

---

## Internet access to heterogeneous home area network devices with an OSGi-based residential gateway

---

Invited Paper

Sherali Zeadally\*

Department of Computer Science and Information Technology,  
Network Systems Laboratory,  
University of the District of Columbia, Washington DC, USA  
E-mail: szeadally@udc.edu

\*Corresponding author

Priya Kubher

Telematics Software, Software and Controls,  
General Motors Corporation, Detroit, MI, USA  
E-mail: priya.kubher@gm.com

**Abstract:** Home area networks are proliferating rapidly in many residential homes. These networks are being designed to enable remote access and control to services and contents such as music, video, and data. It remains a significant challenge to design a home network that exploits different protocol architectures and standards while allowing interoperability among them. We describe the design and implementation of an architecture (known as a residential gateway) that:

- enables full interoperability among different technologies by exploiting the Open Service Gateway Initiative (OSGi) technology
- provides secure internet access to heterogeneous devices connected to a home area network.

**Keywords:** home area network; Jini; Java; OSGi; residential gateway; security; UPnP.

**Reference** to this paper should be made as follows: Zeadally, S. and Kubher, P. (2008) 'Internet access to heterogeneous home area network devices with an OSGi-based residential gateway', *Int. J. Ad Hoc and Ubiquitous Computing*, Vol. 3, No. 1, pp.48–56.

**Biographical notes:** Sherali Zeadally received the BA and MA Degrees in Computer Science from the University of Cambridge, England, and the PhD Degree in Computer Science from the University of Buckingham, England, in 1996. He is an Associate Professor at the University of the District of Columbia. He currently serves on the Editorial Boards of several international journals. He is a Fellow of the British Computer Society and a Fellow of the Institution of Electrical Engineers, UK. His research interests include computer networks (including wired and wireless networks), network and system security, mobile computing, ubiquitous computing, performance evaluation of systems and networks.

Priya Kubher received the MS Degree in Computer Science from Wayne State University. She is currently working as a Software Engineer for General Motors Corporation, Detroit, Michigan. Her research interests include middleware, computer networks, and multimedia.

---

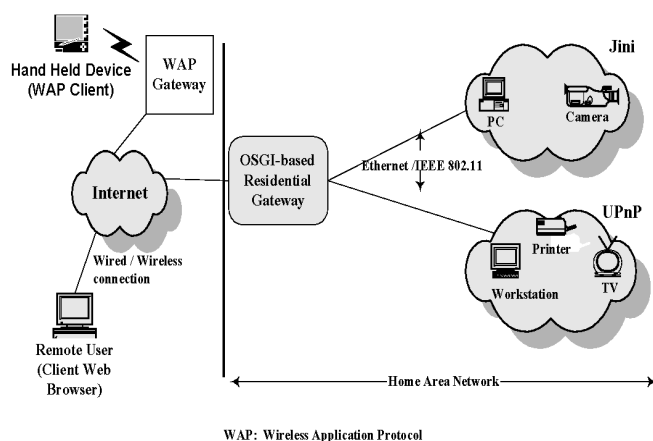
### 1 Introduction

Home area networks facilitate communication among appliances, home systems, entertainment products, and information devices in a home so that they can work cooperatively and share information. A device connected to a home area network gains the capabilities of other networked devices, and as a result the device can provide a service or function that it would have otherwise been

incapable of providing alone (CES, 2005). Several factors are pushing for the development and adoption of home area networks. These include: the advancement in telecommunications technologies, the wide proliferation of personal computers, and the decreasing costs of smart devices that allow users to control and monitor events in consumer-based appliances, and consumer demand for content rich applications.

Home area networks enable users to get information about the home's condition, to remotely control home systems and appliances, and to gain access to information and entertainment resources both from inside (such as a computer hard drive) and outside the home (for instance, from the internet) (Rose, 2001). To provide these benefits to home consumers, different devices of the home network must be able to communicate with each other to provide services despite their implementation differences. This requires the management and coordination of discovery methods that work across heterogeneous device technologies and complex home networking architectures. To achieve this management and coordination transparently without user intervention is a complex task which, until now has been the major factor responsible impeding the wide acceptance and delivery of advanced services into the home. A common solution is to exploit a central connection point often referred to as a residential gateway. The residential gateway acts as a 'bridge' between the wide area network (e.g., internet) and the home network as illustrated in Figure 1. The gateway provides users access to home devices and control over the contents and leverages two technological trends namely, the ubiquity of broadband connectivity and internet access in homes, offices, vehicles and mobile/portable devices, and the emergence of new applications and services. As Figure 1 illustrates, the residential gateway provides a central coordination connection point for different heterogeneous communication technologies such as Jini and UPnP using physical ethernet or IEEE 802.11 network connectivity (Marples and Kriens, 2001).

**Figure 1** Remote internet access to heterogeneous home area network devices via an OSGi-based residential gateway



In this paper, we discuss the design and implementation of such a residential gateway based on the Open Services Gateway Initiative (OSGi) (OSGi Forum, 2005; Gong, 2001). We place special emphasis on the development of an architecture that enables seamless interoperability among different device technologies by exploiting the OSGi architecture.

