Shedding Light on Embedded Systems

Anne C. Lear

From our toasters to our cars' fuel injection systems to our mobile phones, embedded systems have begun to work their way into a wide range of our daily functions. More accurately, engineers are implanting these systems—which range from a small chip controlled by a few thousand lines of code to mini-PCs controlled by scaled-down operating systems—into a growing number of hosts. Limited physical and memory space demands small hardware and software. The lack of direct user involvement or supervision requires highly reliable software, which must work in real time because embedded systems respond to and control real-world events.

Within these constraints, embedded systems have enabled industrial automation, mobile computing, "smarter" appliances, telecommunications, medical equipment, and countless other devices and services. In his keynote address at the 1998 Embedded Systems Conference in San Jose, Jerry Fiddler of Wind River Systems noted that in the 1970s, embedded systems did little more than play Pong; now they control much of the world's infrastructure. What happened in between? Embedded systems are not new; engineers have been building computing power into electronics for more than two decades. What is new is that technologies such as distributed computing, intelligent systems, mobile computing, networking, and the Internet have begun to expand both the reach and the power of embedded systems.

Fiddler declared that embedded systems have barely begun to approach their potential, as these enabling technologies and trends are only now maturing and converging. The "consumerization of computing" is bringing information access to a growing array of non-desktop devices, from palm PCs to cellular phones to pagers. Smaller, faster, cheaper microprocessors make it possible to embed more power into more things. The Web and Internet provide a global standards-based communications network through which these things can link to each other, to databases, and to control centers, enabling remote monitoring, control, and configuration.

Be Invisible, and Do It Well

To provide more functions to more things, embedded systems need more software. Until recently, "engineering" embedded systems has typically meant creating the hardware, then hacking some quick code that told the hardware what to do and when. In 1982, embedded systems were defined as "computer systems that cannot be programmed by the user because they are preprogrammed for a specific task and are buried within the equipment they serve" (*Encyclopedia of Electronics and Computers*, McGraw Hill, p. 330). At that time, embedded systems were typically based on eight-bit processors and were programmed in assembly language to define a certain, immutable set of functions—simple programs that brought some intelligence to mechanical things. Used primarily for industrial automation and military systems, these systems had to be highly reliable and to operate under harsh conditions, hence the need to "bury" them in the equipment they served.

Today, many ESs use a 32- or 64-bit processor. Some, such as those used in PDAs and cellular phones, may have graphical user interfaces, and some may be programmed in a higher-level language such as C or C++. In many cases, they don't just run equipment but often "report to" other systems that monitor their operation and may also monitor such factors as environmental variables, inventory, or time and date of certain events. This requires that the ES gather and pass along data, perhaps even partially process it. Some ESs use embedded PCs for such tasks. ZDNet's Webopaedia allows for such a stretch by defining an ES as "a specialized computer system that is part of a larger system or machine" (www.zdwebopedia.com/TERM/e/embedded_system.html, 1998).

Smarter software means smarter stuff

As hardware evolves and embedded systems get bigger, software must grow as well, which makes the choice of programming languages, platforms, and techniques more significant. Most embedded systems have an operating system that forms the core

EDITOR: Anne C. Lear • Staff Editor • alear@computer.org

0740-7459/99/$10.00 © 1999

(and in some cases the whole) of its software. Unlike mainstream desktop computing, the embedded market currently supports many different operating systems. Some chips come with proprietary OSs, other systems use commercial ones such as QNX, Wind River's VxWorks, LynxOS, Integrated Systems' pSOSystem, Enea's OSE, and Windows CE. Cygnus recently released eCos, an open-source, royalty-free OS that supports multiple platforms and includes source code for all kernel components, HAL layers, C runtime and math libraries, and drivers.

Different OSs support different microprocessors and tailor main and secondary memory usage, real-time capability, network support, and other features to specific tasks or domain areas. Furthermore, some ESs require very streamlined operating kernels, whereas others require a full-scale OS that can handle multiple, complex tasks. These factors make optimization essential and preclude a one-size-fits-all solution. Nonetheless, Java and Windows CE are emerging as "standard" runtime platforms for ES development, in part because they promise to simplify programming and facilitate interoperability. Many developers welcome the compatibility with other Windows platforms that CE offers, but others feel it is too unwieldy for smaller, more nimble ESs. Java offers promises and compromises as well, in particular the ability to program for multiple platforms. But as Sun battles with an industry consortium led by Hewlett Packard over a specification for real-time extensions to Java, some developers remain cautious about getting locked into one or another version of Java.

For smarter software, use smarter development methods

As projects become more complex, programming methods must improve. Embedded systems have always needed to be highly reliable—"you can't just reboot them," Fiddler wryly noted—but larger, complex, interconnected systems have more variables to control and thus anticipate when designing and coding. The software world has long discussed the importance of process and project management, and how to apply software engineering principles to help meet tight time and financial constraints. Like Internet application development, however, ES development has evolved in isolation from mainstream software applications development; moreover, the hitherto light software requirements of ESs made "process" seem not worth the time and effort. But with average ES code size growing from a few thousand lines of code in the 1980s to perhaps a million LOC or more today, hacking is no longer efficient or reliable.

