A Software Architecture for Open Service Gateways

Open-standard residential gateways promise to bring a wide variety of managed services to the home—without introducing new wires to the infrastructure.

Li Gong
Sun Microsystems

Two industrial trends are changing our day-to-day lives. One is the introduction of broadband service to the home, where it promises to deliver complex new services. The other is the increasing resourcefulness, connectedness, and intelligence of home appliances. A residential gateway is a networking device that creates a bridge between the broadband network and the in-home network, and between different networking technologies within the home.

This article gives an overview of this emerging marketplace, and introduces the work ongoing at the Open Service Gateway initiative. OSGi is defining a set of open-standard software application interfaces (APIs) for building open-service gateways, including residential gateways.

Home, Dot-Com Home

Surprising innovations often result from human need. Circa 1991, for example, members of the local coffee club at the University of Cambridge computer laboratory found it increasingly inconvenient to climb several flights of stairs, pushing open heavy doors and squeezing through narrow corridors, only to find an empty coffeepot at journey's end. Sufficiently frustrated, they rigged up a video frame-grabber to a camera focused on the coffee machine, wrote a client program and a server program, and started using the Web to remotely check when a new pot of coffee was needed. The XCoffee system, still viewable at http://www.cl.cam.ac.uk/coffee/coffee.html, was a good indicator of things to come.

Indeed, the Internet has grown tremendously since the early 1990s, and its influence is felt in almost every aspect of our lives. It has changed how we communicate, conduct commerce, and obtain entertainment. Because of the Internet's ubiquity and the vast resources it makes available, a computer's value increases dramatically when it is connected to the Internet and can obtain services from afar. In fact, many users now own a general-purpose computer solely to access the Internet. Moreover, thanks to more pow-

 1089-7801/00/$10.00

erful and less expensive hardware, many smart and connected devices, such as mobile phones, pagers, and PDAs, have emerged on the market. This trend has led industry analysts to dub this the beginning of the "post-PC" era.

With the availability of broadband services such as digital subscriber line (DSL) and cable modem to homes, growing household ownership of multiple PCs, and digitization of all forms of home entertainment, an in-home network seems the next logical step. Several device control networks have already been developed for use inside the home. The timing is perfect for integrating outside networks with those inside the home. (See the sidebar, "The Case for Home Networking," for more.)

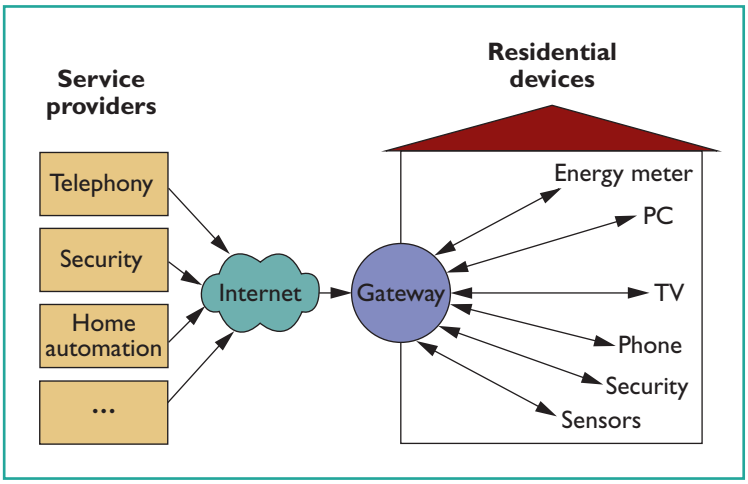

Figure 1. Residential gateway. The gateway connects the home network to the Internet.

Residential Gateway

Traditional service providers such as utility, telephone, and cable TV companies all have dedicated wires into the home. This configuration will be unmanageably complex, however, in the networked home of the future; its diversified portfolio of services, including home security, health monitoring, telephony, and audio/video media, might each employ a different communication technology. To mitigate this potentially chaotic situation, developers have proposed the residential gateway, a centralized device that interfaces between the external Internet and the internal device and appliance networks. Some common hardware components of a residential gateway include processor, persistent storage (disk or flash RAM), networking (such as TCP/IP), and device interfaces (serial or parallel port, for instance), which are typically powered by an operating system (OS) or real-time operating system (RTOS).