The rest of the paper is organised as follows. Section 2 briefly describes related work and the novelty of our

contribution. In Section 3, we describe and discuss the design and implementation of the OSGi-based residential gateway. In Section 4, we discuss security design and implementation issues in our OSGi gateway. Finally, in Section 5, we make some concluding remarks and present future work.

## 2 Related work and novelty of contribution

### 2.1 Related work

The OSGi expert group (OSGi Forum, 2005) is currently focusing on the development of drivers for all possible and probable protocols to be used in home networks. Saif et al. (2001) discussed the design and implementation of a residential gateway in the context of the AutoHan project. Dobrev et al. (2002) proposed a framework to extend the OSGi framework to enable redefining gateway services and to provide a smart querying technique for users and devices to look up services with particular attributes. The authors also addressed the interoperability feature and proposed implementing it through the extension of OSGi services. Wils et al. (2002) discussed the integration of Universal Plug and Play (UPnP) (UPnP Forum, 2005a) and Jini (Sun Microsystems Inc., 2005a) with the OSGi framework.

### 2.2 Novelty of contribution

As mentioned previously, one of the main goals of this work is to investigate the *interoperability* feature of OSGi by design and implementation. In doing so, our proposed design architecture enables different devices using different technologies (e.g., UPnP and Jini) to seamlessly operate with each other to provide services to home network users. In this work, we focus on OSGi to provide this capability and in this context this work differs from Saif and colleagues who do not use OSGi in their architectural design. In contrast to Wils et al. who merely proposed support for separate OSGi-Jini services, and OSGi-UPnP services, our work, on the other hand, *unifies* Jini and UPnP, and we demonstrate seamless communication between Jini and UPnP via actual implementations of Jini and UPnP drivers within the OSGi framework.

## 3 Design and implementation of an OSGi-based residential gateway

The complex diversity of home networking architectures and device technologies need to be coordinated to enable home users to fully exploit and reap their benefits. As mentioned earlier, management of these network architectures and services frequently burden most homeowners and is one of the main reasons impeding the proliferation and acceptance of sophisticated networking services into the home (Dobrev et al., 2002). The advent of OSGi simplifies management of home networks composed of multiple heterogeneous communication technologies. It is worthwhile noting that the OSGi specification provides

only the Application Programming Interface (API) rather than the underlying implementation making OSGi-based gateways both platform and application independent. This independence gives considerable freedom and flexibility to application developers and designers in their service offerings. The OSGi service platform benefits also include security, service collaboration, and multiple network support (OSGi Forum, 2005). In this work, we focus on the capability of OSGi to provide *multiple network support*. OSGi specifies a framework (similar in functionality to an application server) where it is possible to dynamically load and manage software components also known as *service bundles* (Marples and Kriens, 2001). These bundles can be instantiated by OSGi to implement specific services required.

### 3.1 Integration of discovery technologies in OSGi

In recent years, several architectures such as Universal Plug and Play (UPnP), Jini, and Salutation (Salutation Consortium, 1999) have emerged to provide plug and play capabilities requiring little planning and minimal human intervention. A fundamental component common to all of these architectures is the *service discovery concept* exploited by each. Other technologies that provide service discovery include Service Location Protocol (SLP) (Sun Microsystems Inc., 2005b) and Bluetooth (Bluetooth SIG, 2005). The Device Access Specification (DAS) of OSGi allows multiple devices to be discovered and their services advertised thereby making these services available to other devices, services, and applications. Integration of discovery technologies with OSGi is based on an import/export model. Briefly, registered OSGi devices and services are exported out of the OSGi framework. For instance, an OSGi printing service can be exported to a Jini network to appear as a Jini printer. Similarly, devices and services found by native discovery techniques can also be imported into the OSGi framework to appear as valid OSGi entities and accessible to other OSGi entities. It is this importing/exporting feature that allows cross-technology discovery and promotes interoperability among multiple device types (Dobrev et al., 2002).

Our main goals in this paper are three-fold:

- We demonstrate the capability of an implemented residential gateway in providing remote internet access to home area network devices.
- We illustrate, by design and implementation, interoperability between Jini and UPnP devices by exploiting OSGi capabilities. To achieve this, we exploit APIs of OSGi which provide access from the OSGi framework to UPnP or Jini devices and services in the network. The APIs also make OSGi services available to members of UPnP or Jini networks. Consequently, the APIs provide bidirectional service discovery between UPnP or Jini devices and OSGi services: first, UPnP-OSGi or Jini-OSGi – *importing* UPnP or Jini device services into OSGi and second, OSGi-UPnP or OSGi-Jini – *exporting* OSGi services to

UPnP or Jini networks. The software that we have designed and implemented to achieve these functionalities is called a driver bundle: the driver bundle responsible for interacting with the UPnP network will be henceforth referred as the *UPnP driver bundle* or *UPnP base driver*. The driver responsible for interacting with the Jini network will be henceforth referred to as the *Jini driver bundle* or *Jini base driver*.