At the 1998 Embedded Systems Conference, several papers and tutorials addressed the growing importance of defining and implementing software process improvement methods for ES development. Greg Bergsma, senior technology analyst at QNX, pointed out that few process improvement methods exist for embedded system development teams, which face increasing project size and complexity ("Solving the Big Problems: Architecting a Development Framework for Large-Team Software Projects," *Proc. Embedded Systems Conf.*, Miller Freeman, San Francisco, 1998, Vol. 2, pp. 1-8). Hans Brands and Anton Evers detailed Philips' efforts to apply the Capability Maturity Model ("Improving the Software Process for Embedded Systems," *Proc. Embedded Systems Conf.*, Vol. 2, pp. 183-199). Steven Stolper, a former technical staff member at the Jet Propulsion Laboratory, described how his team developed the flight software architecture for the Mars Pathfinder Spacecraft ("Designs That Fly!," *Proc. Embedded Systems Conf.*, Vol. 3, pp. 233-247). Faced with very high performance requirements, tight time and cost constraints, and no precedents, the team came up with a compromise between ad hoc and detailed process-based methods. Stolper emphasized that the process-based methods enabled the team to meet its budget and time constraints.

As embedded systems become more complex, programming methods must improve.

Getting them wired

Not only are more things performing more complex functions, they communicate with each other through infrastructures as varied as wired and wireless buses, LANs, WANs, and the Internet. Indeed, the Internet offers a way to not only negotiate but also capitalize on the diversity and flexibility of embedded systems. One possible means: embedding Web servers into devices.

Further Reading

The IEEE Computer Society Technical Committee on Real-Time Systems' Real-Time Research Repository includes articles, research groups, conference and symposia archives, books, magazines, and tools. **http://cs-www.bu.edu/pub/ieee-rts/**

Embedded.com links to *Embedded Systems Programming*, as well as product news and directories, conference information, and other Internet resources. **www.embedded.com**

Real-Time Engineering targets software and systems engineers developing real-time software for embedded systems. **http://www.realtime-engineering.com/**

Byte magazine explored the history and possibilities of embedded systems in a June 1997 article by Bob Margolin, "Smarter Stuff." **www.byte.com/art/ 9706/sec6/art2.htm**

While PCs accounted for 96 percent of Internet access devices sold in 1997, IDC Research forecasts that by 2002 almost half of all Internet access devices sold will be non-PC devices (www.idcresearch.com/F/Ei/gens19.htm). These will include consumer items such as Web-enabled PDAs, telephones, video game consoles, and set-top boxes. But they will also include nonconsumer embedded systems that control telecommunications, power grids, traffic signals, and other infrastructure services; monitor environmental conditions or inventory; and so forth.

At the 1998 Embedded Systems Conference, Steve Wingard, a senior software engineer at Spyglass, discussed how to "embed the Web" by placing tiny (100K) HTTP servers into systems for remote monitoring, configuration, diagnosing, and updating ("Embedding HTTP Functionality for Web-Based Configuration and Management of Devices," *Proc. Embedded Systems Conf.*, Vol. 3, pp. 347-357). The Web provides a standards-based environment, nearly universal access, and also a familiar interface—a browser, home pages, hyperlinks, forms—for users to interact with and manage connected devices. Unlike the mainstream Web, the embedded Web server would in most cases be secondary to the device's purpose, have a low transaction frequency, and not need a database system for content. These place different requirements on server development for ESs, which Wingard discussed in detail.

Y2K: Defusing the Landmines

If only more embedded systems were already so connected and managed. By nature hidden, taken for granted, intended to perform silently and reliably, these systems have suddenly become a liability as systems administrators at companies, government agencies, and utility companies try to hunt down every single computer system that might be affected by the Year 2000 Problem. If Y2K is a giant ticking time bomb, embedded systems are the landmines. The systems that permitted automation of key operations may not maintain such operations after the new year, making them, for the next year or so, a headache for those who have to find, evaluate, and possibly fix them.

Many experts say only a small percentage of embedded systems will fail on 1 January 2000; the Gartner Group told the US Senate on 7 Oct. 1998 that "only 1 in 100,000 freestanding microcontroller chips are likely to fail" due to Y2K (*TechWeek*, 16 Nov. 1998). But the *billions* of chips currently operating in embedded systems amounts to a huge number of widely distributed failures. Dataquest, a market research firm, estimates that 4.6 billion embedded microcomponents were sold in 1997, not to mention the several billions sold in years past that still operate. Where are they? Everywhere, doing everything from making sure your breadmaker gives the dough enough time to rise, to activating hydraulic systems that control flaps and rudders on airplanes, to letting people know when components in their cars or at electrical generating stations or factories need service. Which of these will fail next year? Perhaps all, perhaps none, depending on which use date calculations and data—but few people know exactly which embedded systems do and which don't, so they have to diagnose them—which means finding them.