The familiar cable set-top box can be transformed into a residential gateway by augmenting it with more "smarts." General Instruments' DCT-5000+ set-top, for example, is equipped with a 300-plus-MHz MIPS processor, 14-Mbyte memory, integrated cable modem, Ethernet, USB, and IEEE 1394 interfaces. Other companies have designed residential gateways from the ground up. Ericsson's E-box is an example; it features a 100-MHz 486 CPU, 32-Mbyte memory, 24-Mbyte flash memory, 10BaseT Ethernet, and serial port.

The residential gateway, pictured in Figure 1, can participate in a wide range of home-based applications, including emergency response, energy management, and distance learning. For example, a home theater system might send a signal to lower the window blinds when a movie starts to play. A gateway that can authenticate users over a cell phone can also let them open a door remotely when a family member forgets the key. The residential gateway thus serves as a staging place for merchants and service providers to manage services that can leverage one service to benefit another.

The familiar cable TV set-top box can be transformed into a residential gateway.

Although the residential gateway may be easy to understand conceptually, building one is quite complicated. One reason for this is the difficulty in developing service software and provisioning services to the home. For example, setting up and configuring the device must be simple and automatic, and service providers should be able to remotely manage complicated appliances and services. Many people cannot even program their VCRs, so for them, managing smart appliances must be as simple as turning on the radio. Moreover, a service like recording TV programs often depends on the presence of another service such as an electronic programming guide (EPG). Therefore, to enable the recording service, the residential gateway must be able to automatically detect the need for the underlying EPG service, discover its presence or absence, and ensure its availability.

Another difficulty in developing a home gateway is the number of competing networking technologies and standards inside the home. The sidebar, "Competing Standards for Networking the Home," provides an overview of the many exist-

The Case for Home Networking

The Cahners In-Stat research firm reported that the number of cable modem subscribers in North America reached 1.8 million in 1999, and the number of broadband cable data subscribers worldwide is expected to be 9.5 million by 2002. In fact, today more than 110 million homes in North America are within earshot of a broadband coaxial cable line, and 77 million homes have cable TV services. The research firm also predicts that the average number of connected nodes per home network will increase from 2.9 in 1999 to 5.0 by 2003.

Thus, the in-home network market potential is huge. Cahners In-Stat estimates that the revenue from cable broadband services will increase from the current US$1 billion to US$4 billion by 2002. Allied Business Intelligence projects that the home-networking equipment market alone will reach US$2.4 billion by 2005, while Parks Associates estimates that the total value of the end-user market will be more than US$4.5 billion by 2004.

Given the market size, it is not surprising that many companies are investing millions of dollars into developing technologies, creating standards, and manufacturing novel products. Enthusiasts predict fascinating applications, such as refrigerators that order groceries automatically, TVs that deliver programs based on personal interests, and climate control systems that are optimally tuned for comfort and energy conservation.

Numerous field trials are also under way. For example, TiVo service allows you to rate TV programs and record your favorites automatically through a recorder connected to the service provider over the phone line. With Echelon's LonWorks (http://demo.echelon.com/) control network, you can use the Internet to control your home's lights, window blinds, and thermostats. In Copenhagen, a trial began in September 2000 in which each household gets a futuristic Electrolux Screenfridge, a Tele Denmark 2-megabit-per-second ADSL high-speed data connection, and an Ericsson WAP phone. For several months, the families will sample an array of applications (Internet e-mail, intra-family messaging, Web access, online grocery shopping, and weather reporting) targeted to improving the quality of life at home.

ing standards in this space. The prevailing view is that all or most of them will coexist for the foreseeable future, which means the gateway must interact with them all in a meaningful way.

Open Service Gateway Initiative

The Open Service Gateway initiative (OSGi, http://www.osgi.org/) is an independent, nonprofit industry group working to define and promote an open standard for connecting the coming generation of smart consumer and business appliances with Internet-based services. The need for standardization in this space is obvious. To provide a common foundation for Internet service providers, network operators, and equipment manufacturers to deliver a wide range of e-services via gateway servers running in the home or remote office, OSGi addresses the following requirements:

- *Platform independence*. Embedded devices are generally much more diverse than desktop computers. Each gateway device usually has its own native platform and uses various combinations of processors and operating systems. It would be virtually impossible for service developers to write applications and port them to every conceivable platform, accounting for the drastically different underlying characteristics of each. This problem can be solved by leveraging Java's "write-once, run-everywhere" value proposition and by agreeing to a common API standard for gateway software.
- *Vendor independence*. Open-specification APIs should let residential gateways from different vendors host services written by different independent software vendors (ISVs) and provisioned by different service providers. Today, when a user switches from one home security provider to another, the entire internal network has to be replaced. By choosing devices compatible with the Open Services Gateway specification, however, users can switch between various vendor offerings with virtually no change to the networking infrastructure.
- *Future-proofing*. An open standard should ensure that a user can keep the same residential gateway hardware box when changing service providers or adding new services. Java's dynamic upgradability allows the same hardware to run additional services or even an entirely different set of services; thus, the user does not have to buy a new gateway box and the network operators do not have to visit the home to do manual upgrades.
- *Coexistence and integration with multiple LAN and device-access technologies*. An open standard-based software platform should accommodate existing technologies so that, for example, a Jini-enabled printer, a HAVi-compatible A/V receiver, and a UPnP camera can all interoperate.

