What Computers Do: Model, Connect, Engage

Butler Lampson
Microsoft Research
June 2012

Abstract

Every 30 years there is a new wave of things that computers do. Around 1950 they began to *model* events in the world (simulation), and around 1980 to *connect* people (communication). Since 2010 they have begun to *engage* with the physical world in a non-trivial way (embodiment—giving them bodies). Today there are sensor networks like the Inrix traffic information system, robots like the Roomba vacuum cleaner, and cameras that can pick out faces and even smiles. But these are just the beginning. In a few years we will have cars that drive themselves, glasses that overlay the person you are looking at with their name and contact information, telepresence systems that make most business travel unnecessary, and other applications as yet unimagined.

Computer systems are built on the physical foundation of hardware (steadily improving according to Moore's law) and the intellectual foundations of algorithms, abstraction and probability. Good systems use a few basic methods: approximate, incrementally change, and divide and conquer. Latency, bandwidth, availability and complexity determine performance. In the future systems will deal with uncertainty much better than today, and many of them will be safety critical and hence much more dependable.

Extended Abstract

The first uses of computers, around 1950, were to *model* or simulate other things. Whether the target is a nuclear weapon or a payroll, the method is the same: build a computer system that behaves in some important ways like the target, observe the system, and infer something about the behavior of the target. The key idea is abstraction: there is an ideal system, often defined by a system of equations, which behaves like both the target system and the computer model. Modeling has been enormously successful; today it is used to understand, and often control, galaxies, proteins, inventories, airplanes in flight and many other systems, both physical and conceptual, and it has only begun to be exploited.

Models can be very simple or enormously complex, quite sketchy or very detailed, so they can be adapted to the available hardware capacity. Using early computers to *connect* people was either impossible or too expensive, compared to letters, telephones and meetings. But around 1980 Moore's law improvements in digital hardware made it economic to use computers for word processing, e-mail, mobile phones, the web, search, music, social networks, e-books, and video. Much of this communication is real time, but even more involves stored information, often many petabytes of it.

So modeling and connection are old stories—there must be little more to do. Not so. Both the physical and the conceptual worlds are enormously complex, and there are great opportunities to model them more accurately: chemical reactions, airplane wings, disposable diapers, economies, and social networks are still far from being well understood. Telepresence is still much worse than

face-to-face meetings between people, real time translation of spoken language is primitive, and the machine can seldom understand what the user doing a search is actually looking for. So there's lots of opportunity for innovations in modeling and connection, especially in education, by providing teachers with power tools.

Nonetheless, I think that the most exciting applications of computing in the next 30 years will *engage* with the physical world in a non-trivial way. Put another way, computers will become *embodied*. Today this is in its infancy, with surgical robots and airplanes that are operated remotely by people, autonomous vacuum cleaners, adaptive cruise control for cars, and a cell phone-based sensor networks for traffic data. In a few years we will have cars that drive themselves, prosthetic eyes and ears, health sensors in our homes and bodies, and effective automated personal assistants. In addition to saving many lives, these will have vast economic consequences. Autonomous cars alone will make the existing road system much more productive, as well as freeing drivers to do something more useful or pleasant, and using less fuel. I have a very bad memory for people's names and faces, so my own dream (easier than a car) is a tiny camera I can clip to my shirt that will whisper in my ear, "That's John Smith, you met him in Los Angeles last year."

What is it that determines when a new application of computing is feasible? Usually it's improvements in the underlying hardware, driven by Moore's law (2 × gain / 18 months). Today's what-you-see-is-what-you-get word processors were not possible in the 1960s, because the machines were too slow and expensive. The first machine that was recognizably a modern PC was the Xerox Alto in 1973, and it could support a decent word processor or spreadsheet, but it was much too small and slow to handle photographs or video, or to store music or books. Engagement needs vision, speech recognition, world modeling, planning, processing of large scale data, and many other things that are just beginning to become possible at reasonable cost. It's not clear how to compare the capacity of a human brain with that of a computer, but the brain's 10^{15} synapses (connections) and cycle time of 5 ms yield 2×10^{17} synapse events/sec, compared to 10^{12} bit events/sec for a 2 GHz, 8 core, 64 bit processor. It will take another 27 years of Moore's law to make these numbers equal, but a mouse has only 10^{12} synapses, so we'll have a digital mouse in 12 years (but it will draw more power than a real mouse).

Hardware is not the whole story, of course. It takes software to make a computer do anything, and the intellectual foundations of software are *algorithms* (for making each machine cycle do more useful work) and *abstraction* (for mastering complexity). We measure a computer or communication system externally by its *bandwidth* (jobs done per unit time), *latency* (start to finish time for one job) and *availability* (probability that a job gets done on time). Internally we measure the *complexity*, albeit much less precisely; it has something to do with how many component parts there are, how many and how complex are the connections between parts, and how well we can organize groups of parts into a single part with only a few external connections.

There are many methods for building systems, but most of them fit comfortably under one of three headings: Approximate, Increment, and Divide and conquer—AID for short.

- An approximate result is usually a good first step that's easy to take, and often suffices. Even more important, there are many systems in which there is no right answer, or in which timeliness and agility are more important than correctness: internet packet delivery, search engines, social networks, even retail web sites. These systems are fundamentally different from the flight control, accounting, word processing and email systems that are the traditional bread and butter of computing.

- Incrementally adjusting the state as conditions change, rather than recomputing it from scratch, is the best way to speed up a system, short of a better algorithm. Caches in their many forms, copy on write, load balancing, dynamic scale out, and just in time compilation are a few examples. In development, it's best to incrementally change and test a functioning system. Device drivers, apps, browser plugins and JavaScript incrementally extend a platform, and plug and play and hot swapping extend the hardware.
- Divide and conquer is the best single rule: break a big problem down into smaller pieces. Recursion, path names such as file or DNS names, redo logs for failure recovery, transactions, striping and partitioning, and replication are examples. Modern systems are structured hierarchically, and they are built out of big components such as an operating system, database, a browser or a vision system such as Kinect.

For engagement, algorithms and abstraction are not enough. Probability is also essential, since the machine's model of the physical world is necessarily uncertain. We are just beginning to learn how to write programs that can handle uncertainty. They use the techniques of statistics, Bayesian inference and machine learning to combine models of the connections among random variables, both observable and hidden, with observed data to learn parameters of the models and then to infer hidden variables such as the location of vehicles on a road from observations such as the image data from a camera.

Some applications of engagement are safety critical, such as driving a car or performing surgery, and these need to be much more dependable than typical computer systems. There are methods for building dependable systems: writing careful specifications of their desired behavior, giving more or less formal proofs that their code actually implements the specs, and using replicated state machines to ensure that the system will work even when some of its components fail. Today these methods only work for fairly simple systems. There's much to be learned about how to scale them up, and also about how to design systems so that the safety critical part is small.

Engagement can be very valuable to users, and when it is they will put up with a lot of hassle to get the value; consider an artificial eye for a blind person, for example. But other applications, although useful, have only modest value, such as a system that tells you which of your friends are nearby. But other applications, such as a system that tells you which of your friends are nearby, are examples of ubiquitous computing that although useful, have only modest value. These systems have to be very well engineered, so that the hassle of using them is less than their modest value. Many such systems have failed because they didn't meet this requirement.