Biography

W. Keith Edwards is a Senior Member of Research Staff in the Computer Science Lab at the Palo Alto Research Center (which until recently was the Xerox Palo Alto Research Center). His research interests lie at the intersection of distributed systems and human-computer interaction. More specifically, he is interested in understanding the ways in which aspects of distributed systems can "show through" to manifest themselves as part of user experience and, in turn, the implications of that for the design of distributed systems.

He has been involved with a number of research projects that explore this area of interest, ranging from infrastructures for peer-to-peer ubiquitous computing, document management systems, computer-augmented whiteboards, auditory user interfaces, flexible version control systems, and replicated, weakly-consistent distributed databases. He holds a Ph.D. in computer science from Georgia Tech, where he did his dissertation work on one of the first infrastructures designed to support context-aware computing applications.

He is the author of over thirty refereed publications, as well as two books on Sun's Jini distributed computing technology and an upcoming book on the architectures of window systems and graphical toolkits. He holds eight U.S. patents.

Bringing Network Effects to Computing Systems

W. Keith Edwards
Palo Alto Research Center (PARC)
kedwards@parc.xerox.com

For the most part, current computational devices and services, although network-capable, are locked into isolated islands of interoperability: systems that were explicitly built to work with one another can interoperate, while others cannot. We pay a price for this lack of interoperability. This price can be framed in terms of the huge cost of software and equipment made obsolete before their time, developer effort spent building "bridges" between otherwise inoperable technologies, and the opportunity costs of users being unable to use the systems around them at the time they need them.

This situation stands in contrast to what we experience daily with the telephone network. My rotary dial phone, circa 1965, is able to connect to devices such as cell phones, over protocols such as GSM, neither of which existed when my telephone was built. Further, it is able to do so without upgrades or driver installation. Our computing systems certainly do not have this property. Network interoperability in the domains of desktop computing, handheld computing, enterprise software services, and others, is often fraught with driver installation, software upgrades, recoding, and hardware installation.

I propose that we in the computing systems community direct ourselves to the challenge of enabling *network effects*, like those seen in the telephone network, for the arbitrary computational devices, services, and applications that play increasing roles in our lives. A network effect is a property of a distributed system in which the overall utility (or *value*) of any part of the system increases as the overall system grows. In the telephone network, for example, my phone becomes more capable, in a sense, as more users come online—it can reach more and more people without any change to the phone itself.

Network effects for computational devices and services would allow similar exponential utility curves: quite simply, the PDA in my hand should be able to do *more things*, as more devices and services come online around it. For this vision to become reality, the calculus of effort must be turned on its head: rather than requiring every existing node on the network to be upgraded to accommodate the newcomer—a task that will become only less and less practical as the numbers and types of networked devices explodes—the addition of the newcomer itself should carry with it the knowledge needed by existing nodes to use it.

This reversal necessitates a change in architectural thinking, but the benefits of delivering such an architecture are compelling. Most directly, network effects enable a new economy of compatible technologies, each of which adds value to the next. These economic benefits have been demonstrated by network effects in areas such as European mobile telephony, the i-Mode service in Japan, and the World Wide Web. Second, by enabling more fluid interoperation we have the potential to greatly increase productivity: users and developers can *serendipitously* recombine the resources around them, without advance planning or coding. Finally, we can allow users to repurpose technologies to uses their designers did not foresee, and did not even plan for. Resources on

the network can be composed into virtually arbitrary functions, limited only by the *semantics* of those resources, rather than the particulars of how they communicate.

How would one begin to enable such network effects? Network effects require the ability to communicate and, fundamentally, all communication between two parties presumes some shared knowledge, agreed to in advance by both. For my laptop to use my cell phone as a modem, both must have been explicitly programmed to speak a common protocol, such as RFCOMM or IRCOMM. The web—inarguably the most powerful example of true network effects for computing systems today—requires agreement between the web browser and the web server on a handful of protocols (HTTP, FTP) and data types (HTML, XML) known *a priori* to both parties. Web service frameworks require that the client of a service know that service's interface, typically as expressed in a description language such as WSDL. There are many other examples.

The challenge for the systems community, then, is to explore *new levels* of shared knowledge among computational entities—these are new *architectures of agreement* among the parties that communicate. These architectures should allow rich, serendipitous interactions among devices and services without the "lowest common denominator" approach of settling on an arbitrary set of protocols assumed to be known in advance to all parties. These architectures should be able to accommodate user-driven interactions (such as in the web), as well as program-to-program interactions (such as those envisioned by web services).

A number of such new architectures of agreement have been explored by the research community already. The semantic web is one example—the semantic web proposes a shared ontology that represents some portion of human knowledge, agreed upon and understood by all parties in a communication. This approach, however, has weaknesses: first, it requires the creation of the ontology itself, and second, it requires that the parties be coded to "understand" a sizeable body of knowledge, albeit encoded in an machine-parsable format. As another example, mobile code-based approaches such as Jini offer the ability for entities on the network to dynamically extend the behavior of their peers. These approaches also have their weaknesses, namely the requirement for agreement on a shared executable format, and the security problems that that brings.

I believe that these, as well as other, approaches must be explored. In particular, I contend that architectures of agreement that require less effort on the part of developers (in other words, *narrower* interfaces) will be required if true network effects are to occur—the burden on builders of devices and services should be as small as possible. I further contend that approaches that require prior agreement on a range of domain-specific interfaces are doomed to failure (this is largely the situation now, where my computer is coded to speak to printers via IPP, file servers via CIFS, and so on; speaking to other sorts of entities requires explicit coding to adapt the system to their particular interfaces).

In summary, the challenge of bringing network effects to arbitrary devices and services requires a fundamental rethinking of the way we architect our systems for communication. I believe the potential payoffs of solving such a challenge greatly outweigh the costs and risks of addressing it.