The OSGi was started, in part, because Sun Microsystems' release of the Java Embedded Server (JES) 1.0 attracted the attention of several software companies, residential gateway hardware vendors, and service providers. A few of them, including Ericsson, Motorola, and Sun, began working to standardize the JES APIs to help ensure interoperability in the residential gateway software market.

In October 1999, fifteen companies formed OSGi and established its Java Expert Group.[2] Sun con-

tributed the JES 1.1 specification (the current version at the time), which became the foundation for the specification work. The group eventually produced version 1.0 of the OSGi Service Gateway specification, which was approved by the OSGi board of directors and released in May at Connections 2000. OSGi has attracted a lot of attention and, as of this writing, has more than 70 member companies.

Figure 2 illustrates OSGi's operational model of the residential gateway, which is strongly influenced by a business model that includes gateway manufacturers, suppliers of OSGi-compliant servers, gateway operators and service providers, and ISVs that produce services that run inside the gateways and on service providers' back-end systems. This implies that, in designing the software architecture, the entire structure was divided into modules that generally correspond to various functions provided by different business entities, and the interfaces between those modules are defined to ensure interoperability.

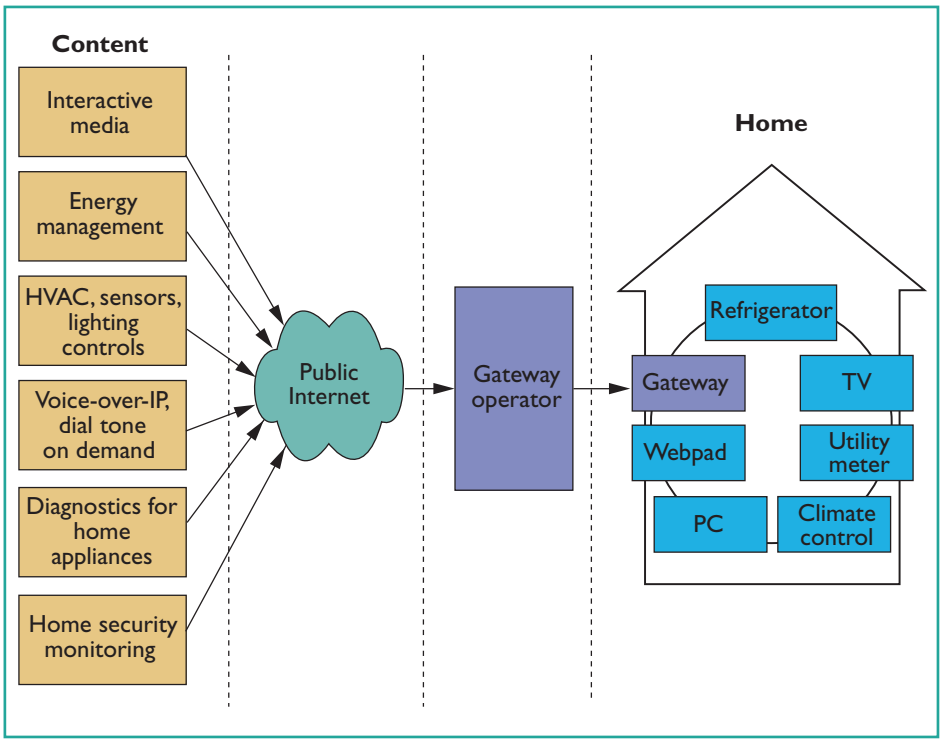

Figure 2. Residential gateway usage scenario. Service providers, service developers, network operators, and gateway device manufacturers work together to provision services to the home.

