, and biological media are capable of collision-based computing. Thus, for example, this type of computation can be implemented as localised excitation in two- and three-dimensional excitable lattices, as breathers and solitons travelling along polymer chains (and DNA molecules), as excitons in mono-molecular arrays (like Scheibe aggregates), and even as quasi-particles in gas-discharge systems and two-component reaction-diffusion systems.

So far we can implement a gate or two in collision of propagating localisations — what about a collision-based chip?

Complexity and Computation

Non-linear sciences are a centrepiece of contemporary sciences, from physics to biology. The dynamics of non-linear media are becoming a crucial tool in understanding complex behaviour in natural and artificial systems, emergent behaviour, complexity, and self-organized criticality. Non-linear dynamics of large-scale massive systems, described in terms of physical systems or their continuous and discrete mathematical and computational models, are typically recruited at the present time to explain the nature of complexity, to predict the behaviour of biological and social systems, and to discover the novel properties of multi-component systems. To develop a solid framework of computing in non-linear media we must answer a number of questions residing at the edge between complexity and computation.

What families of computational tasks are solved in chemical and biological non-linear media?

How are the computing abilities of non-linear media related to the behavioural complexity of the media itself? Is it necessarily true that a system with a more complex behaviour can solve more computationally complex problems than a system with less complex behaviour?

What is a relation between complexity of space-time dynamics of a non-linear medium, and computational complexity of the medium as a computer?

How do we exert some control over the dynamics of non-linear media? What engineering approaches are required to form interfaces between conventional computers and experimental prototypes of non-linear media based computers?

How do we program non-linear medium computers? What is a trade-off between the medium's complexity and the medium's programmability? Does complexity reduce programmability?

Cellular automata: Non-linear medium mathematical machines

The field of cellular automata — massive-parallel locally-connected mathematical machines — flourishes and occupies a significant part of computational sciences. A cellular automaton is a lattice of uniform finite automata; the automata evolve in discrete time and take their states from a finite set. All automata of the lattice update their states simultaneously. Every automaton calculates its next state depending on the states of its closest neighbours.

Cellular automata models of reaction-diffusion and excitable media capture essential aspects of natural media in a computationally tractable form, and thus could be adopted as a tool for automatic design of non-linear media computers, development of reaction-diffusion algorithms, and pre-experiment verifications.

Discovering Computation

How do we find reaction-diffusion or excitable media to fulfil our computational goals in real wet-ware? There is not much choice at the moment. There are dozens of oscillating chemical reactions, yet most look quite similar, and so almost everybody experiments mainly with Belousov-Zhabotinsky media. The advantage of such ubiquity is the chance to verify each other's experiments. At the molecular level the situation is not as good: we can fabricate molecular arrays, but there are almost no reports on any feasible computing experiments, either with classical waves, or with mobile-self localisations. Which problems can be solved in what types of non-linear media? Should we fabricate these media from scratch or could we instead search for already existing species in nature?

What are the principal characteristics of spatially-extended non-linear media (chemical, physical or biological) that enable them to implement useful computation?

Journey: Artificial Immune Systems

The inspiration and the analogy

There is a growing interest in the use of the biological immune system as a source of inspiration to the development of computational systems [24]. The natural immune system protects our bodies from infection with a complex interaction of white blood cells, called B Cells and T Cells. Upon encountering an antigen (an infecting item), B Cells are stimulated by interacting with the antigen, and, with the help of T Cells, undergo rapid cloning mutation. This is an attempt by the immune system to kill off the invading antigen and prepare the immune system for another infection from that antigen (or similar antigen). The immune system maintains a *memory* of the infection, so that if ever exposed to the same antigen again, a quicker response can be elicited against the infection.

There are many facets of the immune system that can be considered useful for computation, including pattern recognition, feature extraction, learning, noise tolerance, memory, and inherent distributed parallel processing. For these and other reasons, the immune system has received a significant amount of interest as a metaphor within computing. This emerging field of research is known as Artificial Immune Systems (AIS).

Essentially, AIS is concerned with the use of immune system components and processes as inspiration to construct computational systems. AIS is very much an emerging area of biologically inspired computation. This insight into the natural immune system has led to an increasing body of work in a wide variety of domains. Much of this work emerged from early work in theoretical immunology [45] [29] [10], where mathematical models of immune system process were developed in an attempt to better understand the function of the immune system. This acted as a mini-catalyst for computer scientists, with some of the early AIS work being on fault diagnosis [43], computer security [33], and virus detection [50]. Researchers realised that, although the computer security metaphor was a natural first choice for AIS, there are many other potential application areas that could be explored, such as machine learning [25], scheduling [38], immunised fault tolerance [13], and optimisation [37]. In addition, AIS has been offering better understanding of the immune system [85] [71], whose mechanisms are hugely complex and poorly understood, even by immunologists. The field of AIS is both a powerful computing paradigm and a prominent apparatus for improving understanding of complex biological systems.

