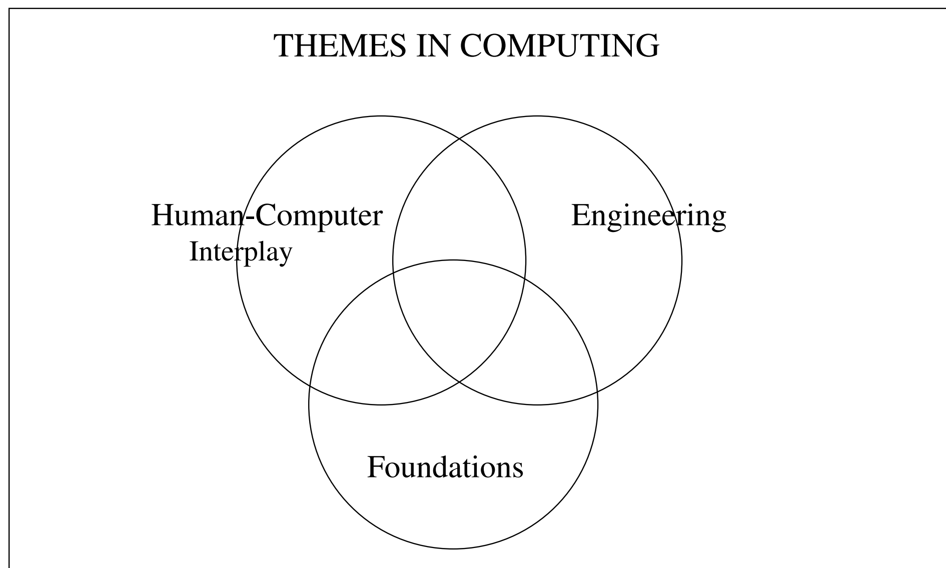
Computing in Space

A lecture by Robin Milner, for the opening of
the Computer Laboratory's William Gates Building
at the University of Cambridge, on 1 May 2002.

Introduction When I joined the Computer Laboratory in '95, and especially when I took over as Head in '96, I tried to comprehend what goes on here. This was a rewarding task; I learned a huge amount just by getting a grasp of how all things we do in the Lab relate to each other. In a very broad sense, one can classify the whole intellectual endeavour of Computer Science into three large overlapping themes: Engineering, Human-Computer Interplay, and Foundations.



Within Engineering, you have not only hardware but software engineering, and also communications; within Human-Computer Interplay you have not only linguistic and graphical interfaces, but also arti£cial intelligence; within Foundations you have not only the structure and complexity of algorithms, but also the semantics underlying computational processes and the logical tools to verify them.

It's a tribute to all my colleagues, and especially my immediate predecessor as Head, Roger Needham, that one could £nd ten or a dozen substantial activities in the Lab – all successful – spanning this space in an extraordinarily balanced way, and informing each other. At that time I drew a diagram of these research groups and their interaction on my blackboard, for a visit by Geoff Hoon, then a science minister (now Secretary of State for Defence). I think it was effective, and the same picture works just as well with incoming PhD students. The story is much the same today, mutatis mutandis; we can be intensely proud of our broad and equal span of the subject.

On an occasion like this, we ought to wonder what we might achieve in, say, the next two decades. I'm not quali£ed – perhaps none of us is – to predict across such a wide spectrum of activity. But I shall stick my neck out, which is rare for a theoretician, and speculate on one way in which we might change the face of our subject in response to modern technology, rather than simply adapt to it. More forcefully, in this lecture I put forward a Grand Challenge which Computer Scientists might address, and this Lab is well placed to address it.