- We describe the security features we have incorporated in our residential gateway design to provide secure access to the OSGi services offered by the different device technologies that constitute the home area network.

### 3.2 Internet access to home area network devices residing on UPnP and Jini networks

By enabling the HyperText Transfer Protocol (HTTP) service (as shown in Figure 1) in the OSGi framework of the residential gateway, a remote user can use a username/password combination to gain access and perform operations on the bundles and services from anywhere on the internet. This access allows users to remotely retrieve information from, and send control to, services supported by the residential OSGi-based gateway using a standard web browser. Bundle developers typically need to develop communication and user interface solutions using standard technologies such as HTTP, HTML, XML, and servlets to enable users to access the OSGi resources via the HTTP service of the OSGi gateway. The HTTP service in the Java Embedded Server executes as a lightweight HTTP server. By exploiting the HTTP server, we can register OSGi services as HTTP services. In our gateway implementation, we exploit servlets to deliver OSGi services remotely via web browsers to clients. A Uniform Resource Identifier (URI) is used to remotely access the residential gateway. Once access is granted, OSGi services become available to the remote user. These services have been previously registered with the OSGi registry by using the `registerResources()` method of `HTTPContext` object.

### 3.3 UPnP device providing an OSGi service

The UPnP device architecture specification provides the protocols for a peer-to-peer network. It specifies how to join a network and how devices can be controlled using XML messages sent over HTTP. The UPnP specifications leverage internet protocols (IP, TCP, HTTP, XML).

This section describes our implementation of an OSGi bundle that interoperates with UPnP devices and UPnP control points. The implementation is based on the UPnP device architecture specification (UPnP Forum, 2005b) and the OSGi Service-Platform Release 3 (OSGi Forum, 2005). However, we had to make some changes where necessary to test it on a *Release 2* framework as implemented by Sun on the Java Embedded Server (JES2.0). Our implementation

does not support exporting of OSGi services to a UPnP network.

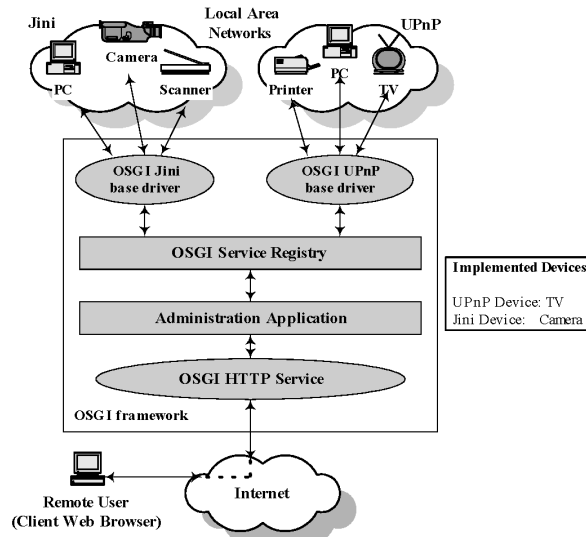
To provide a comprehensive service for a UPnP device in an OSGi framework, we designed and implemented a UPnP base driver. This base driver enables the OSGi platform to offer UPnP services to its users by means of an interface registered with the service registry. This interface is made available through the implementation of the *org.osgi.service.upnp* service package that includes core components such as *UPnPDevice* (represents an imported UPnP device on a local network), *UPnPService* (representing services provided by the device), *UPnPAction* (providing device controlling mechanisms).

There are some Software Development Kits (SDKs) available to implement UPnP devices and control points in Java, but most of them are not open source making it difficult to incorporate into the OSGi framework. The only open source SDK available (and which we chose in our implementation) is from CyberLink for Java (Konno, 2005). The implemented base driver listens on the UPnP network for device announcements. As soon as the UPnP base driver detects a new device, it extracts the information about that device according to the UPnP specification for further processing. Using this extracted (advertised) information, the UPnP base driver creates a *UPnPDevice* object and registers it in the OSGi Service Registry with a property of *DEVICE-CATEGORY = 'UpnP'*, along with other properties, such as a unique device name called UDN, similar to a Unique Resource Locator (URL) for device matching purposes. This *UPnPDevice* object is an instance of a class that represents an interface in the package '*org.osgi.service.upnp*'. This package has been implemented according to the OSGi's specifications and provides an interface for a UPnP device in an OSGi framework. The UPnP device interface exposed by this package contains the *UPnPDevice* java interface and other java interfaces which together with the UPnP base driver constitute a UPnP Device Service in an OSGi framework.