Questions can also be asked such as: How do we construct truly autonomous evolving systems that are capable of adapting to an ever-changing environment? How do we construct systems that can implement complex control mechanisms that are beyond the capabilities of current approaches? How do we cope with the massive increase in complexity of systems that are being given to the information technology society as a whole?

AIS algorithms have the possibility of breaking the *algorithmic paradigm* in two ways. First, they capture the immune system's mechanism of exploiting randomness, therefore removing the idea that "randomness is bad". Secondly, the immune system is inherently a continually learning system with no end point, with no "final output". Clearly, current solutions to such problems have made some progress. However, with the increases in scale and complexity come new and ill-understood demands on computational systems. This has resulted in many systems being inflexible, *ad hoc*, difficult to configure, and impenetrably arcane to maintain. Therefore, alternative ways to construct a new generation of more autonomous and self-organising computational systems are being sought.

The models

There are two main models for AIS: the population based models (or selection models), and the network model (see [24] for details), which have impacts on different areas of the main Grand Challenge.

The Selection model

The immune selection model is computationally inspired by the processes during early maturation of immune cells, before they are released into the lymphatic system. It uses some particular algorithm (positive, negative, clonal, …) to select a set of *recognisers* (supervised learning) or *classifiers* (unsupervised learning), of self or non-self (details depending on the precise algorithm).

This model fits well with the other bio-inspired soft learning systems, such as neural nets and genetic algorithms. The major contributions to the Grand Challenge are in the area of *breaking the refinement paradigm*.

In all these soft learning approaches, there is a discontinuity between the problem statement and the bio-inspired solution. With both NNs and AISs, the solution is distributed over the entire system. Each artificial antibody may recognise several different antigens: the specific response to a particular antigen is a global property of all the antibodies. The complex response emerges from the simpler behaviour of individual parts.

The way point questions specific to AIS include:

- What are the effects on the selection algorithm of parameter selection, with regards to the outcome and applicability of these algorithms?
- Can we observe the computational trajectory taken during selection and recognition to get useful information?

The immune system selection model forms an excellent exemplar for breaking the refinement paradigm. The challenge is to develop a *science of non-classical refinement*, that permits quantitative reasoning about *all* bio-inspired algorithms, including AISs, in both a bottom up and top down manner:

- *understanding* and *predicting* the global recognisers and classifiers that emerge from a collection of local non-specific agents
- a means to *design* and *implement* appropriate sets of recognisers or classifiers for particular applications, in a rigorous (but possibly non-incremental) way
- *quantitative description methods* that enable rigorous reasoning about the behaviour of AISs, such that they can be used reliably in critical applications

Taking inspiration and input from all the bio-inspired learning algorithms, major way points on the Non-Classical Computation journey are

- a *general theory of learning systems* that includes neural, evolutionary, and immune systems as special cases
- use of the general theory to develop *more effective kinds* of learning systems, inspired by, but not based upon, any known biological processes

The Network model

The immune system network model is computationally inspired by the biological processes used to maintain a dynamic "memory" of immune responses, in a system where the lifetime of individual immune memory cells is on the order of weeks, yet the memory itself persists on the order of years or decades.

Investigations and a deeper understanding of the nature of scale free networks and their relation to complex systems is required, to allow a greater understanding of a variety of network type structures. In particular, the formalisation of IS molecular-cellular network by means of modern graph theory (small-world models, scale-free networks theorisation) should be pursued, to depict the topological features and attributes affecting the functionality of the network. Graph theory is one of the most effective and advantageous instruments for understanding the evolution of network systems and a comprehension of the basic principles of their structural organisation and evolution. As such, it is needed to find the best solutions to the problems of real world networks. This approach, proceeding from the formalisation of elements of the network and their interactions as nodes and links, allows structuring a topology whose characterising features can be derived from analytical and numerical solutions. Modern graph theory has already been successfully exploited for studies of the topological and connective features of existing real world networks like, for example, citations of scientific papers and networks of collaborations, WWW and Internet, biological networks as neural networks, metabolic reactions network, genome and protein network, ecological and food webs, world web of human contacts and languages, telephone call graphs, power grids, nets of small world components. As far as we know, a similar approach has never been applied to the study of realistic (beyond the "undistinguishable clones" hypothesis) IS network peculiarities (see also [1]). By exploring the nature of scale-free networks, immune system offer insight into *breaking the von Neumann paradigm* in terms of allowing for massive parallelism at sensible computational costs.