COMPUTING IN SPACE – EXAMPLES

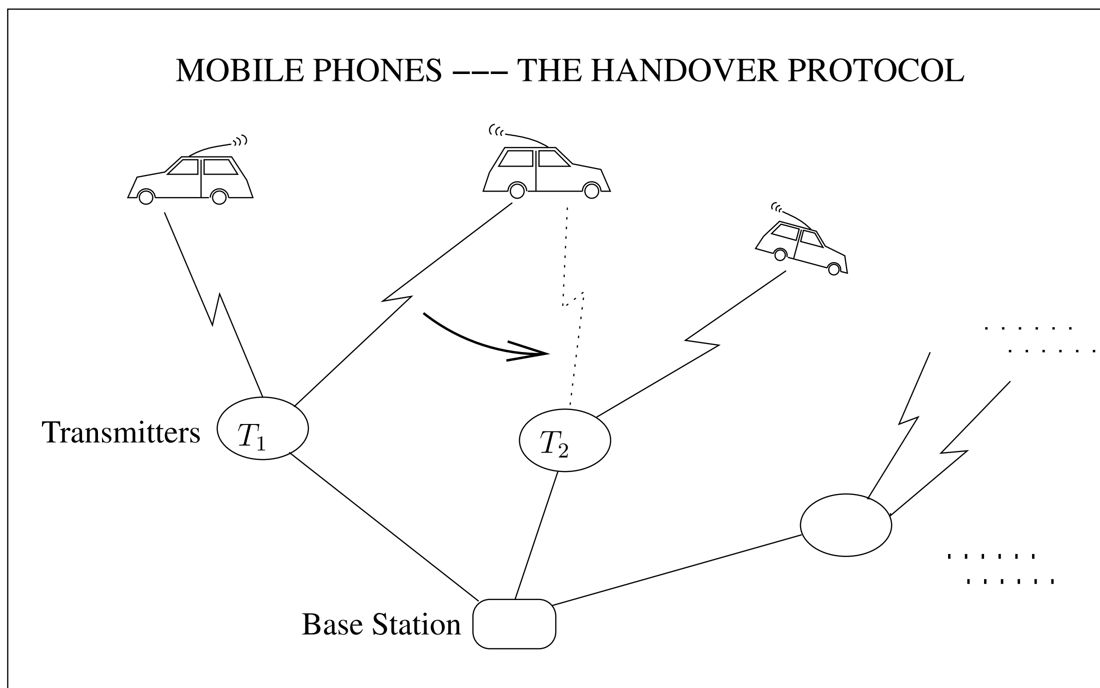
- Multimedia Infrastructure for a Building
- Mobile Telephone Systems
- Molecular Computers in Biology

The Building Before we moved into this new building we had long discussions about how we could link up all the lecture halls, teaching rooms and meeting rooms, using advanced audiovisual technology, to give a rich and ¤exible platform for teaching and research – and at the same time provide a live laboratory for our own research into computational platforms for multimedia. My colleague Peter Robinson had, and no doubt still has, a strategy for this. We had hoped for £nancial support from the Joint Infrastructure Funding initiative. This didn't mature, but I would not be at all surprised if this building were to become involved in studies of how a community's interactions in a complex physical space can be supported more and more deeply by computation. As a very simple example, suppose that you are starting a lecture and the projector is missing; then instead of phoning the caretaker to £nd it, why not send a message to the projector itself saying "Where are you?" or better still "Please arrange that you appear in Lecture Room 1 as soon as possible." (At least the £rst part of this was essentially done by Professor Andy Hopper's so-called Active Badge project, for locating people, when we lived on the New Museums Site.) The more adventurous possibilities are endless; not the easiest part is working out what the community actually wants.

Handover Protocol Ten years ago, my colleagues and I worked out a so-called Calculus of Mobile Processes, which we called the Pi Calculus, to explore the complex interplay between computation and communication. Later in the lecture I'd like to say more about this kind of work. For now, I'd like just to illustrate what it was aiming at, by a simple experiment we did with Swedish Telecomm to test out the calculus.

Imagine a mobile phone system (see the picture below); there is a single base station, a number of £xed transmitters around the country, and as many mobile phones as you like. Each phone is connected – using some wavelength – to the transmitter, say T_1, closest to it. When the phone moves the signals get faint, and T_1 alerts the base station of this. After deciding to have the phone connected to another transmitter T_2, the base station initiates what is called the Handover Protocol. It works like this: the base station identi£es a new channel on which the phone will interact with T_2; it tells this to T_1, which tells the phone and severs connection with it; the base station also tells the channel to T_2, which then enters dialogue with the phone.

What the Pi Calculus does is to represent the *potential* interactive behaviour of each agent in the system by a little set of equations; then one can derive equations describing the *actual* interactive behaviour of the whole. Using the mathematics of the calculus, one can immediately exhibit properties of the Handover Protocol. A simple example is an *invariant*, for example that no phone is ever connected to more than one transmitter. Proving these things amounts to a veri£cation of the Protocol. Of course, this is a simple task. All the same, to verify the same properties for a protocol written in C, say, is rather like using a nut to crack a sledge-hammer . . .

