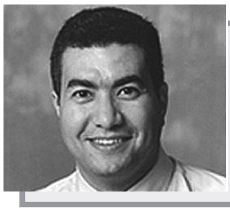


Editor: Sumi Helal ■ University of Florida ■ helal@cise.ufl.edu

Standards for Service Discovery and Delivery

Sumi Helal, University of Florida

For the past five years, competing industries and standards developers have been hotly pursuing automatic configuration, now coined the broader term *service discovery*. Jini, Universal Plug and Play (UPnP), Salutation, and Service Location Protocol are among the front-runners in this new race. However, choosing service discovery as the topic of the hour goes beyond the need for plug-and-play solutions or support for the SOHO (small office/home office) user. Service discovery's potential in mobile and pervasive computing environments motivated my choice.

Mobility means getting away from configured environments and into foreign networks with unknown infrastructures. However, because a mobile computer can't predict such infrastructures, it might not know to take advantage of them or even have the capabilities to interact with them. For example, a mobile computer might not be able to use a nearby printer because it does not have the appropriate printer driver, or perhaps a PDA will experience slow Web access because it is not aware of a nearby Web proxy caching server. As mobile computing evolves beyond the ability to wirelessly connect to read email or surf the Web anywhere and on any device, it is bound to exploit local resources, peers, and services. With the advent of location-based services and peer-to-peer computing, service discovery is taking on a new role as a critical middleware for mobile computing.

Service discovery can also benefit pervasive computing environments, where numerous computing elements and sensors often must interact to achieve the desired functionality and intelligence. In such environments, self-advertisement and peer discovery can enable the pervasive space to dynamically change and evolve without major system reengineering.

SERVICE DISCOVERY PROTOCOLS

Consider the following three scenarios. First, imagine finding yourself in a taxi without your wallet. Fortunately, you have a Jini technology-enabled cellular phone, and your cellular provider uses Jini technology to deliver network-based services tailored to your community. On your phone screen, you see a service for the City Cab Company, so you download the electronic payment application to authorize paying your taxi fare. The company's payment system instantly recognizes the transaction and sends a receipt to the printer in the taxi. You take the receipt, and you're on your way.

Second, consider an insurance salesman who visits a client's office. He wants to brief the client on new products and their options, which are stored in his Windows CE handheld PC. Because his handheld PC has wireless network and supports UPnP, it automatically discovers and uses an Ethernet-connected printer without any network configuration and setup. He can print whatever he wants from his handheld and promote the new products.

Finally, consider an intelligent, online overhead projector with a library client. After being authenticated, the user might select a set of electronically stored charts or other documents for viewing. Rather than bringing transparencies to a meeting, the user accesses them through the LAN server in the library.

Scenario 1 is a Jini demo scenario from Sun Microsystems, Scenario 2 is a UPnP scenario from Microsoft, and Scenario 3 comes from Salutation. At a glance, they all seem to talk about the same stories: mobile devices, zero configuration, impromptu community enabled by service discovery protocols (SDPs), and cooperation of the proximity network. Without mentioning the trademarks, we would hardly know which company is telling which scenario. These SDPs, however, have different origins. They see the problem from different angles and have different approaches for solving it.

GLOSSARY

DA	Directory agent
DHCP	Dynamic Host Configuration Protocol
IETF	Internet Engineering Task Force
RMI	Remote method invocation
SA	Service agent
SDP	Service discovery protocol
SLP	Service location protocol
SOHO	Small office/home office
SSDP	Simple service discovery protocol
UA	User agent
UDP	User Datagram Packets
UPnP	Universal Plug and Play