The biological models

Like many other biologically inspired computational ideas, the computer science and biology of immune systems are developing in parallel. The natural immune system, in particular, is exceedingly complicated, and not understood at all well. Additionally, the immune system does not act in isolation. Indeed, there are many interactions with other biological systems, such as the nervous systems and endocrine (or hormonal) systems. The interactions between these systems lead, in part, to the biological concept of *homeostasis*: a steady internal state. By exploring these ideas, there is the potential to break many of the well-held paradigms outlined in this challenge, such as the *algorithmic* and *refinement* paradigms. Limited work has already begun on this approach, with the development of a small controller for robots [68].

Currently, the field of AIS is limited to the development of algorithms *in silico*; questions are still to be asked similar to that of DNA and quantum computing, such as: it is possible to construct computational devices based on the chemical process inherent in the immune system? The immune system has a wealth of complexity for computation: rather than just extract metaphors from it, is it possible to exploit the biological mechanisms? The current discipline of AIS may have been inspired by biology, but it is painfully clear that AISs are but a pale shadow of the vast complexity of subtlety of the natural immune system. Computer scientists, mathematicians and

immunologists working together can ask, and answer, some deep and interesting questions. For example:

- How might we use the real immune system, and other real physical and biological systems, for computation?
- To what extent is the working of the immune system, and other biological systems, dictated by the physical substrate? Can all putative "immune" responses be realised on all substrates? Do some diseases exploit *computational* constraints of the immune system to defeat it?
- How can we use models to decide which parts of the biology are necessary for correct robust functioning, which parts are necessary only because of the particular physical realisation, and which parts merely contingent evolutionary aspects?
- How can we use nature inspired computation to build "better than reality" systems? What are the computational limits to what we can simulate?

Conclusions

AIS do not break all the classic computational paradigms: for example, they do not (yet?) use concepts from quantum physics. However, they do challenge some of the major paradigms. The selection model is a good exemplar for examining alternatives to the refinement paradigm, and the network model is an excellent exemplar for examining open network dynamics and emergence, necessary for a full science of complex adaptive systems.

Journey: Non-Classical Interactivity – Open Dynamical Networks

Dynamic reaction networks can have complex non-linear interactions, and feedback where reaction products may themselves catalyse other reactions in the network. They exhibit the emergent complexity, complex dynamics, and self-organising properties [6] [47] of many far-from-equilibrium systems. These systems, and others, can self-organise into regions "at the edge of chaos", neither too ordered nor too random, where they can perform interesting computations (or computation analogues). There are many dynamic network models that occur in biological and social systems, from Kauffman's autocatalytic networks [47], and genomic control networks, through dynamical models of neural networks and cytokine immune networks, to ecological food webs, and social and technological networks.

All these subject areas could benefit from better networks models [86]. Much of the existing mathematical network theory is restricted to static, homogeneous, structured, closed networks, since these are the simplest, most tractable models to work with. However, these are not realistic models of biological networks: for example, antibodies rove around the body (network, system, …) looking for the anomalies, and new kinds of attacks call for new kinds of defence. The journey is

to develop a pragmatic theory of *dynamic, heterogeneous, unstructured, open networks*

Dynamic: the network is not in steady state or equilibrium, but is far from equilibrium, governed by attractors and trajectories. (*Swarm networks* may offer insights to this kind of dynamics [12])

Heterogeneous: the nodes, the connections, and the communications can be of many different types, including higher order types.

Unstructured: the network connectivity has no particular regularity: it is not fully regular, or fully connected, or even fully random. Clearly there need to be *some* kinds of regularity present, but these are likely to be of kinds that cannot be reasoned about in terms of simple averages or mean field notions; they are more likely have fractal structure. Some recent advances in *Small World* networks offer intriguing new insights [8] [88].

Open (metadynamic): the structures are unbounded, and the components are not fixed: nodes and connections may come and go; new *kinds* of nodes and connections may appear.

A general theory of such networks would have wide applicability. Such a theory is a basic requirement of complex systems development in general, one application of which is pervasive, or ubiquitous, computing (the subject of another Grand Challenge). Such a theory a necessary way point for answering many challenging questions.

Computation at the edge of chaos. What are it capabilities? How can we hold a system at the edge, far from equilibrium, to perform useful computations? How can we make it self-organise to the edge?

Designed emergence. How can we design (refine) open systems that have desired emergent

properties? And do not have undesired emergent properties?

Open systems science. What are the fundamental properties of open systems? How can we predict the effect of *interventions* (adding new things, or removing things) to the system? How can we understand the effect of a gateway event that opens up new kinds of regions of phase space to a computation? How can we design a system such that gateway events, natural changes to phase space, can occur endogenously?

Computation as a dynamical process. Physical dynamical processes are characterized by motion in a phase space, controlled or directed by various attractors (so called because they "attract" the trajectory of the system to them). As various parameters of the system change, the shape of the resulting attractor space can also change, and so the trajectory may find itself being attracted to a different region of the space. [48], for example, uses these and related ideas to explain many features of organisms' behaviour, from gait patterns to learning and recognition tasks.