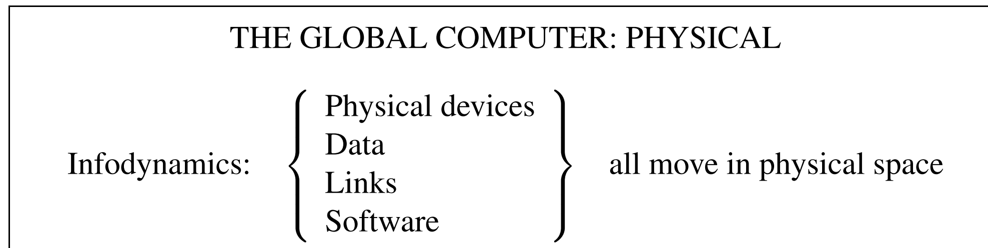


Nanocomputers Professor Ehud Shapiro, of the Weizmann Institute in Israel, gave a talk a few weeks ago at the Royal Institution in London about recent work published in the Journal Nature, in 2001. He and his colleagues have built molecular computers out of DNA and enzymes. So far these automata do only simple things, and slowly; but 10^{12} (a million million) of them can live in a tenth of a litre of water, at room temperature, consuming virtually no energy. His vision is to build a general purpose computer which can be installed – in the plural – within the human body, identifying malfunctions and correcting them. Of course there is a long way to go; in the body's space there would have to be a whole organisation of these machines, coordinating with the body's own mechanism, both for the purpose of diagnosis and to put a remedy into effect.

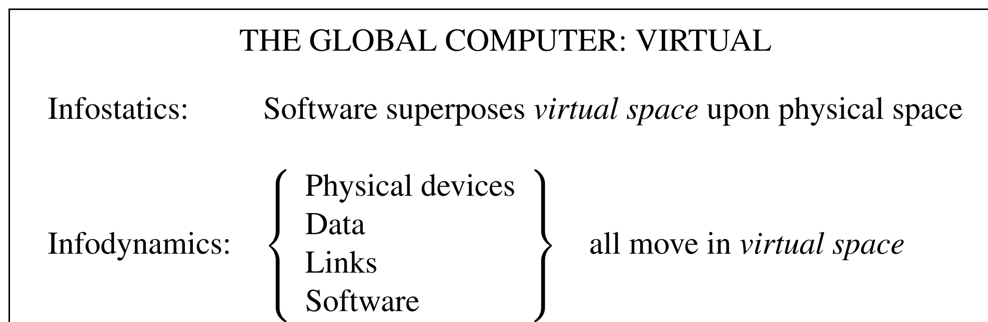
Shapiro has a second project, related but so far independent, to model biological processes and information pathways. Traditionally, differential equations or Monte Carlo methods have been used, but these do not capture the spatial structure of the processes and pathways. Shapiro is therefore using recent computer science models of concurrent mobile processes, including the Pi Calculus which I have just mentioned, and also the Ambient Calculus of Cardelli and Gordon at the Cambridge Microsoft Laboratory. By adding stochastics he is able to get informative models of such things as signal transduction in cells.

Space and Movement These three examples – the building support, the Handover Protocol and the biocomputers – show us that communication across space has become inextricably bound up with computing. In a case like the Handover Protocol we can even say that the process, which involves many communications, *actually is* a computation; for an interaction between neighbouring agents is as much a computational primitive as, say, writing a symbol onto the tape of a Turing machine, or fetching an operand from a register in an original stored program computer such as the EDSAC of Sir Maurice Wilkes. With the advent of the Internet, and the worldwide web on top of it, this interactivity has been multiplied a million fold. Quite simply, a *global computer* has come

into being. What are the events in this computer? It's all to do with movement, and I shall call it *infodynamics*.



We have illustrated most of these movements; for example, in the Handover Protocol a link was moved from one transmitter to another. As for software moving, think of viruses, and also the applets that we download, enlivening a website. But as you look deeper, things get more dif£cult and *much* more interesting. For by means of software – the critical component in all of this – a *notional*, or *virtual*, space is laid on top of the physical space; that's what the worldwide web does for us! The physical separation of things – whether physical devices, data or software itself – is hidden from us in virtual space, where they may appear to be contiguous. Conversely neighbours in physical space may be totally unconnected in virtual space, like two adjacent mobile phone users on a train. And there is movement in the virtual space too. To go a step further, one can see *all* computation in the global computer as movement, or unending recon£guration, in this totally new hybrid between physical and virtual space, which I shall call *infospace*.