Service Framework

Central to the OSGi specification is the service framework. Its primary goal is to use the Java language's platform independence and dynamic code-loading abilities to simplify developing and dynamically deploying applications for small-memory devices. The framework also defines several important concepts for creating extensible services:

- A *service* is a self-contained component, accessible through a defined service interface, that includes the Java classes that perform certain functions. An application is built around a set of cooperating services and can extend its functionality during runtime by requesting more services. The framework registers and installs services, maintains a set of mappings to their implementations, and manages the dependencies among them. It also has a simple query mechanism (LDAP-based syntax) that lets an installed service request and use other available services.

Different types of gateway devices usually have their own native platforms.

Note that a gateway operator and a service provider are often the same party, although a gateway operator might aggregate and package services from different service providers.

Software Architecture

The OSGi specification is a collection of standard APIs that define a service gateway. They address service delivery and remote service administration, as well as dependency, life-cycle, resource, and device management. It is beyond this article's scope to go into the sometimes complicated details of the APIs, such as the semantics and usage.[1] Instead, I give a broad-brush view of the architecture's major concepts and compare the technical advantages and disadvantages of the OSGi approach and a few alternatives.

Competing Standards for Networking the Home

There seems to be consensus among industry players that the existing infrastructure should be leveraged without introducing new wires to network the home. Promising contenders for the physical media to be used include

- *Radio frequency*. Transmission speeds range from 1 to 2 Mbps. (Home Radio Frequency working group, http://www.homerf.org).
- *Infrared*. Transmission speeds range from 9600 bps to up to 4 Mbps. (Infrared Data Association, http://www.irda.org).
- *Phoneline*. Transmission speeds range from 1 to 10 Mbps. Standard defined by the Home Phoneline Network Alliance (HomePNA, http://www.homepna.org).
- *Powerline*. Transmission speed is 14 Mbps. (Home-Plug Powerline Alliance, http://www.homeplug.org).
- *TV cable*. Transmission speeds are 27 Mbps downstream and 500 Kbps to 10 Mbps upstream. (Cable Television Laboratories, http://www.cablelabs.com).

There are also many interfaces for connecting with devices.

- Serial port with a data rate of 115 Kbps.
- Standard parallel port with a data rate of 115 Kbps.
- ECP/EPP parallel port with a data rate of 3 Mbps.
- IDE with a data rate of 3.3-33 Mbps.
- USB (Universal Serial Bus) with a data rate of 12 Mbps.
- SCSI with a data rate of 5-160 Mbps.
- IEEE 1394 (FireWire or iLink) with a data rate of 100-400 Mbps.

Many higher-level protocols are also competing. These protocols are generally built on top of the physical and data-link layers, and focus more on the service functionality provided by applications.

- *LonWorks* (http://www.echelon.com) uses powerline, twisted pair, and so on to control devices such as sensors and switches.
- *CEBus* (Consumer Electronics Bus, http://www.cebus.org) uses powerline, twisted pair, coaxial cable, and radio frequency for communication among residential consumer products.
- *Bluetooth* (http://www.bluetooth.com) uses a 2.4-GHz radio link for sending and receiving voice and data between portable wireless devices within 10 meters.
- *HAVI* (Home Audio Video Interoperability, http://www.havi.org) uses APIs defined on top of IEEE 1394 that allow digital consumer electronics and home appliances to communicate with one another.
- *Jini* (http://www.sun.com/jini) uses Java APIs to enable a device to join a Jini federation and publish its service without prior setup. Others in the Jini community can look up and discover the service, which is represented as a Java object that defines the service interface; the object is exchanged using RMI.
- *UPnP* (Universal Plug and Play, http://www.upnp.org) uses IP networks to let devices discover, utilize, and control devices, which are located by IP addresses; the services they provide and how they are controlled are described by XML-encoded messages.

Some of these protocols address particular application domains (LonWorks, CEBus, Bluetooth, and HAVI, for example) while others (such as Jini and UPnP) attempt to provide a generic framework for enabling all devices.

- A *bundle* is the functional and deployment unit for shipping services. It is a JAR file that contains the resources implementing zero or more services—these can be Java class files or any other data (such as HTML help files or images). It also contains a Manifest file with headers specifying various parameters (the Java packages the bundle provides, the packages it needs, the way it locates its activator, and so on) that allow the framework to correctly install and activate the bundle. A bundle can be in any one of the following states: `INSTALLED`, `RESOLVED`, `STARTING`, `STOPPING`, `ACTIVE`, and `UNINSTALLED`.
- A *bundle context* represents a bundle's execution environment in the framework. For example, a bundle context can tell how to access the persistent storage area for a particular bundle.
- The framework can generate three kinds of *events*: a `ServiceEvent` reports the registration, unregistration, or property changes for services; a `BundleEvent` reports changes in a bundle's life cycle; and a `FrameworkEvent` reports that the framework has started or has encountered errors. For each kind of event, there is a corresponding listener interface that lets interested parties receive event notifications and act on them.
- *Security* in the framework is based on the Java 2 security architecture.[3] Many methods defined by the API require the caller to have certain *permissions*, and services may also have specific permissions that provide finer grained control over the operations they can perform. Thus, a bundle that exposes services to other bundles might also need to define permissions specific to the exposed services. The gateway operator must configure the gateway's system policy to grant the required permissions to bundles that are to be installed and run in the gateway.