One might like to think of this dynamical behaviour in computational terms, with the attractors as "states" in the phase space, and the trajectories between them as "state transitions". This is a suggestive analogy, yet the conventional state transition model has a rather static feel to it. States and their transitions tend to be predefined, and the execution of the transitions has to be explicitly implemented by the computational system. Contrastingly, the attractors are natural consequences of the underlying dynamics, and new attractors and resulting trajectories are natural consequences of changes to that underlying dynamics. A dynamical system is relatively robust (a small perturbation to the trajectory will usually leave it moving to the same attractor), and computationally efficient (the computation is a natural consequence of the physical laws of the system, and does not need any further implementation beyond that of the dynamical system itself).

The challenge continues thus:

to develop a computational paradigm in terms of dynamical attractors and trajectories

Does the state transition analogy hold? What are the various attractors of a dynamical computation? Can a computation be expressed as a trajectory amongst various attractors, each changing as the result of some parameter/input? How can we encourage the system to move to a "better" attractor? How can we map the route through intermediate attractors that it should take? What are the programming primitives and higher level languages? What are the logics, reasoning approaches, and refinement calculi? What are the compilers and other development tools? What kinds of algorithms are most suited to this paradigm? What are the implementation mechanisms? How can we simulate these systems on classical machines?

Journey: Non-Classical Architectures – Evolving Hardware

This journey of Non-Classical Computation is

to develop (biologically-inspired) computing hardware that can adapt, evolve, grow, heal, replicate, and learn

Computation Models

Biological inspiration in the design of computing machines finds its source in essentially three biological models:

- ***phylogenesis*** (P), the history of the evolution of the species
- ***ontogenesis*** (O), the development of an individual as orchestrated by his genetic code
- ***epigenesis*** (E), the development of an individual through learning processes (nervous and immune systems) influenced both by the genetic code (the innate) and by the environment (the acquired).

These three models share a common basis: the genome.

Phylogenesis: evolution

The process of evolution is based on alterations to the genetic information of a species through two basic mechanisms: *selective reproduction* and *variation*. These mechanisms are non-deterministic, fundamentally different from classical algorithms, and potentially capable of providing astonishingly good solutions to problems that are formally intractable by deterministic approaches. Existing analytical and experimental tools are not designed for tackling such stochastic search algorithms, however. We need new tools and methodologies for generating novel results.

Phylogenesis already provides considerable inspiration for algorithm design, in the discipline of *evolutionary computation* [5] [59], which includes

genetic algorithms [35] [67] and genetic programming [7] [52]. It has yet to have such an impact on the conception of digital hardware, however. Koza *et al*. pioneered the attempt to apply evolutionary strategies to the synthesis of electronic circuits when they applied genetic algorithms to the evolution of a three-variable multiplexer and of a two-bit adder. Evolutionary strategies have been applied to the development of the control circuits for autonomous robots, and other research groups are active in this domain. Although technical issues pose severe obstacles to the development of evolvable electronic hardware, there is still much to be gained from evolutionary design given the appropriate hardware and software mechanisms.

Ontogenesis: growth

Ontogenesis concerns the development of a single multi-cellular biological organism. A set of specific mechanisms define the *growth* of the organism: its development from a single mother cell (*zygote*) to the adult phase. The zygote divides, each offspring containing a copy of the genome (*cellular division*). This continues (each new cell divides, creating new offspring, and so on), and each newly formed cell acquires a functionality (liver cell, epidermal cell, …) depending on its surroundings, its position in relation to its neighbours (*cellular differentiation*).

Cellular division is therefore a key mechanism in the growth of multi-cellular organisms, impressive examples of massively parallel systems: the $\sim 6x10^{13}$ cells of a human body, each one a relatively simple elements, work in parallel to accomplish extremely complex tasks. Development processes inspired by biological growth should provide relevant insights on how to handle massive parallelism in silicon. There are also great gains to be achieved by using ontogenetic mechanisms with regard to fault tolerance and reliability.

Epigenesis: learning

The human genome contains $\sim 3x10^{9}$ bases, yet an adult human body contains $\sim 6x10^{13}$ cells, of which $\sim 10^{10}$ are neurons, with $\sim 10^{14}$ connections. The genome cannot contain enough information to completely describe all the cells and synaptic connections of an adult organism. There must be a process that allows the organism to increase in complexity as it develops. This process, *epigenesis*, includes the development of the nervous, immune, and endocrine systems.

Epigenetic, or *learning*, mechanisms have already had considerable impact on computer science, and particularly on software design. The parallel between a computer and a human brain dates to the very earliest days of the development of computing machines, and has led to the development of the related fields of artificial intelligence and *artificial neural networks*.

