Grand Challenges Conference Application

Macrosystems

Jim Morris
School of Computer Science, Carnegie Mellon

New Problems

We've gotten good enough at engineering computer systems that the practice seems "incremental" now. Building an operating system used to be a research challenge; now it's not. However, our ability to construct large systems has outstripped our ability to understand and control them. Exhibit A is the Internet. A set of excellent engineering decisions creating the basic protocols, connection methods, and user interfaces created an object that grew exponentially. The problem is that the Internet is now not under anyone's control. Its emergent properties haven't been designed in a conventional engineering sense; they have grown like a city. The Internet is the canonical example, but there are many others. The testing and integration of any large-scale software project is the most immediate. The management of telecommunications networks without centralized network operations centers is an emerging art needing some scientific support. Storage deployed is approximately doubling every year; communications is making it all reachable. Data/information/knowledge problems are exploding in scale. In fact, even the software running on a PC seems out of anyone's control because it has come from so many different sources without central design control. Such systems are so big and chaotic that we can't comprehend them in the conventional terms of designed artifacts.

A New Approach

Rather than try to regain control, I suggest we should try to get over losing it. The standard approach of computer systems research—build something and study it—is no longer applicable. We need a new field called *macrosystems*, named in the same spirit as macroeconomics. The attitudes and methods of this field will come from economics and other social sciences like urban planning. Those fields ask questions like "Will lowering interest rates revive consumer spending?" and "Will building highways around the city center be good for the community?" Such questions require deep understanding of the dynamics of the systems they are attempting to influence rather than exquisite control over the implementation process.

We need new intellectual tools to understand our increasingly complex infrastructure. To understand and live with macrosystems we need a more general form of computer science. We need some useful laws that guide our intuitions and actions. Could there be laws of computer-based systems as useful as those of thermodynamics? What general statements can be made about systems that one doesn't have the source for? So far, our stock of laws is pretty meager: Murphy's Law, Moore's Law, Metcalfe's Law, a few others.

A Practice in Search of a Law

An example of a law yet to be formulated and explained is represented by a practice that everyone knows: If your system is malfunctioning, reboot it or unplug it. Every tech support person practices this. There has been some good engineering analysis of the practice by Trivedi and Kintala[1].

Notice that this is an idea that didn't exist before digital systems. When a Model-T Ford was malfunctioning the list of remedies did not contain "Turn it off and on again." More likely were suggestions like, "Make sure it has gas," "Check for worn spark-plugs," etc. Digital devices and the

[1] http://www.software-rejuvenation.com/

way they are designed are different somehow. Dan Siewiorek told me of someone having to disconnect her modern car's battery after it shut down after a sharp turn.

The law might be stated as "Any digital system that is shut down and restarted will be restored to its original state, expect for what's changed on the disk." This doesn't sound dignified enough to be a law, but it's the best I can do today.

Andreas Nowatzyk pointed out that it takes a lot of engineering to get a complex IC to initialize and start. In other words, the restartability of digital systems doesn't come for free but is achieved through engineering discipline. At the software level, this law seems to depend on how we test systems: by repeatedly restarting them from their initial state. The longer a system is used, the further it migrates from the region where its behavior has been well-tested.

So this law depends upon an artificial fact about how systems are built. It is not a natural law like Boyle's law. The people who build systems might not follow the design rules. So the law is more like a covenant between implementers and users that says, "Maybe the system will not perform properly in all particulars, but you can always depend upon it functioning if you restart it." In other words, the system we are trying to formulate rules for includes us! The same objection could be made to economics and other social sciences, as explained so well by Herb Simon[2].

However, design practices might not be so voluntary. Perhaps there are evolutionary market pressures that eliminate systems that don't obey the rules. There are some biological analogs to restarting (sleep, seizures, and death) that suggest the restart law might evolve through natural selection. Even the dubious "ontogeny recapitulates phylogeny" seems to have a digital analog: When rebooting, a system tends to bring up its components in the order they were created. Just as an early stage fetus looks like a reptile, Windows[tm] starts out looking like DOS.

New People

Who is going to take this new approach to systems research? It might be someone who gets a BS in Economics and a Ph.D. in Computer Science. Neither the CS department nor the Economics department will like this work initially. The study of macrosystems will seem unsatisfying to classical computer scientists who expect results to be implemented systems or proven theorems. Economists, I'm told are even less tolerant of work that doesn't involve optimizing Lagrangians.

In the meantime, several researchers contribute to the beginnings of this field. Most important are measurements and benchmarks of real systems; recent work by Jim Gray[3], Dave Patterson[4], and Phil Koopman[5] come to mind. David Garlan[6] and Mary Shaw's studies of software architecture also point the way.

BIO:

Dr. James H. Morris is dean of the School of Computer at Carnegie Mellon University. He held the Herbert A. Simon Professor of Human Computer Interaction from 1997 to 2000. He is a native of Pittsburgh and received a Bachelor's degree from Carnegie Mellon, an M.S. in Management from MIT and Ph.D. in Computer Science from MIT. He taught at the University of California at Berkeley where

---

[2] Herbert A. Simon, The Sciences of the Artificial, MIT Press, 1996

[3] http://www.hdcc.cs.cmu.edu/may01/Jim_Gray.ppt

[4] http://www.cs.berkeley.edu/~pattrsn/talks/HP.ppt

[5] http://www-2.cs.cmu.edu/~koopman/ballista/index.html

[6] http://www-2.cs.cmu.edu/afs/cs.cmu.edu/project/able/www/paper_abstracts/archmismatch-icse17.html

he developed some important underlying principles of programming languages: inter-module protection and lazy evaluation. He was a co-discoverer of the Knuth-Morris-Pratt string searching algorithm. For ten years he worked the Xerox Palo Alto Research Center where he was part of the team that developed the Alto System, a precursor to today's personal computers. From 1983 to 1988 he directed the Information Technology Center at Carnegie Mellon, a joint project with IBM which developed a prototype university computing system, Andrew. He has been the principal investigator of several NSF and DARPA projects aimed at computer-mediated communication. He is a founder of the MAYA Design Group, a consulting firm specializing in interactive product design.