The architecture diagram in Figure 3 shows how other services can be plugged into the framework once it is up and running.

This pluggable framework is ideal for just-in-time service delivery because it allows the gateway device to download a service over the network only when it is needed. The service can be used once and then discarded, or it can be kept on the residential gateway. The framework also supports service updates and versioning. It can be used to quickly check the version of a service that is running in a gateway and to update this service dynamically from a remote location, which is very useful in developing software for embedded devices. These devices have traditionally been loaded with a static application environment, which means any application bug can quickly become critical and expensive to fix. Using the OSGi framework, however, a newer version of the software can be loaded via the network.

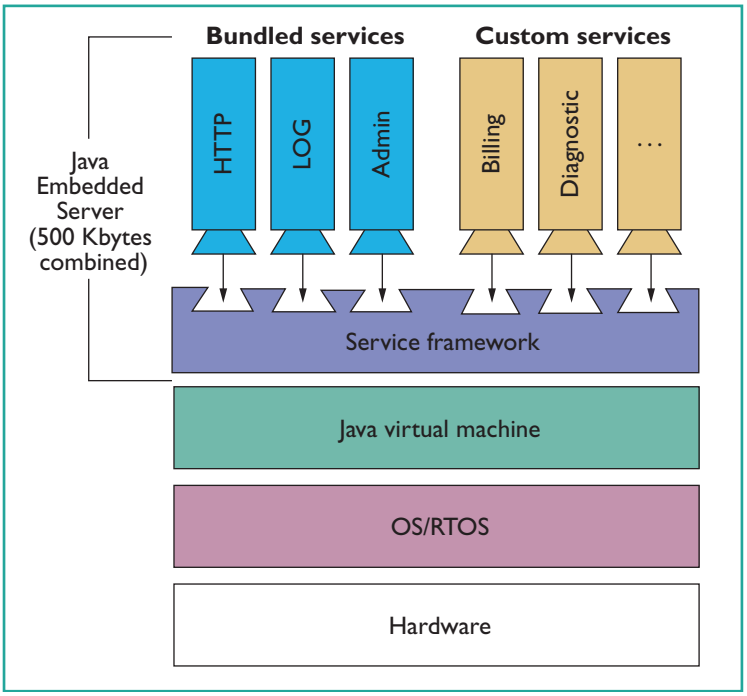

Figure 3. Service framework with pluggable services. The OSGi framework enables users to access modular services from various providers.

Implementations

Several software vendors are producing OSGi 1.0-compliant products. Gatespace (http://www.gatespace.com/) provided OSGi's reference implementation, and Sun, IBM, and others are now offering competing products. Developers can thus create one set of services that will run on all products compliant with the OSGi specification. In this section I dissect the JES, partly to show that an actual product offering tends to be more complex than a typical software architecture diagram shows, and partly to set the stage for our comparison later in this section.

JES 2.0 offers life-cycle management and dependency management, which the Service Gateway Specification 1.0 says are mandatory, as well as three OSGi-defined services (such as APIs for HTTP, log, and device access), which are optional. As for the underlying Java platform, JES 2.0 runs on top of PersonalJava 3.0 and JDK 1.2.2. To make life easier for developers, the product comes with a tool for remote diagnostics over HTTP.

Having a gateway is not an end in itself; rather, its goal is to deliver managed services. A companion product, the Provisioning Pack, serves this purpose by letting service providers manage multiple residential gateways. For example, when a new version of a service is rolled out, service providers must arrange orderly updates for all customers, ensuring a smooth transition with minimal interruptions. The Provisioning Pack can provide a high-level topology, including status, of all the gateways under its control, and it allows service providers to configure services for each gateway. Since the home is likely to have several different networking technologies, a Connectivity Pack enables devices equipped with these diverse networking technologies to interoperate with each other using the gateway as an intermediary.

The companion Provisioning Pack lets service providers manage multiple residential gateways.