Any bundle (*ServiceListener*) listening at the OSGi's Service Registry for a service with a *DEVICE-CATEGORY = 'UpnP'*, will be informed about the UPnP device. The listener bundle can query the interface about the properties offered by this *UPnPDevice* service, and if interested can accept the service (control the device) according to the OSGi's UPnP Device Service Specification. However, in most cases, a 'refined' service is easier to use, if provided by the vendor. This refined service can provide direct access to the functionalities of the registered devices instead of dynamically finding the services and functionalities of a UPnP device. For UPnP devices in OSGi, a refined service is an OSGi service that knows about a specific device and its functionalities in detail. Such a service provides targeted functionalities of the device it is serving, through its interface in the OSGi Service Registry. This refined service does not communicate directly to a UPnP device on a UPnP network; instead it utilises already registered generic *UPnPDevice* (as mentioned above). The difference between a generic

*UPnPDevice* Service and a refined service is that a generic device service exposes the device's functionalities dynamically and a user does not know a priori these functionalities whereas a refined service is static, and the consumer knows in advance the functionalities available. For example, a *UPnPDevice* Service interface provides a service for getting and invoking an action name, its arguments, and possible values associated with a state variable of the UPnP device on the network. Thus, to utilise a generic *UPnPDevice* service, a user input using menu options and their description for achieving particular goals becomes necessary. However, in the case of a refined service for the same device, the exposed functionalities such as *setPower* (on or off operations), or *getPowerStatus* (to know whether the UPnP device is on or off) may be offered. In our implementation, we simulated a TV device (henceforth referred to as *UPnPTV*) that provides a UPnP service (as shown in Figure 2). The Unique Device Name (UDN) for this TV device is *edu.wayne.upnp.tv001* (in our implementation). As mentioned earlier, this UDN is a unique URL. However, the given format is our choice and is not specified by the UPnP forum. The UPnP base driver in the OSGi's framework detects this device, communicates with it, and after some information exchange, the base driver finally registers a *UPnPDevice* Service in the OSGi's Service Registry. Based on the device matching criteria using properties such as UDN registered with the device service, or exposed by registered service interface, a refined service named *edu\_wayne\_upnp\_tv001* (can be offered by some device vendor – but in this case provided by our implementation) associates itself to this *UPnPDevice*. This refined service exports its interface so that other bundles in the OSGi framework know the functionalities this service offers. Some of the functionalities we have implemented include operations such as turning ON and turning OFF the TV, or getting the current status of the TV.

**Figure 2** Design of a residential gateway enabling internet access to UPnP and Jini devices and interoperability among them



**Implemented Devices**  
 UPnP Device: TV  
 Jini Device: Camera

Some functionality of a TV may involve video playback from some video source. This video stream may be provided by a Jini camera service in the OSGi framework. Our OSGi implementation can be further extended to monitor activities of these services. For instance, based on some stored correlation of functionalities in a database, an automated control command may be sent to the other device if one device starts an activity. For example, if a correlation is set such that whenever a camera playback is turned ON, its video stream output is used as an input video stream for playback by the TV after the TV is switched on. As soon as the camera status changes to stop or record, a signal is sent to turn the TV off. The interoperability is achieved by monitoring the two services and then invoking appropriate interface methods based upon correlation stored in a database.

### 3.4 Jini device providing an OSGi service

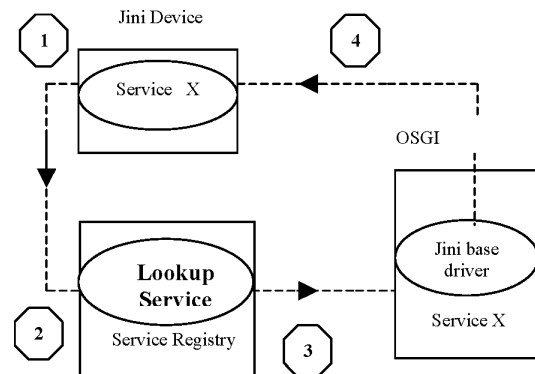
Jini (Sun Microsystems, 2005a), from Sun Microsystems, is a framework based on the idea of federating groups of users and the resources required by those users. The focus of the framework is to make the network a more dynamic entity that better reflects the dynamic nature of the workgroup by enabling the ability to add and delete services flexibly. Jini systems provide mechanisms for service construction, lookup, communication, and use in a distributed system. Jini technology uses a *look-up service* with which devices and services register. When a device joins a Jini network, it goes through a phase called discovery and join-in. The device first locates the *look-up service* (discovery) and then uploads an object that implements all of its services' interfaces (join in). To use a service, a user or a program locates it using the *look-up service*. The service's object is copied from the *look-up service* to the requesting device where it will be used. Integrating Jini Services with OSGi would mean registering the services in the OSGi as OSGi services. This will make the registered Jini services accessible to any service or bundle in OSGi. The OSGi specification works with various devices access standards and is compatible with and can enhance a Jini environment. As mentioned previously, an OSGi implementation can also provide bridging capability between the Jini environment and non-Jini devices, such as UPnP, HAVi, and others.