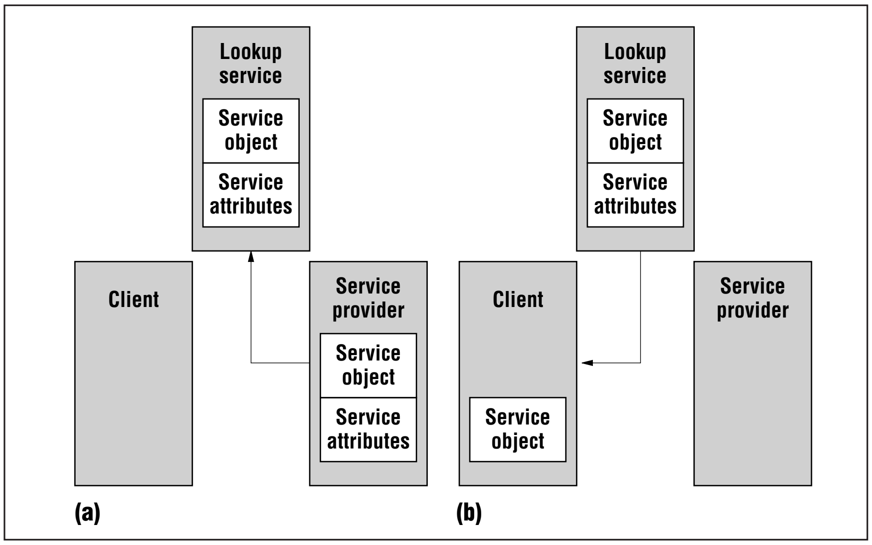


Figure 1. (a) A service provider registers a service object and its service attributes with the lookup service. (b) A client requests a service from service attributes, and a copy of the service object moves to the client.

JINI

Sun Microsystems introduced Jini, based on the Java technology, in 1998. The heart of Jini is a trio of protocols: discovery, join, and lookup. A pair of these protocols—discovery and join—occurs when you plug a Jini device into a network; discovery occurs when a service looks for a lookup service with which it can register, and join occurs when a service locates a lookup service and wants to join it. Lookup occurs when a client or user locates and invokes a service described by its interface type (written in the Java programming language) and possibly other attributes. For a client in a Jini community to use a service,

- The service provider must locate a lookup service by multicasting a request on the local network or a remote lookup service known to it a priori (see Figure 1a).
- The service provider must register a service object and its service attributes with the lookup service. This service object contains the Java programming language interface for the service, including the methods that users and applications will invoke to execute the service, along with any other descriptive attributes (see Figure 1a).
- A client then requests a service by Java type and perhaps other service attributes. The lookup server ships a copy of the service object over the network to the client, who uses it to talk to the service (see Figure 1b).
- The client interacts directly with the service via the service object (see Figure 1b).

Jini technology consists of an infrastructure and a programming model that address how devices connect with each other to form an impromptu community. Jini uses the Java remote method invocation (RMI) protocol to move code around the network.

We can view the Jini lookup service as a directory service or broker. Jini uses three related discovery protocols. When an application or service first becomes active, the multicast request protocol finds lookup services in the vicinity. Lookup services use the multicast announcement protocol to announce their presence to services in the community that might be interested. The unicast discovery protocol then establishes communications with a specific lookup service known a priori over a wide-area network.

However, a Jini lookup service is not just a simple name server. It maps the interfaces that clients see to service proxy objects. It also maintains service attributes and processes match queries. Clients download the service proxy, which is usually an RMI stub that can communicate back with the server. This proxy object lets clients use the service without knowing anything about it. Hence, there is no need for device drivers in case the service provided is a device (such as a printer). Although service proxy objects represent a typical scenario of service invocation, the downloaded service object can be the service itself or a smart object capable of speaking in any private communication protocol.

Leasing in Jini

Jini grants access to its services on a lease basis. A client can request a service for a desired time period, and Jini will grant a negotiated lease for that period. This lease must be renewed before its expiration; otherwise, Jini will release the resources associated with the service. Leasing lets Jini be robust and maintenance-free when faced with abrupt failures or the removal of devices and services.

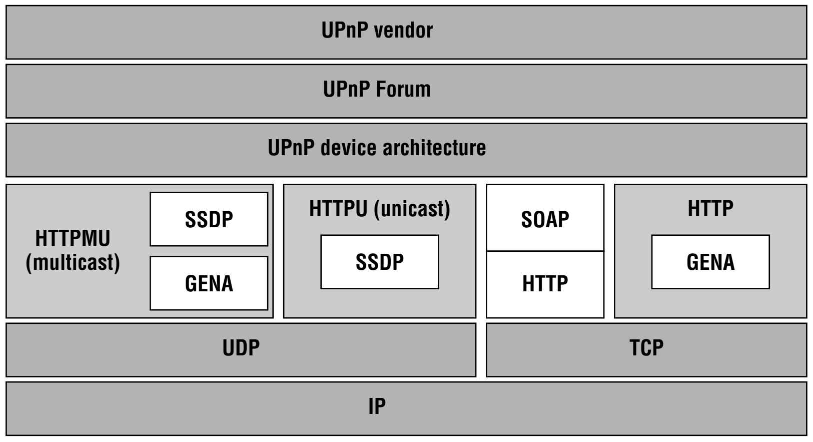
Distributed programming in Jini

Besides the basic service discovery and join-and-lookup mechanism, Jini supports *remote events* and *transactions* that help programmers write distributed programs in a reliable and scalable fashion. Remote events notify an object when desired changes occur in the system. Newly published services or some state changes in registered services can trigger these events. For example, the lookup service can notify a Jini palmtop that has registered its interest in printers when a printer becomes available. Also, Jini supports the notion of transactions and flexible notions of atomic commitment.