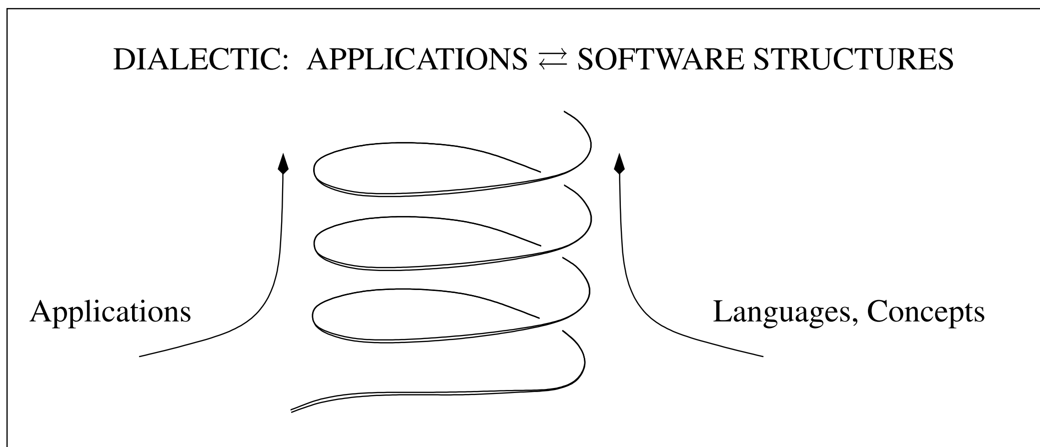


This is a lot to swallow! It does somehow re¤ect our experience on the Internet, even though we don't have to think deeply about infospace when we are exploring it, any more than we think deeply about physical space when exploring *that*. But note one detail: this structure is in a sense *universal*, for every physical entity can be modelled by a virtual one; so parts of the global computer – even the whole – can be modelled, virtually, within it.

Is it really true that physical devices can move in virtual space? Yes; a hand-held device, on receipt of some password, can enter a privileged virtual *domain* where new interactions become admissible. Can software really move in the virtual space created by software itself? Well, yes; an applet, say, may arrive through physical space and appear in your own virtual workspace. It may then be quarantined within a virtual *£rewall* while it is checked out for reliability in some way, and only then released from the £rewall; or, for security, it may be kept within a *wrapper* which mediates and monitors all interaction with it. These semi-technical terms – domain, £rewall and wrapper – connote some sort of control; this can be interpreted as controlling a spatial region.

Progeny of infospace We have to realise that the father of this hybrid infospace is network technology; networking is what has brought it to life. Pursuing the analogy, infospace has a mother too – the *software space* which has grown up with the stored-program computer over the past £fty years. This is the space explored by algorithms, data structures and programming languages, which were in turn brought to life by stored-program computers such as the EDSAC £fty years ago.

It's worth tracing one thread in this growth of software space. This will show that its familiar structures – the data structures and procedure structures of conventional programming – were not delivered along with the idea of a stored program, but actually developed through a cycle of applications, formulation, wider application and so on. Of course, the cycle is really a spiral:

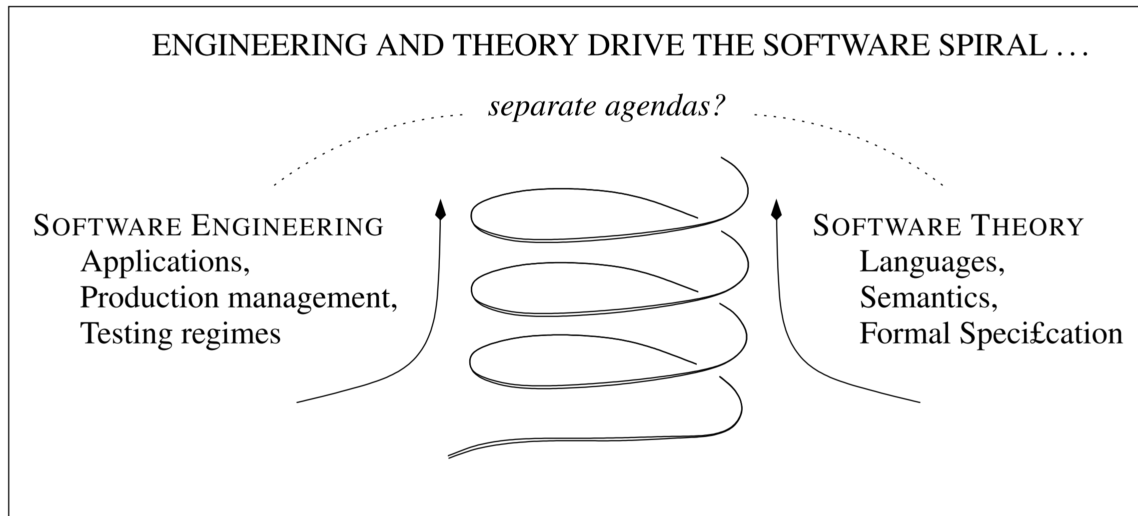


This spiral is well illustrated by so-called *object-oriented programming*. There were faint beginnings of this idea even in the language called ALGOL 60, forty years ago. ALGOL 60 had its notion of procedural (or subroutine) structure; for example, a procedure for £nding the shortest route between any two given cities, say Cambridge and Birmingham, could be used on successive occasions for many pairs of cities. The feature which ALGOL 60 pioneered was the idea that any procedure could usefully exploit its previous experience. The language allowed a procedure to have what was called its *own variables*; these gave it a memory from one invocation to another, so that it could act more ef£ciently on successive occasions. In the case of £nding shortest routes, its memory today could recall certain shortest sub-routes which it worked out yesterday for another purpose.

Thus computational procedures achieved real existence as *computing agents*, by being able to record their past. This notion struggled into the culture incrementally over the next two decades. One challenging application was in the area of computer simulation of real-world processes, like production lines. These simulating agents developed, via the language Simula, into the so-called *objects* of object-oriented programming. They became a new model of software behaviour which had a dramatic in¤uence, both on software engineering and on the further design of programming languages. Note the obvious spiral in mutual development of the model and the real world of applications. There are many other examples of software concepts evolving in this way.

Software Engineering Before getting up to the present date, and asking what happens next, I want to digress a little, and look at the effect this spiral has had upon the software industry. The rise of the spiral – i.e. the change in how we think about applications and program them –

has been dramatic, but the growth of demand for software products, enabled by the advance of technology, has been unprecedented and almost overwhelming. Software houses have been forced to rush products out, in order not to lose the market. Even if they could pause and contemplate the programming models, looking for theories on which to base rigorous methodologies, they would not have found those theories complete.

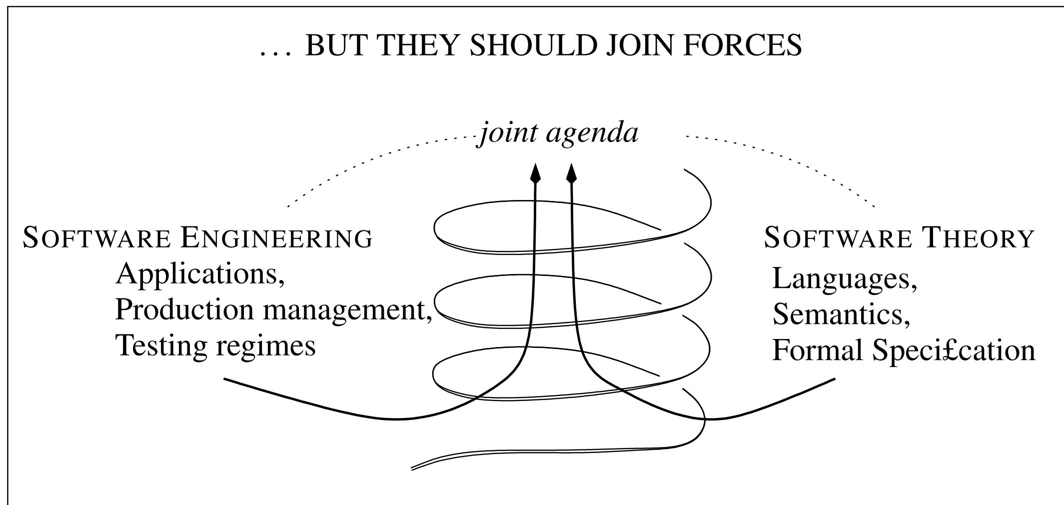


So on the one hand, computational theories have been linked to the spiral via languages and models. In exceptional cases they have in¤uenced applications directly, for example through spec-i£cation methodologies and analytical tools. On the other hand, software engineering has placed itself £rmly on the other side of the spiral. It has concentrated on disciplines for software *testing*, and upon *management* of the software production process, rather than upon the nature of its raw material, the software. It's a bit like carefully designing the production and testing regime in a car factory, without paying much attention to the properties of the steel that cars are made of, and the fuel they use.