### 3.5 Registering Jini services as OSGi services

To register Jini services in the OSGi framework, we designed and implemented a Jini base driver in the OSGi environment. The Jini driver is a bundle in the OSGi framework that imports Jini services into OSGi services. To import services, the Jini driver service transforms any Jini service discovered on the network to OSGi services thereby defining a bridge between a Jini network and an OSGi service platform. Importing Jini services enables applications in the OSGi framework to interact with Jini services. Thus, OSGi bundles need not include extra components to use Jini Services and furthermore they do not even have to be aware of the fact that the Service Platform

is Jini enabled. Figure 3 illustrates the steps involved when a device is introduced into the Jini network and registering its services as OSGi services in the OSGi Platform.

**Figure 3** A Jini device joins a Jini network and registers its services with OSGi



After the Jini base driver (bundle) has been installed into the OSGi framework, it waits for a Jini device to join the network. When the *JiniCamera* (a Jini device) is introduced into the network, the Jini base driver detects that a Jini device has joined the network and imports all the information needed to register that service as an OSGi service. The registration of the discovered Jini services is only possible if the interfaces under which they are registered are available to the OSGi framework. The *JiniCamera service* is registered as an OSGi service using a `BundleContext` object which provides a reference to a `ServiceRegistration` object. If the registration is successful, the following properties are set:

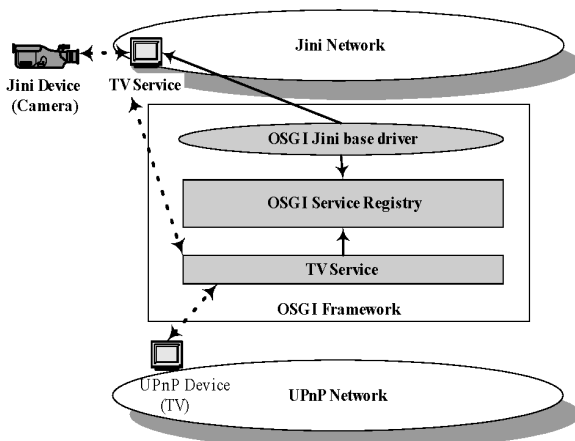
- **DEVICE\_CATEGORY.** The property that must be used in order to participate in the OSGi Device Access mechanisms. The Jini base driver sets the value of this property to Jini when it imports a Jini service.
- **SERVICE\_ID.** A unique Jini service identifier. Each service is required to have a unique service identifier. The Jini base driver sets this property to the identifier of the service in the Service Registry.
  - 1 *Discovery:* Jini device discovers available *LookUp Service(LUS)*.
  - 2 *Join:* Jini device registers its service proxy with the LUS.
  - 3 Registration of the Jini proxy as an OSGi Service.
  - 4 The Jini service of the device is now available as an OSGi service.

After registration, any application that is running on the OSGi framework can access and use the Jini services that are registered as the OSGi services. Applications that use these services do not and need not be aware that these are Jini and not OSGi devices. OSGi bundles need not include extra components to use Jini services.

When a Jini device ((the *JiniCamera* in our implementation (as shown in Figure 4)) is started, it registers its services with the LUS. The implemented

*JiniCamera* service allows the following operations: start, stop, record and playback.

**Figure 4** Integrating Jini and UPnP services through OSGi



The *JiniCamera* service provides an importable service that can be used by any other applications in the Jini network. The *JiniDriver*, a bundle in OSGi, gets access to this service and registers a proxy of this importable Jini Service with the OSGi Service Registry.

### 3.6 Integrating UPnP and Jini services in OSGi

One of the main goals of this paper is to demonstrate, by design and implementation, the seamless interoperability between Jini and UPnP devices and services by exploiting the OSGi framework. To achieve this we implemented an integration bundle we called the *UPnP-Jini bundle*. The UPnP-Jini bundle obtains a reference to both the *JiniCamera* and *UPnP\_TV* by searching the Service Registry using `DEVICE_CATEGORY` as the search keyword and the value Jini or UPnP device. Once objects to *JiniCamera* and *UPnP\_TV* are obtained, the UPnP-Jini bundle uses methods for both devices to pass information and images between the *JiniCamera* and *UPnP\_TV*. Thus in our implementation, the UPnP-Jini bundle acts as a bridge for the Jini and UPnP devices. In this way, both the *JiniCamera* and *UPnP\_TV* have access to the services offered by each other and seamlessly interoperable and communicate with each other via OSGi as depicted in Figure 4.

The major task in our design was the implementation of the base drivers, which conforms to OSGi specifications, and handles the registration of UPnP or Jini services in the OSGi's registry. To test the driver and service implemented, a UPnP and Jini device were implemented. In our prototype implementation, we have the UPnP device (simulating a TV) and the Jini device (simulating a camera) each of which executes separately on a workstation. The OSGi bundles (UPnP, Jini, UPnP-OSGi bundles) were all executed on a third workstation running JES 2.0. All workstations resided on a 100 Mbits/s Ethernet local area network.