Anyone interested in Jini can participate and contribute to the standard by joining the Jini Forum (www.jini.org). Sun Microsystems acts as the steward for this forum.

UNIVERSAL PLUG AND PLAY

UPnP is an evolving Microsoft-initiated standard that extends the Microsoft Plug-and-Play peripheral model. It aims to enable the advertisement, discovery, and control of networked devices, services, and consumer electronics. In UPnP, a device can dynamically join a network, obtain an IP address, convey its capabilities on request, and learn about the presence and capabilities of other devices. A device can also leave a network smoothly and automatically without leaving any unwanted state behind. UPnP leverages TCP/IP and Web technologies, including IP, TCP, UDP, HTTP, and XML. It uses the protocol stack in Figure 2 for service discovery, advertisement, description, and eventing.



GENA:	General Event Notification Architecture
SOAP:	Simple Object Access Protocol
SSDP:	Simple Service Discovery Protocol
UDPs:	User datagram packets
UPnP:	Universal Plug and Play

Figure 2. Universal Plug and Play protocol stack.

Joining and discovery in UPnP

UPnP uses simple service discovery protocol (SSDP) for service discovery. This protocol announces a device's presence to others and discovers other devices or services. Therefore, SSDP is analogous to the trio of protocols in Jini. SSDP uses HTTP over multicast and unicast UDP, referred to as HTTPMU and HTTPU, respectively.

A joining device sends out an *advertisement* (ssdp:alive) multicast message to advertise its services to control points. Control points function similar to Jini's lookup services. A control point, if present, can record the advertisement, or other devices might also directly see this multicast message. In contrast to Jini, UPnP can work with or without the control points (lookup service). It sends a *search* (ssdp:discover) multicast message when a new control point is added to a network. Any device that hears this multicast will respond with a unicast response message.

UPnP uses XML to describe device features and capabilities. The aforementioned advertisement message contains a URL that points to an XML file in the network that describes the UPnP device's capability. By retrieving this XML file, other devices can inspect the advertised device's features and decide whether it is important or relevant to them. XML allows complex and powerful description of device and service capability as opposed to Jini's simple service attribute.

UPnP service description

After a control point has discovered a device, it learns more about how to use it, control it, and coordinate with it by retrieving its XML description file. Control is expressed as a collection of Simple Object Access Protocol (SOAP) objects and their URLs in the XML file. To use a specific control, a SOAP message is sent to the SOAP control object at the specified URL. The device or the service returns action-specific values.

A UPnP description for a service includes a list of actions to which the service responds and a list of variables that model the service's state at runtime. The service publishes updates when these variables change, and a control point can subscribe to receive this information. Updates are published by sending event messages that contain the names and values of one or more state variables. These messages are also expressed in XML and formatted using the General Event Notification Architecture.

UPnP features an additional higher-level description of services in the form of a user interface. This feature lets the user directly control the service. If a device or service has a *presentation URL*, then the control point can retrieve a page from this URL, load the page into a browser, and (depending on the page's capabilities) let a user control the device or view the device's status.

Automatic configuration of IP

Another important feature of UPnP is automatic configuration of IP addresses. AutoIP lets a device join the network without any explicit administration. When a device connects to the network, it tries to acquire an IP address from a Dynamic Host Configuration Protocol server. However, in the absence of a DHCP server, an IP address is claimed automatically from a reserved range for local network use. The device claims an address by randomly choosing one from the reserved range and then making an ARP request to see if anyone else has already claimed that address.

Headed by Microsoft, the UPnP Forum (www.upnp.org) oversees the standard's developments. The standard development process is similar to the Java Community Process.