Furthermore, there has rarely been time to *document* software products in any accurate way. This has led to some amazing statistics. About £ve years ago it was estimated that between 80% and 90% of software engineers' time was spent re-engineering old software, now called *legacy* software, to adapt it to new requirements. This re-engineering often consists in poring over millions of lines of old code, trying to £nd the right way to change it, because the documentation does not help, and – even after the changes – not daring to jettison any part of the 'old' code because it may still get used in a way that isn't evident. Thus the pile of legacy software grows, and presents a bigger challenge for the next time it has to be changed.

Who's at fault, and what to do? It is tempting, but wrong, to blame companies or academics for this. The fact is, the pressures of opportunity and the market have driven things out of control. Moreover, informed lay opinion seems not to expect that the solution will lie in £nding a rigorous science of software. For example, the Economist in its recent Technology Quarterly bemoans the current state of software reliability, but the furthest it goes in discussing remedies is to point to the ef£cacy of testing-cum-management techniques such as extreme programming or the £ve-step capability maturity model.

It would be downright foolish to suggest, or hope, that a rigorous scienti£c model can *replace* a management-cum-testing regime for software. But it is a Grand Challenge for the academic-industrial partnership, over the next two decades, to join a scienti£c model with software methodology, each informing the other in the way that *applications* and *languages* inform each other – in other words, taking up their position at the centre of the spiral, where they should be:



There are two things in our favour here:

- First, the world of web applications, e-business etc is so *different* that old software just can't be used; things are being done in new ways. So if we work quickly enough we may be able to build scienti£c models before piles of inscrutable new software build up;

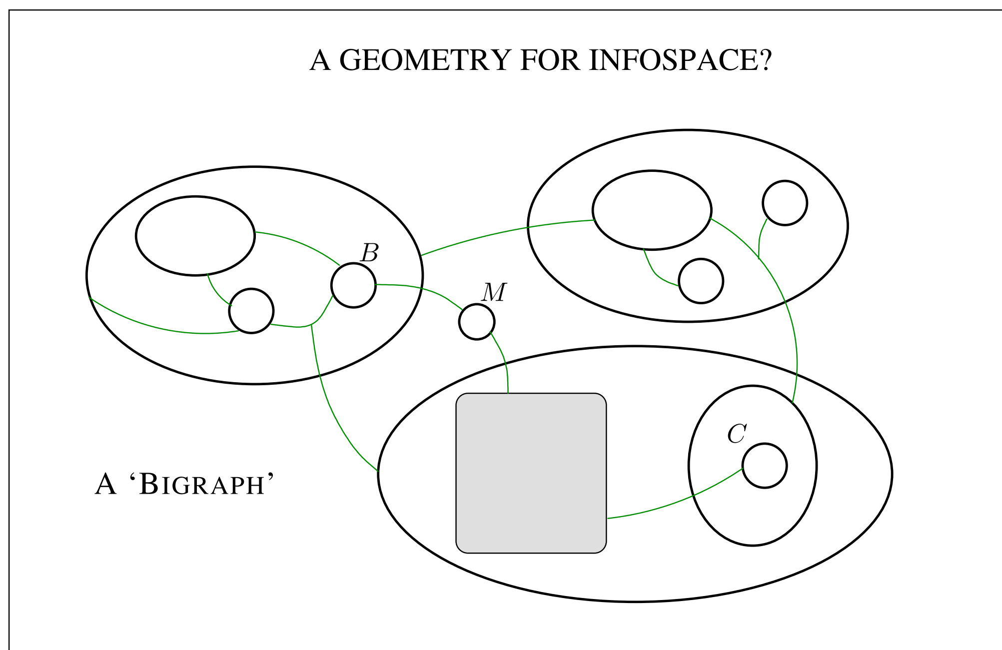
- Second, distributed and interactive computing is so *dif£cult* that we may just be shocked, or forced, into a more scienti£c habit.

Modelling the hybrid infospace On that note let us return to infospace or the global computer, and let's ask what an underlying model of such a huge, interactive and spatially distributed system might look like.