Living organisms interact with their environment and respond to sensory inputs. In many cases this behaviour is learnt over a period of time, after which a specific stimulus will trigger the same, possibly context dependent, response. Such behaviour is mainly controlled by *spiking neurons* and their interactions. Novel hardware developments are being inspired by these observations.

A more recent addition in the general area of hardware systems and epigenetic processes are *artificial immune systems*. Here the sophisticated mechanisms associated with "fault tolerance" in nature have been adapted for electronic hardware system designs [13].

Complexity and Reliability

As systems become more complex it becomes increasingly difficult to provide comprehensive fault testing to determine the validity of the system. Hence faults can remain in a system, and manifest themselves as errors. Furthermore, faults may be introduced into hardware from external sources, such as electromagnetic interference. Components within a system can die; no transistor will function forever. These faults can ultimately cause a system to fail. The ability of a system to function in the presence of such faults, to become *fault tolerant*, is a continually increasing area of research.

Through millions of years of refinement, biology has produced living creatures that are remarkably fault tolerant. They can survive injury, damage, wear and tear, and continual attack from other living entities in the form of infectious pathogens. Biology manages to take huge amounts of potentially unreliable matter and use self-checking, self-repair, self-reconfiguration, multiple levels of redundancy, multiple levels of defence, even removing suspected cells, to develop complex biological organisms that continue to work in an extremely hostile environment.

While we consider our systems to be complex, how might one compare a 747 jet with the complexity of an ant, of a 2 year old child, let along the human nervous system, the human immune system? As technology moves towards nano- and quantum-devices the current issues relating to complexity will appear trivial. How might we design systems with such parallelism, such complexity? How will we ensure that they continue to function correctly over long periods of time, and in unpredictable environments?

The Journey

How can we "evolve" systems of the complexity we will be dealing with produced by technology in 10-20 years? How can we "grow" systems high-reliability designs? How can we build systems that can learn from, and adapt to, their environment in a way that improves their performance, that can become immune to attacks, both internal and external, that can learn to use all of the resources available to them.

What is the effect of "design by evolution" on silicon systems whose microscopic computing paradigm is itself biologically-inspired? What is the interaction between evolutionary processes and the natural imperfections in non-digital chips? How can evolutionary processes capture functionality from such an imperfect computing substrate that conventional design cannot? In particular, when the silicon system is itself adaptive and can "learn", what is the optimal interaction between "design by evolution" and subsequent adaptation for specific purpose? Natural systems use both methods: how can silicon computation or its successors benefit?

Journey: Non-Classical Architectures – Molecular Nanotechnology

Molecular Nanotechnology presents research challenges that will lead to a greatly enriched and more general science of computation. Safety and dependability will present unprecedented demands; the science will be responsible not only for robust design to meet these demands, but for robust analysis that shows they have been met.

Background and context

Nanotechnology is the design, development and use of devices on the nanometre (atomic) scale. Here we are not so much concerned with nano-scale artefacts that take the current trend of miniaturisation a few orders of magnitude further. Rather we are interested in *active* physical nano-devices that themselves manipulate the world at their nano-scale in order to manufacture *macroscopic* artefacts. This is Drexler's [27][28] vision of nano-scale *assemblers* that build (assemble) macroscopic artefacts. (Such assemblers are often known as *nanites* or *nanobots*.)

In order for nanites to build macroscopic objects in useful timescales, there needs to be a vast number of them. A starting population of a few nanites assembles more of their kind, which then assemble more, with exponentially growing numbers. Once they exist in sufficient numbers, they can build, or become, the macroscopic artefact. This view of nanotechnology promises many awe-inspiring possibilities.

Some argue that such a technology is too good to be true, or at least question the detail of Drexler's predictions. But one should note that there is no conclusive counter-argument to them; indeed, proteins and their associated cellular machinery routinely assemble macroscopic artefacts, or, to use more biological terminology, they *grow organisms*. Here we discuss computational structures that will be relevant whenever *some* technology for sophisticated populations of nanites is achieved, even if not all that has been predicted.

In principle it is possible for nanites to assemble *any* physical artefact, by carefully controlled placement of every component atom (possibly requiring the use of much scaffolding). But in general this is infeasible: in the worst case it could need the *global* control and choreography of the behaviour of every individual nanite. A more feasible approach is to exploit mainly *local* cooperation between suitably-programmed neighbouring nanites, possibly mediated by their shared local environment (which also more closely mirrors the way biological organisms grow).

In order for nanotechnology to be possible, the initial nanites must be fabricated somehow. This complex engineering problem requires collaborative research by physicists, chemists, engineers, and biologists. To the extent that the nanites need to be *programmed* to perform their assembly tasks, computer science (CS) also has a crucial role. We need to develop capabilities to design, program and control complex networks of nanites, so that they safely and dependably build the desired artefacts, and so that they do *not* accidentally build *un*desired ones.