### 3.7 Web access to home area network devices using handheld devices

To facilitate access to home area network devices via the OSGi-based gateway anytime, anywhere using small, portable handheld devices such as mobile phones, Personal Digital Assistants (PDAs) (as shown in Figure 1) we exploit the Wireless Application Protocol (WAP). WAP defines the Wireless Markup Language (WML) and WMLScript for mobile applications to display web content by addressing the limitations of small screens in mobile clients. One of the key components of a WAP-based network is the WAP gateway (Ma and Irvine, 2002) which is connected to the web. The function of the WAP gateway is to provide operator specific services or to perform content and protocol translation between Transport Control Protocol/Internet Protocol (TCP/IP) and the WAP stack. The WAP gateway also resolves domain names to IP addresses by performing all required Domain Name Server (DNS) services, and translates web pages into compact encoded formats that reduce the size and number of packets required to be transmitted over the wireless communications network. The result of these operations is to offload many TCP/IP functions from phones, PDAs, enabling these devices to be smaller, less powerful, requiring less memory resulting in lower-cost of producing the devices. User interaction with the WAP gateway is initiated through the WAP browser. The terminal sends its request to the WAP gateway, which then requests the content from the web server.

In the case of mobile internet applications, data transfer speeds are low, and mobile handheld devices are much slower compared to computers connected to fixed, wired networks. Moreover, these devices typically have small screens and require that the web content be displayed differently from the display on typical personal computers. WAP (Aust et al., 2002) is a protocol designed to specifically transmit web-like content to mobile phones and other (usually wireless) devices with low bandwidth access (Saha et al., 2001).

A WAP enabled portable device can access websites similar to the personal computers connected to the internet by sending WAP requests to the WAP gateway. Using WAP, a small, portable device views internet information that is presented in a format to suit the device's screen. Web browsing over the internet uses HyperText Transfer Protocol (HTTP) to transport the web pages and the language that defines the manner by which content is displayed is the HyperText Markup Language (HTML). Thus, to view the HTML pages associated with home area network devices, we need to convert HTML pages to WML pages dynamically. By dynamically converting HTML to WML we eliminate the need for having an HTML version and a WML version of the same web site. Dynamic conversion of HTML to WML can be achieved using the following approaches:

- *WAP gateway (Deri, 2001)*. A plug-in at the WAP gateway can perform the necessary dynamic HTML to WML conversion. The WAP Gateway receives the request (in WML) from the WAP client for a particular HTML page. The gateway contacts the necessary web server for that page using TCP/IP. After obtaining the HTML page, the WAP gateway runs a program (Perl or JavaScript) that converts the HTML to WML on-the-fly and forwards the converted WML file to the WAP client. The WAP gateway approach takes control away from web content providers since they will have to rely on WAP gateways to provide this plug-in in order to make their websites accessible to small, portable, mobile devices. Alternatively, the content provider will have to provide a WML version of their site which increases costs.
- *WAP browser*. A WAP Browser add-in that converts HTML to WML on-the-fly can also be used (Vantroys and Rouillard, 2002). The disadvantage of this solution is that it increases resource demands on the small mobile devices which already lack resources. Thus, such a solution is ineffective.
- *Web server*. Software such as the *mobile converters* could be added to web servers (Christianson et al., 2002) to make the servers capable of distinguishing between requests from WAP and those from regular web browsers. Depending on the source of the request, the web server will either serve a WML or an HTML page. This solution is effective but it requires all web servers to have the plug-in for it to be efficient. With this approach, the WAP gateway and web server functionalities are merged together.

It is worthwhile noting that two common methods of implementing the dynamic conversion of HTML to WML include:

- *Using Perl scripts (Vanderdonckt et al., 2001)*. There are a few Perl scripts available that convert HTML to WML. The Perl script formats a WML page by ignoring tags in HTML that are allowed in WML and replaces those that are not with the corresponding WML tags (for instance, replacing the body tag in HTML with the card tag for WML).
- *Using eXtensible Stylesheet Language Transformation (XSLT)*. A language for transforming one eXtended Markup Language (XML) document into another XML document. XSLT is designed for use as part of eXtensible Stylesheet Language (XSL), which is a style sheet for XML (Forstadius and Loytynoja, 2001).

Since an HTML document is not governed by strict standards and rules, it cannot be directly converted into a WML document. It has to be first parsed into an XHTML or XML document by using a tool such as tidy.exe and then converted into WML using XSLT. Tidy.exe is a HTML tool that was originally written by Dave Raggett of the World

Wide Web Consortium (W3C). It is designed to fix mistakes in HTML, tidy up the layout, assist with web accessibility, and convert HTML to XHTML (Tsybalenko and Munson, 2001).

The various approaches discussed above can be used to convert HTML to WML to enable small portable device users to access and operate home area network devices through the OSGi-based residential gateway. In our implementation, we opted to place the HTML to WML converter at the WAP gateway. We argue that this is the most efficient solution as the converter will use gateway resources and will not place resource demands on the limited resources of small, portable handheld devices. This approach will also require no changes to end-users of WAP-enabled devices and web server administrators.