In New Jersey, for example, it took Public Service Electric & Gas Co. a year to find all the embedded processors in its 250 power plants, substations, and other locations (A. Pollack, "Chips Are Hidden in Washing Machines, Microwaves and Even Reservoirs," *The New York Times*, 4 Jan. 1999). These and similar examples have fueled fears of utility services shutting down as 1999 rolls over into 2000, but most US utility companies claim to be pretty far along in finding and fixing non-Y2K-compliant ESs, as do elevator companies, building administrators, and others who could face serious liabilities if their systems fail.

But what if some of the 15 or so processors in your car fail? What about the 1,000 processors in Boeing's late-model 777 jets? Who will fix the devices already known to have problems, let alone all those that might?

From Dispersed Systems to Coherent Industry: The Future

While some have painted ESs as the gremlins of the Y2K crisis, the publicity has not been entirely negative. Just knowing how much we have come to depend on ESs has raised awareness of both their current and potential uses. It also makes clear how

valuable networked management of embedded systems could and will be. To see whether their embedded systems might be affected by Y2K, administrators or technicians could examine, diagnose, and fix systems as needed from a central location. They could also remotely upgrade system software or adjust functions based on changing system or device needs.

Embedded computing has emerged from within dozens of industries, from consumer electronics to military suppliers, each creating its own, highly specialized applications. As software becomes a larger part of the typical embedded system, good software development practices become essential to creating robust, reliable systems. The next generation of ESs will, according to Jerry Fiddler, be scalable, communications-centric, based on virtual platforms, multimedia-enabled, application-specific, and programmer-friendly. These traits far exceed early instances and definitions of embedded systems, and increase demands on their developers. More important, and key to the future of embedded systems, they depend completely on good software. ❖

Netscape Starts Over

Netscape released its Communicator source code to the open-source community last March. Now the company has decided to start from scratch with a new browser engine that it hopes will make its browser small, fast, and modular. Originally called NGLayout, now officially named Gecko, it fits, compressed, on a single diskette. Netscape's ultimate goal is to fully comply with Level 1 standards, including HTML 4.0, XML, CSS, and DOM. Web designers currently must follow special design criteria for each Web browser as well as older versions of the same browser. Gecko may compel the other major browsers, particularly Microsoft's Internet Explorer, to also become 100 percent standards-compliant.

Mozilla, the generic term referring to Web browsers derived from the Communicator source code, is really a family of browsers. Brendan Eich, Mozilla technical director, said that the decision of Mozilla browser project members to use XPFE and NGLayout (now Gecko) followed from "the judgment of the module owners; the fervent wishes of Web content authors; and my personal judgment as mozilla.org technical bigshot. Not only will many bugs and lacks be resolved, the road ahead can better support a full mail client, user-scriptable hooks, and many other clients and proxy-like modules" (*WebTools*, 23 Dec. 1998; http://www.webtools.com/story/TLS19981223S0001/). ❖

In Brief

Open-source petition. A group of software professionals launched a petition that asks the US government to evaluate open-source software, in addition to commercial products, when it considers upgrades. Written by Professor Clay Shirky of Hunter College and sponsored by the Open Source Initiative and O'Reilly and Associates, the petition asks the Federal Technology Service, a branch of the General Services Administration, to recognize that open-source software has in many cases met or exceeded commercial standards.

According to the petition, open-source software use would lower the cost of IT procurement to taxpayers, reduce the federal government's dependency on individual vendors, and secure future improvements to such software without further cost. The petition doesn't ask the federal government to use open-source software, only to evaluate it based on the same criteria it currently uses to evaluate commercial software. Over 4,000 people have added their names to the petition, which can be found at http://www.wethepeople.com/etp/affiliates/national/fullview.cfm?ETPID=0&PETID=74386&ETPDIR=affiliates/national.

Encryption export controls eased. The US Department of Commerce released a press statement on 30 Dec. 1998 stating that it will publish new regulations on the export of encryption products, eliminating many of the previous restrictions. The amendments, which implement policy changes announced in September by Vice President Al Gore, streamline reporting requirements for US firms, offer new allowances for US encryption manufacturers to share their source code with their own foreign subsidiaries, and virtually eliminate restrictions on selling powerful data scrambling products to subsidiaries of US corporations.

The new rules also ease controls on sales to health and medical organizations and insurance companies headquartered in 46 countries. For additional information and a list of the eligible countries, visit the US Department of Commerce Web site at http://www.doc.gov/.

Job outlook good. According to *EE Times* (30 Dec. 1998), 1999 will be a strong year for the technical job market, with most of the activity in the software field. The National Software Alliance has said that the supply of skilled software workers in the US is shrinking and that 59 percent of US technology firms report that they do not have enough software workers.

US companies, organizations, and government agencies will need 137,000 new programmers each year through 2006 to meet protected industry growth (*Computer*, Jan. 1999). Due to this increased demand, recruiters predict increased salaries and benefits, including signing bonuses and stock options, for qualified workers.