Initial CS research needs to focus on potential ways of designing and assembling artefacts in ways that can be described in terms of predominately local interactions, that is, in terms of the *emergent properties of vast numbers of cooperating nanites*. This requires *analysis* of emergent behaviour; given the orders of magnitude involved, this can be done only with a hierarchy of computational models, explaining the assembly at many different levels of abstraction.

Required computational advances

What CS theory and practice do we need in order to be able to design, program and control networks of nanites?

Emergent properties

We need a pragmatic theory of *emergent properties*.

In much the same way that an organism is an emergent property of its genes and proteins (and more), the assembled artefact will be an emergent property of the assembling nanites and their programming. In general, this problem is computationally irreducible, that is, there are no "short cuts" to understanding or prediction, beyond watching the behaviour unfold. Thus reasoning about the precise behaviour of arbitrary networks with a number of nodes comparable to the number of cells in the human body ($\sim 10^{13}$) is (currently) well beyond the state of the art. However, inability to solve the general problem, in principle or in practice, does not prevent exploration of large classes of specific and interesting problems. So we merely need a *sufficient* theory, one that enables us to design nanites to build the many artefacts of interest, and to analyse them for safety and dependability. Certain classes of useful emergent properties may well be tractable to reasoning. For example, many organisms contain emergent hierarchical branching structures, such as arteries, lungs, nervous systems, and, of course, prototypical tree branches. Such emergent structures are particularly straightforward to "program", as evidenced by L-systems [80].

Growth and Development

We need a pragmatic theory of *development and growth*.

A population of nanites first "grows" a vastly larger population, then "grows" the artefact in question. Again, we need a *sufficient* theory of growth – to enable us to reason about structures that are the result of a growth process.

Biological insights from embryology and development will be fruitful here, and the relevant ideas need to be abstracted and adapted for nanite assemblers. This "artificial development" also has its own properties: for example, the use of scaffolding will probably be much more important.

Which features of biological organisms are consequences of growth in general, and which are consequences of "wet-ware" growth, and so are different in assembled hardware? What constraints are there in the growth process: is it possible to "grow" a *cooked* steak *ab initio*, or must if first be grown raw (isolated, or as part of a cow), and then chemically modified?

Complex Networks

We need a pragmatic theory of *dynamic, heterogeneous, unstructured, open networks*, as espoused in the existing Journey: *Non-Classical Interactivity – Open Dynamical Networks.*

Complex Adaptive Systems

All these CS advances mentioned above would have application well beyond nanotechnology. All are basic requirements for the general area of *Complex Adaptive Systems*, of which nanotechnology is but one exemplar. Real world examples of CASs include swarms and flocks, ants, immune systems, brains, autocatalytic networks, life, ecologies, and so on. Artificial CASs include complex control systems (industrial plants, Air Traffic Control, etc), eCommerce supply chains and webs, telecoms systems and the Internet, and ubiquitous computing with its hordes of communicating smart devices, economic systems, and so on.

Behavioural Modelling

The pragmatic theories for Complex Adaptive Systems, above, must be developed in response to the challenge of nanotechnology, but they need not start from scratch. During the last two or three decades computer scientists have eroded the boundary between programming, which *prescribes* behaviour of a system, and modelling, which *analyses* it. This trend arises naturally from a change of emphasis, from stand-alone computers doing one thing at a time to *distributed* systems – networks of devices each acting independently, with no centralised control. The resulting computational models are in varying degrees logical, algebraic, non-deterministic, stochastic. They have been effectively used to analyse programming languages and communication disciplines. They have also been applied to computer security, mobile phone systems, behaviour in ant colonies, business processes, and signal transduction in biological cells.

A large system such as the Internet can be modelled at many levels of *abstraction*, correlated where possible with the structure of the system. At the higher levels, the analysis of agents' behaviour need not depend on the underlying technology used to realise them. A natural research direction is therefore to extrapolate existing CS models to nanosystems where, despite orders of magnitude increase in population size (compared with, say, the Internet), many of the same general principles of emergence and behaviour should apply.

At the lowest levels of abstraction, which may be called *embodiment*, the analysis of agents' behaviour depends crucially the underlying technology used to realise them. For example, individual nanites are made of only small numbers of atoms, so a one-atom mutation to a nanite – caused by faults in manufacture, by other nanites, by random impact of cosmic rays – could have a dramatic effect on behaviour. In order to reason about the kinds of changes that mutations might make (to reason about the "adjacent possible" [47] of the nanite), it is essential to know the detailed make-up and characteristics of the system undergoing mutation.

Close cooperation is therefore needed among many research disciplines, of which CS is one, in order to understand nanopopulations fully. From the CS viewpoint, the gain will be a greatly enriched and more general science of computation.