In our implementation, an HTML page request from the WAP client is converted into a supplied WML deck (a single WML document is known as a *deck*) that is provided by the WAP gateway. The user enters a Uniform Resource Locator (URL) into a form field in a WML page, and sends the request to the WAP gateway. The Perl Script residing at the gateway extracts the desired URL from the query string and fetches the desired web page and converts (using a Perl script) the HTML page into a WML page using string replacements and regular expressions. Other Perl script tasks such as translation of the links (for instance, from body tag to card tag, using the *replace* method), limiting the card size by splitting into several cards (cards in WML represent an element and every deck contains one or more cards), compilation, debugging, and checking the result using validation functions are also performed.

The resulting WML page is then forwarded to the WAP-enabled portable device that will then be able to access (through the WML page) the home area network devices via the OSGi-based residential gateway.

#### **4 Security implementation issues for secure internet access to the home area network devices**

Security is an essential requirement for residential gateways and often involves many different aspects. Security in the OSGi framework is based on the Java 2 security architecture (Gong, 2001). Many methods defined by the OSGi API require the caller to have certain permissions. *Permission* is the authority to access some resource or to perform some operation. The bearer of certain permission is allowed to access a resource or to perform an operation specified in the permission, and the lack of it denies the caller an access or operation. For any service gateway, such as the residential gateway we have designed and implemented, the issue of security is of paramount importance since the gateway will be accessed by service providers and homeowners. In our residential gateway design and implementation, we address the following security issues during remote access to OSGi-based services:

- *Dispatch the right service to the right person.* When enabling internet access to a home area network, security and authentication become an important concern. A basic authentication that is used to verify a user identity can also be used as a means to identify the services to which the user is authorised. An encrypted username/password is used to map to the services to which the user has access rights. Our implementation exploits OSGi-based HTTP authentication. The Java Embedded Server's HTTPService provides the necessary callback (via the `handleSecurity` method) to implement any HTTP-related authentication schemes. The `handleSecurity` method uses the `BasicSchemeHandler` (belonging to `HttpContext` object) which performs the necessary authentication by checking the user's username/password combination.
- *Maintain the integrity of the services.* The integrity of the services is guaranteed by exploiting the three-level security model of the OSGi framework: service (level 1), package (level 2), and admin (level 3) level privileges defined by the `ServicePermission`, `PackagePermission`, `AdminPermission` classes of OSGi (subclasses of `java.security.BasicPermission`) respectively. The service privilege is the very first or basic protection as it permits authentication on a per service level. This privilege permits a user to 'register' a service or to 'get' a service from the service registry. The package privilege represents a permission to import or to export a package. A package name and an action must be specified. This kind of permission grants a user access to all the resources that are contained in that package whether it is a service or a bundle. Similarly, a user can also export a package. The admin privilege is highest permission level and users with this kind of privilege can perform administrative operations in the OSGi-based residential gateway. However, the admin privilege can be subject to abuse in that a bundle with such privilege has full control over the OSGi service platform.

We have implemented all the three types of permissions mentioned above. In our implementation, we exploit the `java.security.Policy` to grant permission to classes loaded using the Java security manager which enforces the permissions that have been granted. It does so by examining the set of permissions owned by a calling class against the required permission. If a class has the needed permission then the execution proceeds; otherwise, access is denied.

- *Protect the services from intruders.* The services and their implementation details are protected from malicious attacks by means of secure sockets and authentication. Even if a user has access to the service it is not possible to modify the service implementation (i.e., code) since package or admin level privileges will still be required.

By implementing the three-level security permission in the OSGi framework we have protected the service and bundles

that reside in the OSGi-based residential gateway. Security and authentication for remote access is provided by the HTTP basic authentication scheme. By combining these security features, we are able to provide secure, internet access to the home area network via the OSGi-based residential gateway.

## 5 Conclusions and future work

In this paper, we describe the design and implementation of an OSGi-based residential gateway that provides seamless communication and interoperability between heterogeneous Jini and UPnP devices. We achieve this by designing and implementing Jini and UPnP base drivers. Our proposed solution enables the capability to bridge disparate discovery technologies and allows a richer device interaction and service delivery. In addition, our OSGi-based gateway also provides secure, remote access to the home area network devices and services.

We will report on a full performance evaluation of our implemented design architecture in the future and we plan to investigate the performance impact of the residential gateway on local and remote internet access to heterogeneous home area network devices exploiting different communication technologies and protocols.

We are also currently working on automation of device actions based on a one-time initialisation by the user. This feature will make it possible to trigger several devices automatically to achieve the desired actions of several devices that the user may want to activate based on some specific set criteria or home user preferences.