The concepts which evolved in software before the worldwide web, especially objects and concurrent processes, go some way towards a model of the global computer. We have become somewhat familiar, both in our languages and in our models, with handling interactions between processes – via such notions as concurrent threads (say in Java). So there is a temptation to keep these concepts unchanged, but just to add a notion of *place*, or *locality*, which will do for both the physical and the virtual space, together with some way of representing movement among these localities. But this is wrong! Why? Because it violates the principle of Occam's razor, that you should create no more entities than you need. For there are prototypical localities lurking around every corner in traditional software; many of them are re¤ected in the levels of syntactic structure of our programming languages.

So we would like to weave space into the very basis of our new model. Instead of merely using spatial metaphor to explain or teach our ideas and theories, let us make the model itself

unashamedly geometric. The time is ripe for the closet geometry of computing to, as it were, *come out*. Let me show you what I mean in a diagram or two, representing one approach among several that computer scientists are now proposing.

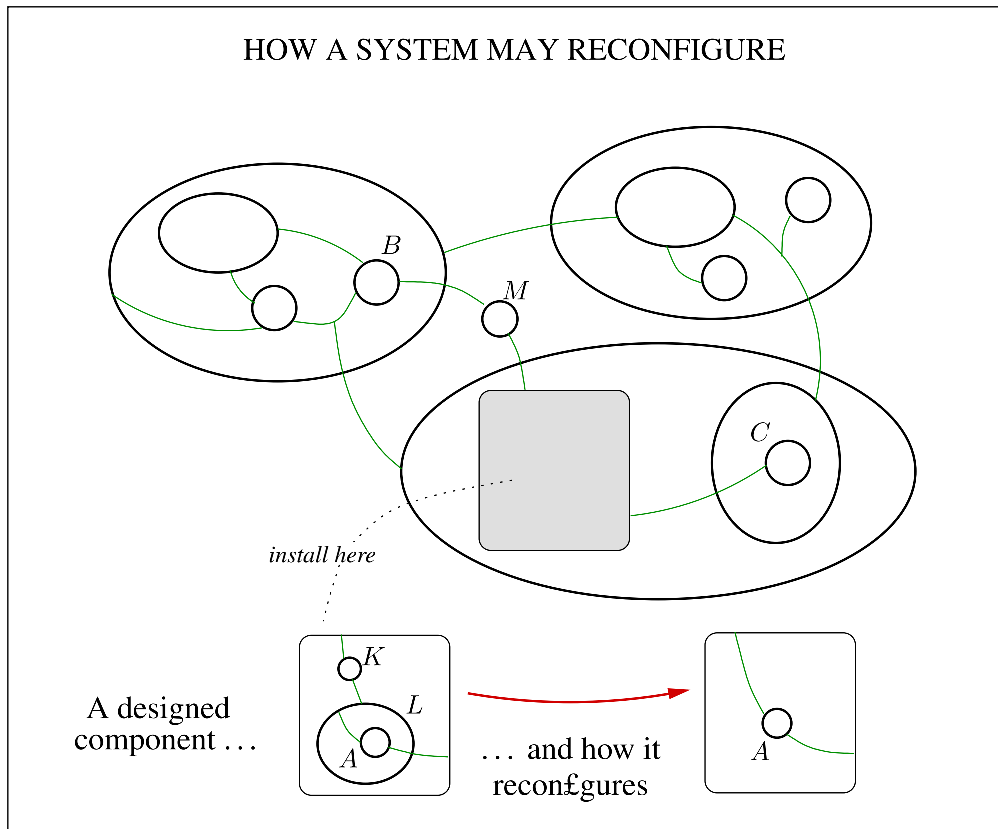


This diagram shows what I call a *bigraph*. A single bigraph, billions of times larger, would represent the state of the global computer at one instant, so the bigraph at the next instant would be different. The bigraph, and the way it changes from instant to instant, is the whole story of infospace. Primitive software actions are represented in just the same way as movements of devices in physical space are represented. These, and only these, are the computational events.

Details of bigraphs The stuff of a bigraph is its *nodes*; the ovals and circles. A node may be a city, a building, a computer, a hand-held device, a website, a message, a piece of software, a piece of data, – in short, anything that may have physical or virtual location. It is called a bigraph because there are *two* structures on these nodes; the *topographical* or *spatial* structure, represented by the way the nodes are nested, and the *connectivity* represented by the thin lines. These are totally independent: 'Where you are does not effect who you can talk to'.