We continue this section by summarising some of the concepts, theories and tools that CS can bring to the cooperation at the outset. We cite only a selection from the large literature.

Stand-alone computation

Before distributed computing systems became the norm, much computing research laid foundations for the models and tools that those systems need. A start was made in establishing the *verification* of computer programs as an activity in formal logic [31][39]. Tools for computer-assisted verification, especially for computer hardware designs [36], were pioneered. The status of computer programs as mathematical descriptions of behaviour was established [83]. Theories of types began to emerge as a powerful aid to behavioural analysis as well as to programming [81]. Even in the 1940s, von Neumann's model of self-reproducing cellular automata anticipated some of the central ideas of nanotechnology [69].

Abstract machines and process calculi

The first model to capture the complex interplay between non-determinism and concurrency in distributed systems was Petri Nets [77], these nets were designed information flow in natural as well as man-made systems. In the early eighties, algebraic process calculi [14][40][63] were designed to model interactive systems hierarchically, and to model their behaviour abstractly. The Chemical Abstract Machine [9] captured the spatial structure of systems. The π-calculus [64][65] and mobile ambient calculus [16] made a further step in modelling systems that can reconfigure both their spatial arrangement and their connectivity.

These models have influenced the design of programming and specification languages, for example LOTOS, occam and Handel-C, and Ada. They have been developed to model systems stochastically, and to deal with hybrid discrete/continuous systems. Recently their theory has been seen to extend to graphical models that are *a priori* suitable for populations of agents such as nanites.

Logics and Tools

Allied to algebraic calculi are new forms of mathematical logic, especially modal logics, specially designed to specify the properties that an interactive system should satisfy. Well-known example are dynamic logic [79], temporal logic [78], the temporal logic of actions [58] and the mu calculus [55]. These logics often have a close link with algebraic calculi; an algebraic term denotes (part of) a system. while a logical formula says (in part) how it should behave. This underlies a successfully applied incremental methodology for system analysis; one verifies more and more properties of more and more parts (even the whole) of a system. Such verification is aided by software tools: model-checkers that can automatically verify properties of fairly complex finite-state systems [20]; and semi-automated tools that can perform verifications with human guidance [21].

Safety and dependability

Nanites can disassemble, as well as assemble, structures. This has led to the notion of the so-called "grey goo" problem: nightmare visions of hordes of rogue nanites disassembling the wrong things, disassembling people, or even disassembling the entire planet. It is potentially the ultimate terrorist weapon.

Even if nanites are not deliberately engineered to be destructive, such objects will "naturally" appear in any replicating swarm of nanites. We are dealing with such vast numbers of nanites that some will spontaneously "mutate". Given the three features of reproduction, variation, and selection, some form of evolution will inevitably occur, leading to populations of "adjacent possible" undesigned nanites. Computer science, allied with biology, is crucial to the task of investigating and understanding these artificial evolutionary processes, and the defences we can design against them.

Dependability – the quality of a system that justifies its use even in critical conditions – is already a topic of extensive research in computer science. It involves mathematical analysis, as in the case of program verification and computer security; more widely, it involves making systems aware of, and able to report upon, their behaviour. It cannot exist without good modelling. The modelling of nanopopulations with dependability

in mind, given their emergent properties and the inevitability of mutation, offers a huge challenge to CS.

Conclusion

Nanotech assemblers offer the promise of fantastic rewards. Some forms of nano-assemblers may well be exploitable and exploited in many ways without much CS input. Before we can achieve the full promise, however, there are many hard Computer Science problems to solve, concerning the design of emergent properties, the growth of physical artefacts, the programming and control of nanites, and defences against the "grey goo" and other safety critical scenarios.

Journey: Non-von Architectures – Through the Concurrency Gateway

This journey of Non-Classical Computation is

to enable concurrency to be a fundamental modelling and programming concept, with a clean and simple conceptual model, and efficiently implemented

Breaking the von Neumann paradigm

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 GC7 hint at something stronger: central points of control actively work against the logic and efficiency of whatever we are trying to control, model, or understand.

Today, concurrency is not considered a fundamental concept, to be used with everyday fluency. It is considered an advanced topic, to be avoided unless there is no other way to obtain specific performance targets.

Classical concurrency technologies are based on multiple threads of execution plus various kinds of locks to control the sharing of data between them; get the locking wrong and systems will mysteriously corrupt themselves or deadlock. There are also performance problems. Thread management imposes significant overheads in memory and run time. Even when using only 'lightweight' threads, applications need to limit their implementations to only a few hundred threads per processor, beyond which performance catastrophically collapses.

Yet air traffic control over the UK requires the management of far greater concurrency than standard practice will directly and safely and simply allow. Common web services need to be able to conduct business with tens of thousands of clients simultaneously. Modelling even the simplest biological organisms quickly takes us into consideration of millions of concurrently active, autonomous, and interacting, agents.