## Acknowledgements

We express our sincere gratitude to Nadeem Ansari for his help and efforts on the implementation of the UPnP portion of our proposed architecture. We are grateful to Farhan Siddiqui for her feedback on early drafts of this paper. We thank the anonymous reviewers for their useful comments and suggestions. We would also like to thank Han-Chieh Chao and Yuh-Shyan Chen for their encouragements and great support during the preparation of this manuscript. This work was supported by grants from Sun Microsystems (Palo Alto), Microsoft Corporation (Seattle), and Ixia Corporation (Calabasas).

## References

- Aust, S., Fikouras, N. and Gorg, C. (2002) 'Enabling mobile WAP gateways using mobile IP', *Proceedings of European Wireless 2002*, Florence, Italy, February, <http://docenti.ing.unipi.it/ew2002/proceedings/157.pdf>.
- Bluetooth SIG (2005) *Specification of the Bluetooth System*, Available at <http://www.bluetooth.org/> (Date of access: November 12).

- CES (2005) *Consumer Electronics Association*, Available at [http://www.ce.org/Press/CEA\\_Pubs/816.asp](http://www.ce.org/Press/CEA_Pubs/816.asp) (Date of access: November 12).
- Christianson, L., Brown, K. and Tomar, N. (2002) 'Text reorganization for wireless web devices', *Proceedings of Communications Systems, Networks, and Digital Signal Processing (CSNDSP 2002)*, Staffordshire, England, July.
- Deri, L. (2001) 'Beyond the web: mobile WAP-based management', *Journal of Network and Systems Management*, Vol. 9, No. 1, March, pp.15–29.
- Dobrev, P., Famolari, D., Kurzke, C. and Miller, B. (2002) 'Device and service discovery in home networks with OSGi', *IEEE Communications Magazine*, Vol. 40, No. 8, August, pp.86–92.
- Forstadius, J. and Loytynoja, M. (2001) 'XML in dynamic multimedia content management', *Proceedings of 2001 Nordic Interactive Conference*, Copenhagen, Denmark, Also available at: <http://www.mediateam oulu.fi/publications/pdf/80.pdf>.
- Gong, L. (2001) 'A software architecture for open service gateways', *IEEE Internet Computing*, Vol. 5, No. 1, January–February, pp.64–70.
- Konno, S. (2005) *CyberLink for Java*, Available at <http://www.cybergarage.org/net/upnp/java/> (Date of access: November 12).
- Ma, I. and Irvine, J. (2002) 'Characteristics of WAP traffic', *Proceedings of European Wireless 2002*, Florence, Italy, February, <http://docenti.ing.unipi.it/ew2002/proceedings/168.pdf>.
- Marples, D. and Kriens, P. (2001) 'The open services gateway initiative: an introductory overview', *IEEE Communications*, Vol. 39, No. 12, December, pp.110–114.
- OSGi Forum (2005) *OSGi Service-Platform, Release 3*, Available at <http://www.osgi.org>, (Date of access: November 12).
- Rose, B. (2001) 'Home networks: a standards perspective', *IEEE Communications Magazine*, Vol. 39, No. 12, December, pp.78–85.
- Saha, S., Jamtgaard, M. and Villasenor, J. (2001) 'Bringing the wireless internet to mobile devices', *IEEE Computer*, Vol. 34, No. 6, June, pp.54–58.
- Saif, U., Gordon, D. and Greaves, D. (2001) 'Internet access to a home area network', *IEEE Internet Computing*, January–February.
- Salutation Consortium (1999) *Salutation Architecture Specification Version 2.0, C – Part 1*, June.
- Sun Microsystems Inc. (2005a) *Jini Connection Technology*, Available at <http://www.sun.com/software/jini/> (Date of access: November 12).
- Sun Microsystems Inc. (2005b) *Service Location Protocol*, Available at <http://www.openslp.org/> (Date of access: November 12). Also RFC 2608, *Internet Engineering Task Force (IETF)*, June 1999.
- Tsybalyenko, Y. and Munson, E.V. (2001) 'Using HTML metadata to find relevant images on the World Wide Web', *Proceedings of Internet Computing*, Las Vegas, USA, June, pp.842–848.
- UPnP Forum (2005a) *Universal Plug and Play Device Architecture*, Available at <http://www.upnp.org/> (Date of access: November 12).
- UPnP Forum (2005b) *UPnP Device Architecture*, Available at <http://www.upnp.org/download/UPnPDA1020000613.htm> (Date of access: November 12).
- Vanderdonckt, J., Bouillon, L. and Souchon, N. (2001) 'Flexible reverse engineering of web pages with VAQUISTA', *Proceedings of IEEE 8th Working Conference on Reverse Engineering*, Stuttgart, October, Los Alamitos.
- Vantroys, T. and Rouillard, J. (2002) 'Workflow and mobile devices in open distance learning', *Proceedings of IEEE International Conference on Advanced Learning Technologies (ICALT 2002)*, September, Kazan, Russia.
- Wils, A., Matthijs, F., Berbers, Y., Holvoet, T. and DeVlaminck, K. (2002) 'Device discovery via residential gateways', *IEEE Transactions on Consumer Electronics*, Vol. 48, No. 3, August, pp.478–483.