SALUTATION

The Salutation Consortium is devel-

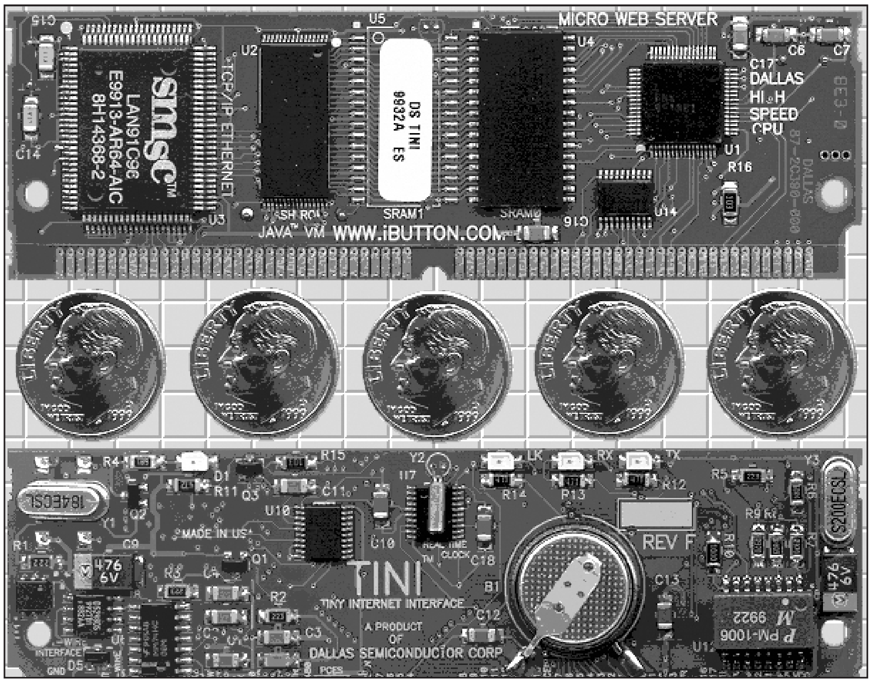


Figure 3. Dallas Semiconductor's Tiny Internet Interface board running Jmatos (front and back).

oping another standard, called Salutation, for service discovery and use, especially among devices and services of dissimilar capabilities. The Salutation architecture provides a standard method for applications, services, and devices to describe and advertise their capabilities to other applications, services, and devices. The architecture enables search and discovery based on particular capabilities.

The architecture is composed of two major components: the *Salutation manager* and *transport manager*. The Salutation manager is the core of the architecture and is similar to the lookup service in Jini or control point in UPnP. It is defined more as a service broker. A service provider registers its capability with a Salutation manager. When a client asks its local Salutation manager for a service search, all the Salutation managers coordinate to perform the search. Then, the client can use the returned service. A Salutation manager sits on the transport managers that provide reliable communication channels, regardless of the underlying network transports.

The Salutation manager provides a transport-independent interface to server and client applications. This interface (SLM-API) includes service registration, service discovery, and service access function. The interface between the Salutation manager and transport manager (called SLM-TI) achieves communication protocol independence in the Salutation architecture. The transport manager is an entity, dependent on the network transport it supports. A Salutation manager might have more than one transport manager, in case it is attached to multiple, physically different networks. However, the Salutation manager sees its underlying transport through the SLM-TI, and it performs the following tasks.

Service registration

The Salutation manager contains a registry to keep information about services, and a client can register or unregister itself. All registrations are done with the local Salutation manager or a nearby one connected to the client.

Service discovery

The Salutation manager discovers other Salutation managers and registered services. It discovers remote services by matching types and sets of attributes specified by the local Salutation manager. This unique feature, called *capability exchange*, is needed because services are basically registered with the local Salutation manager in the same equipment. This cooperation among Salutation managers forms a conceptually similar lookup service to Jini. One difference, though, is that it is distributed over the network.

Service availability

A client application can ask the local Salutation manager to periodically check the availability of services. This procedure is done between the local manager and the corresponding manager. This is a weaker version of Jini and UPnP's eventing services.

Service session management

The service session is operated in one of three modes: *native*, *emulated*, or *salutation*. The Salutation manager might not be involved in message exchanges in the service session, depending on the modes. The native mode exchanges through a native protocol—the Salutation manager is never involved in the message exchange. In the emulated mode, the Salutation manager protocol carries messages between the client and service but doesn't inspect the contents, and in the salutation mode, Salutation managers not only carry messages but also define the formats to be used in the session.

A *functional unit* is a basic building block in the Salutation architecture. It is the minimal meaningful function to constitute a client or service. A collection of functional units defines a service record. For example, the functional units [Print], [Scan], and [Fax Data Send] can define a fax service. Each functional unit is composed of a descriptive attribute record, specified in ISO 8824 ASN.1.