Limited by programming and performance constraints, we compromise on the degree of concurrency in our application design and implementation. The compromises add significant complexity that, combined with the semantic instability of the concurrency mechanisms we do practice, lead to mistakes and the poor quality, late delivery and over-budget systems that are accepted as normal – for now – by our industry and its customers.

We urgently need more natural models and implementations of concurrency. Fortunately, we have them. Pushing through this particular gateway, by the mainstream computing community, will additionally help establish a mindset for the much grander challenges.

Hypothesis

All computer systems have to model the real world, at some appropriate level of abstraction, if they are to receive information and feedback useful information. To make that modelling easier, we should 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.

Our thesis is that computer science has taken at least one wrong turn. Concurrency should be a natural way to design any system above a minimal level of complexity. It should simplify and hasten the construction, commissioning, and maintenance of systems; it should not introduce the hazards that are evident in modern practice; it should be employed as a matter of routine. Natural mechanisms should map into simple engineering principles with low cost and high benefit. Our hypothesis is that this is possible.

We propose a computational framework, based on established ideas of process algebra, to test the truth of the above hypothesis. It will be accessible from current computing environments (platforms,

operating systems, languages) but will provide a foundation for novel ones in the future.

Hoare's CSP [40] has a *compositional* and *denotational* semantics, which means that it allows modular and incremental development (*refinement*) even for concurrent components. This means that we get no surprises when we run processes in parallel (since their points of interaction have to be explicitly handled by all parties to these interactions). This is not the case for standard threads-and-locks concurrency, which have no formal denotational semantics, and by which we get surprised all the time.

We need some extensions to CSP to describe certain new dynamics. We want to allow networks of processes to evolve, to change their topologies, to cope with growth and decay without losing semantic or structural integrity. We want to address the mobility of processes, channels and data and understand the relationships between these ideas. We want to retain the ability to reason about such systems, preserving the concept of refinement. For this we turn to Milner's π-calculus [64].

The framework will provide highly efficient practical realisations of this extended model. Its success in opening up the horizons of GC7 will be a long term test of the hypothesis. Shorter term tests will be the development of demonstrators (relevant to a broad range of computer applications – including those that are of concern to GC1, GC4 and GC6) with the following characteristics:

- they will be as complex as needed, and no more (through the *concurrency in the design* being directly delivered by the *concurrency in the implementation*)
- they will be scalable both in performance and function (so the cost of incremental enhancement depends only on the scale of the enhancement, not on the scale of the system being enhanced)
- they will be amenable to formal specification and verification
- the concurrency models and mechanisms in their design and implementation will be practical for everyday use by non-specialists: concurrency becomes a fundamental element in the toolkit of every professional computer engineer
- they will make maximum use of the underlying computation platform (through significantly reduced overheads for the management of concurrency, including the response times to interrupts)

Current State of the Framework

Over the past ten years, the Concurrency Research Group at Kent [90] has been laying the foundations for such a framework. They have developed, and released as open source, concurrency packages for Java (JCSP), C (CCSP), C++ (C++CSP), J# (J#CSP), and occam (occam-π). These all provide the mobile dynamics fed in from the π-calculus.

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. 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, yet with exceptionally low performance overheads and all the safety guarantees of classical occam and the lightness of its concurrency mechanisms. It supports the dynamic allocation of processes, data and channels, their movement across channels and their automatic de-allocation (without the need for garbage collection, which otherwise invalidates real-time guarantees). 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 and data 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.

The low performance overheads mean that dynamic systems evolving hundreds of thousands of (non-trivial) occam-π processes are already practical on single processors. Further, occam-π networks can naturally span many machines: the concurrency model does not change between internal and external concurrency. Application networks up to millions of serious processes then become viable, on modest clusters of laptops. Moore's Law indicates that in the next few years networks of tens of millions of (non-trivial) processes will become possible.

Enabling other Journeys

Such a platform provides an enabling technology for modelling emergent properties, including those mentioned above, such as Open Dynamical

Networks, Molecular Nanotechnolgy, Artificial Immune Systems.

Hierarchical networks of communicating processes are particularly suitable for these problems. The languages used to support modelling and simulation must be simple, formal, and dynamic, and have a high-performance implementation. The models must be simple, and amenable to manipulation and formal reasoning. The topologies of these networks of agents will evolve, as they support growth and decay that comes from agents moving, splitting, and combining. Individual agents must be mobile, and aware of their location and neighbourhood. Simulations will require very large numbers of processes, so the implementation must have minimal overhead.

occam-π is a good candidate for modelling and programming such systems: it is robust and lightweight, and has sound theoretical support. It can be used to construct systems to the order of 10^6 processes on modest processor resources, exhibiting rich behaviours in useful run-times. This is enough to make a start on our journey..