This isn't just the space *within which* computing occurs; computing *consists in* recon£guration of this space. In this picture, a node could represent a *physical device* entering a domain of control, a *virus* doing the same thing, or a *number* about to be operated on. What happens in the machine is dictated by rules, which may vary from place to place, saying that a certain con£guration can change into another one.

As a simple example, suppose you are going to design a piece of the system to £t in the blank grey box. It may be a simple piece of message routing software; think of M being a message from B in MIT to C in the Cambridge Computer Lab. Here is one possible thing your little subsystem might do:



Your subsystem, when installed, will allow M to turn the key K; this will unlock the lock L, giving access to an administrative agent A that will in turn purvey M to C. (Of course, when you build your system, you also de£ne the recon£guring powers of keys and locks.)

This is a trivial example. The essential point is that substantial recon£gurations will be modelled in the same way (just as a tiny program is made of the same stuff as one with millions of lines). In this example the recon£guration is a *local* one – it involves only entities that are neighbours in the topography. But a recon£guration need not be local; it might consist of synchronized change involving widely separated parts, connected by the thin lines. This is the £ction of *action at a distance* which the web and the Internet allow us to entertain.

This points to a crucial feature of the models we are looking for. The stuff of the model must be able to represent both the £ction or *abstraction* which is entertained by a participant (say a human participant) as well as the *reality* by which this £ction is engineered. My colleague Peter Sewell points out that it is only through representing both the abstract and the concrete with the same modelling stuff that one can hope to verify or refute the claim that the concrete does indeed *realise* or *implement* the abstract. In fact, with a model of this kind, his group have recently veri£ed some protocols for communication between mobile entities that come close to those actually used in the Internet.

One further point before we leave this model. Any con£guration of the Internet and software running on it will be vast; coping with an abstract as well as a concrete representation is a necessary device to cope with this scale of things, but it's not the only one. We need to analyse the behaviour of *fragments* of a bigraph. For example, we need to be able to say that there will be, or won't be, any noticeable difference if one fragment is replaced by another with the same connectivity interface.

(For example, the fragment in the top right corner of the picture might represent the hardware and software of an Internet banking service; the bank might want to replace it by a system using not a single computer, but six of them linked by local area network.)

Ideally, we want to know if the environment can observe any difference in behaviour after such a replacement. To formulate this question accurately, and then to answer it, we have no choice but to refer to a *mathematical* notion of the *behaviour* of a fragment, de£ned by how it interacts at its interface – *not* by how it looks if you take it apart. Suitable behavioural theories have been developed by computer scientists over the last twenty years, often needing new mathematics and logics. There are signs that they can be extended to a geometric model such as I have shown you, and I regard it as a big part of the Grand Challenge to push this work forward.

Challenge for the Computer lab To summarise: We are faced with the extraordinary task of understanding what is probably the most complex artifact in human history, the global computer. Only by understanding it well can we expect it to serve us well. But at present it is developed by software methods which – however successful – rely upon *no previously developed science*. Compare this with any other standard engineering discipline; structural engineers, for example, can rely on material sciences and ultimately upon physics.

A CHALLENGE FOR TWO DECADES

Build a scienti£c model of the GLOBAL COMPUTER, in collaboration between engineers and theorists, providing one conceptual frame for

- *describing* the whole,
- *prescribing* the parts we build, both hardware and software,
- *designing* those parts in a way that is seen to match the prescription, and
- *modifying* parts whose prescription is modi£ed.

The global computer gives us both opportunity and tremendous incentive to develop a theory and a new style of software construction and adaptation which are aspects of one and the same scienti£c model. The theory will be *descriptive* of all that happens in the global computer, while the software will be *prescriptive*, determining the parts that we build; they will be expressed in the same terms. To get this to happen certainly involves half-baked prototypical models of the kind I showed you. But to succeed, it must involve experts in communications networks, programming languages, security, interface speci£cations, special hardware, machine assisted reasoning, and more. It will only come about through the normal process of scienti£c advance, and will probably take two decades. It's a much bigger aspiration than, for example, a £ve-year programme managed by a research funding council.

Who will do it? I fervently hope that the Computer Lab can play a large part. We have many of the kinds of people it needs. Also, thanks to the Gates Foundation, we have the opportunity to do our science and engineering under one splendid roof. Far from splitting our theories and practices apart, we can use this challenge to bring them close together than they ever have been.