Salutation-Lite

Salutation-Lite is a scaled-down version of the Salutation architecture targeted at devices with small footprints. It is obviously more applicable to small information appliances such as palm-size and handheld computers. Salutation-Lite also lends itself well to low-bandwidth networks such as IR and Bluetooth.

The Salutation standard is overseen by the Salutation Consortium (www.salutation.org), which provides five levels of membership.

SERVICE LOCATION PROTOCOL

Service Location Protocol is an Internet Engineering Task Force (IETF) standard for decentralized, lightweight, and extensible service discovery. It uses service URLs, which defines the service type and address for a particular service. For example, "service:printer:lpr://hostname" is the service URL for a line printer service available at hostname. Based on the service URL, users (or applications) can browse available services in their domain and select and use the one they want.

There are three agents in SLP: the *user*, *service*, and *directory*. The UA is a software entity that sends service discovery requests on a user application's behalf. The SA broadcasts advertisements on behalf of a service. As a centralized service information repository, the DA caches advertisements from SAs and processes discovery queries from UAs. An SA advertises itself by registering with a DA. The registration message contains the URL for the advertised service and for the service's lifetime, and a set of descriptive attributes for the service. The SA periodically renews its registration with the DA, which caches the registration and sends an acknowledge message to the SA. A UA sends a service request message to the DA to request the service's location. The DA responds with a service reply message that includes the URLs of all services matched against the UA request. Now, the UA can access one of the services pointed to by the returned URL. In SLP, the DA is optional. A DA might not exist in a small network, in which case the UAs' service request messages are directly sent to the SAs.

SLP supports service browsing and string-based querying for service attributes, which let UAs select the most appropriate services from among available services in the network. SLP lets UAs issue query operators such as AND, OR, comparators, and substring matching. This is more powerful than Jini and UPnP service attribute matching, which can be done only against equality.

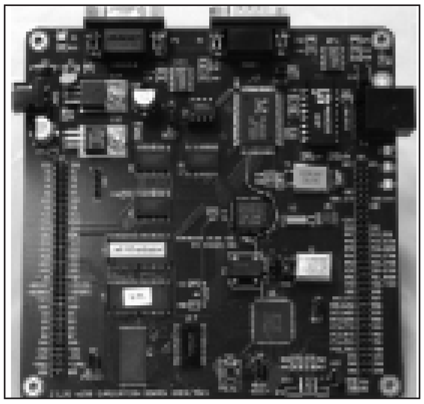


Figure 4. A Zilog's eZ80 Micro Web server running Metrolink IPWorks with UPnP support.

The SLP standard is accessible from the IETF SvrLoc working group Web site (see www.ietf.org).

BLUETOOTH SDP

Unlike Jini, UPnP, Salutation, or SLP, the Bluetooth SDP is specific only to Bluetooth devices (see www.bluetooth.com). It primarily addresses the service discovery problem. It doesn't provide access to services, brokering of services, service advertisements, or service registration, and there's no event notification when services become unavailable. SDP supports search by service class, search by service attributes, and service browsing. The latter is used when a Bluetooth client has no prior knowledge of the services available in the client's vicinity. SDP is structured as a Bluetooth profile and runs on a predefined connection-oriented channel of the L2CAP Logical Link layer. Salutation has proposed a mapping between its service discovery and Bluetooth SDP. Such mapping is synergistic because it complements Bluetooth by adding advertisements, brokering, and eventing. Bluetooth, on the other hand, serves Salutation by fitting in as a transport (Salutation is transport-independent) in the heart of the devices.

MARKET ACCEPTABILITY

Not many Jini products are available on the market today. In 1999, a year after Jini's introduction, companies such as Epson, Canon, Seagate, and Quantum agreed to embed Jini in some of their product lines. However, later in the same year, these companies warned that it might take up to two years to accomplish this. People then predicted that Jini storage devices would be first to hit the market, but plans for Jini products are still uncertain. On the brighter side, PsiNaptic, a leader in pervasive computing, has delivered a critical Jini product, Jmatos. Jmatos supports Dallas Semiconductor's (now Maxim's) Tiny Internet Interface (known as Tini), which has an embedded Java Virtual Machine. Tini is a small-footprint microcontroller with a rich set of on-board interfaces (see Figure 3).