To Java or Not to Java

In the first version of the standard, OSGi specified software for service gateways in terms of Java APIs.[4,5] This does not mean there are no other technical alternatives, or that OSGi will not issue specifications in other formats in the future, but in this case, Java's advantages outweighed its disadvantages. It was especially useful toward OSGi's goal of enabling remotely managed, dynamically downloadable or upgradeable services.

Specifications for open standards that can be implemented by multiple vendors are normally expressed in terms of wire-format protocols (think of the TCP/IP specification) or interfaces in a programming language (think of WinSock). In the OSGi situation, we can examine three possible choices: wire-format protocols, Java APIs, and APIs in other languages such as C/C++. Table 1 summa-

Table 1. Comparison of alternative approaches for specifying the OSGi standard.

Approach	Platform independence	Expressiveness	Extensibility	Security	Footprint
Wire format	plus	minus	minus	neutral	plus
Java	plus	plus	plus	plus	minus
C/C++	minus	plus	plus	minus	plus

rizes the merits of these options using a plus to indicate an advantage and a minus for a disadvantage.

The advantage of specifying a wire-format protocol is that it is possible (though not always feasible) to avoid requiring a heavyweight runtime environment. Engineers thus have room to develop lightweight implementations. The disadvantage of using a wire-format protocol is that expressiveness and extensibility are limited. Once a gateway is deployed, it could be hard to upgrade the software to offer new services unless a framework like OSGi's is also installed to manage upgrades. Also, it is hard to imagine developing rich services using a set of fixed wire-format protocols.

The advantages of specifying Java APIs include platform independence, the ability to dynamically load code, a rich and extensible object-oriented programming language, and a built-in security architecture. The biggest drawback is that a Java runtime environment must be present to run Java programs. Although the OSGi 1.0 specification does not require a specific underlying runtime Java platform, the smallest platforms for practical purposes are PersonalJava 3.0 or JDK 1.1. Each has a footprint of a couple of megabytes and demands a few extra megabytes of RAM beyond what is required for the underlying OS or RTOS. It could be argued that many features in these platforms are unnecessary for OSGi, but throwing them out would break Java platform compatibility. Going forward, perhaps the best way to reduce the footprint would be to find a suitable profile—or to specify a new one for gateway devices—that is powered by much smaller runtime environments and can run on top of Java 2 Micro Edition (J2ME). Such a standard would have to go through the Java Community process.

The biggest advantage of specifying C/C++ APIs is that no Java runtime environment is required. This reduces the footprint, but it also means you lose platform independence and an extensive security architecture that can guard against downloaded code.

Conclusion

The introduction of broadband services to the home and advances in home appliance and networking technologies have brought into the spotlight the role of the residential gateway as the platform for provisioning managed services. The gateway will coordinate and bridge the heterogeneous set of home networking standards and multitude of smart appliances. Many service providers are working with ISVs to design a wide range of services. It won't be long before innovative applications, some perhaps as unexpected as XCoffee, are deployed and delivered to our doorsteps.

Acknowledgments

I would like to thank the Java Embedded Server product team and OSGi's Java Expert Group members for producing the technical work described in this article. I would also like to thank Kirk Chen who is coauthoring a forthcoming book on OSGi and JES programming[1] from which this article has borrowed.

References

1. K. Chen and L. Gong, *Programming Open Service Gateways with Java Embedded Server*, Addison Wesley Longman, Reading, Mass., 2001.
2. L. Gong, "Java Supports Service Gateways," *EE Times*, 1089:112, Nov. 1999.
3. L. Gong, *Inside Java 2 Platform Security*, Addison-Wesley, Menlo Park, Calif., June 1999.
4. J. Gosling, B. Joy, and G. Steele, *The Java Language Specification*, Addison-Wesley, Menlo Park, Calif., Aug. 1996.
5. T. Lindholm and F. Yellin, *The Java Virtual Machine Specification*, Addison-Wesley, Menlo Park, Calif., 1997.

Li Gong is a Distinguished Engineer and director of engineering, peer-to-peer networking, at Sun Microsystems. He was the founding chair of OSGi's Java Expert Group where he led the completion of the Service Gateway Specification 1.0. He was director, server products, at the Consumer and Embedded division, where he led Sun's effort in home networking technologies and products. Gong received a BS and MS from Tsinghua University, Beijing, China, and a PhD from the University of Cambridge, England. He is on the editorial board of *IEEE Internet Computing*.

Readers can contact the author at li.gong@sun.com.