Unlike Jini, many UPnP products are available on the market today. In addition to the hundreds of thousands of Windows XP machines (which come with UPnP support), the following products are currently available:

- Gatespace and MetroLink's OSGi/UPnP connectivity products. These products significantly move use of the OSGi standard toward pervasive computing. OSGi is primarily designed to deliver Web services to the home. Figure 4 shows a UPnP version of the Tini product, a Zilog's eZ80 Micro Web server running Metrolink IPWorks with UPnP support.
- IBM's Home Director, an X10-based home networking software system for appliance control. Incorporating UPnP with Home Director makes its ControlPoint software accessible from a variety of devices such as Web pads and PDAs.
- Intel's AnyPoint Home Router, the first in its class to be UPnP certified, removes the burden of setup by the SOHO user and makes creating a LAN at home really hard. Several other vendors of home gateways, including Linksys and D-Link, now use UPnP.
- Software development kits. At least

eight UPnP SDKs are available today that would let developers of devices, consumer electronics, and embedded systems build UPnP support into their products (Allegro Software, Virata, Intel, Lantronix, Atinav, Metrolink, Microsoft, and Siemens). Support for these SDKs is not limited to the Windows or Windows CE platform—it includes the Linux platform and supports C, C++, and Java.

Several Salutation products are available but most are office automation products—fax machines, printers, copiers, and scanners. IBM's NuOffice, a networked office system based on Lotus Notes, lets users import and export data to any Salutation device.

Backed by IETF and aligned with other established protocols (including Lightweight Directory Access Protocol, Domain Naming System, and DHCP), developers have widely accepted SLP as a simple, minimum requirement service discovery protocol. Another source of this acceptance is SLP's scope, because it attempts only to locate—not access or deliver—the service. SLP is used by Hewlett Packard's JetSend technology, which supports HP's office equipment and consumer electronics. Other vendors with SLP printer and network products include Axis, Lexmark, Xerox, Minolta, IBM, Novel, and Zephyr, and Axis also offers SLP storage devices. In addition to office and networking equipment, several platforms support SLP, including Sun, Caldera, Novel, and Apple.

Service discovery has come a long way to becoming a major standardization and development effort, but the picture is not as impressive when we consider market acceptability and available products. In addition, in their current form and shape, most service discovery standards do not address mobility's needs and special requirements. Their potential use in mobile and pervasive environments is therefore uncertain.

Current SDPs are designed for use in local area networks. The IP multicast range, for example, limits discovery in Jini. This is inadequate for mobile clients requiring access to services from wide area networks. A few research projects currently underway are dealing with this problem.

Another problem with existing SDPs is their lack of support for mobile devices. For instance, Jini requires JVM and RMI capabilities on the client slide, which has hindered its widespread use on mobile devices. A quick fix to this problem was to introduce the Jini Surrogate Architecture (www.surrogate.jini.org). Using surrogates, a device does not have to have or understand JVM or RMI. It only must remember Jini code that uses RMI and be able to send that code to a proxy (the surrogate) on the local network to act on its behalf. Unfortunately, surrogates are more a solution to stationed devices than to mobile devices.

One limitation of current service discovery frameworks is that they do not consider important context information. For example, there is no support to service routing and selection based on the client's location. Other unexploited contextual information includes distance to service, time, service load, and quality of service instances. A few research projects have started to address this need. P

Sumi Helal is an associate professor in the Computer and Information Science and Engineering Department at the University of Florida. Contact him at helal@cise.ufl.edu; www.cise.ufl.edu/~helal.

next issue

The emerging .NET Compact Framework and its role in mobile application development

How to Reach Us

Writers
For detailed information on submitting articles, write for our Editorial Guidelines (pervasive@computer.org) or access http://computer.org/pervasive/author.htm.

Letters to the Editor
Send letters to

Shani Murray, Associate Lead Editor
IEEE Pervasive Computing
10662 Los Vaqueros Circle
Los Alamitos, CA 90720
smurray@computer.org

Please provide an email address or daytime phone number with your letter.

On the Web
Access http://computer.org/pervasive or http://dsonline.computer.org for information about *IEEE Pervasive Computing*.

Subscription Change of Address
Send change-of-address requests for magazine subscriptions to address.change@ieee.org. Be sure to specify *IEEE Pervasive Computing*.

Membership Change of Address
Send change-of-address requests for the membership directory to directory.updates@computer.org.

Missing or Damaged Copies
If you are missing an issue or you received a damaged copy, contact membership@computer.org.

Reprints of Articles
For price information or to order reprints, send email to pervasive@computer.org or fax +1 714 821 4010.

Reprint Permission
To obtain permission to reprint an article, contact William Hagen, IEEE Copyrights and Trademarks Manager, at whagen@ieee.org.

IEEE pervasive COMPUTING MOBILE AND UBIQUITOUS SYSTEMS