A Framework for Interoperability in Home Networks

by

Dinesh Bhat

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Master of Science

Department of Computer Science

Faculty of Graduate Studies

University of Manitoba

May 2006

THE UNIVERSITY OF MANITOBA

FACULTY OF GRADUATE STUDIES

*****

COPYRIGHT PERMISSION

A Framework for Interoperability in Home Networks

BY

Dinesh Bhat

A Thesis/Practicum submitted to the Faculty of Graduate Studies of The University of Manitoba in partial fulfillment of the requirement of the degree

OF

MASTER OF SCIENCE

ABSTRACT

Recently, there has been an increase in the development in technologies related to Home Networks exploiting the untapped research potential in the home environment. This surge of research in Home Networks has led to the invention of many, capable smart devices that can be used to automate the functioning of homes. A modern home contains many sophisticated devices and technologies compliant with different protocols. Automation in such heterogeneous environments cannot be effectively achieved without the support of a middleware layer. The key element for success in home networks is interoperability among the heterogeneous devices operating in the network. Ideally, all the devices that participate in the network would run the same standard protocols for device discovery, configurations, and control. However, there is no standard that has been accepted by the major industry players and a de-facto standard has been slow to arise. This lack of a unified standard has resulted in a large number of technologies including UPnP, Jini, and HAVi that satisfy the requirements of a small, confined Home Network with homogeneous devices.

The objective of this thesis is to develop a framework based on OSGi that can integrate popular home network standards such as UPnP and Jini thereby achieving interoperability within homes. My framework extends OSGi specifications to integrate UPnP and Jini. This framework has been designed with the home network scenario in mind. A Proof-of-concept prototype has been implemented which also demonstrates the integration of low-powered devices such as Pocket PCs and embedded network processors such as TINI devices.

DEDICATION

I dedicate my master's degree to my brother Mahesh; my inspiration, my guide and my role-model. I would also wish to thank my parents and family for their support. It would not have been possible without you all.

ACKNOWLEDGMENTS

I would like to acknowledge the kind help that I received from Dr. Rasit Eskicioglu throughout my program. I also wish to thank TRLabs for giving me an opportunity to pursue my research in their facility. I would like to thank Dr. Jeff Diamond for accepting to be on my committee as the external examiner and I would like to thank Dr. Peter Graham for agreeing to be the internal examiner and for all the useful brainstorming sessions that we had. My sincere thanks to Hossein Pourreza for all the help and guidance during my thesis. I have learned a lot from him. Finally, I would like to acknowledge the help that I received from Dr. Neil Arnason during my thesis proposal submission.

List of Tables

List of Figures

Contents

Chapter 1

Introduction

Home networks are the result of advances in digital media coupled with advances in the network capabilities of home devices. In recent years, the world has witnessed many advances in digital media, computers, consumer electronics and communication networks. An average household contains many complex digital devices like DVDs, CD players, and VCRs which can be integrated to form a home network. The deployment of home networks promises high speed Internet coupled with rich media content experience. Embedded systems have also become a common part of our daily life. However, the multiple specifications on which embedded home devices are built are different from each other. In a ubiquitous computing environment [9], where devices communicate with other devices on the network, we need standard middleware specifications to make the devices interoperate. Devices compliant with one common specification would ease the process of communication among them in a home network environment. Standard interfaces like the Java AWT or the Java runtime environments like the Java Virtual Machine (JVM) that are used by present systems alone are not sufficient to allow all devices to communicate [34]. Home automation applications tend

to be complex involving variety of devices potentially using different standards. Moreover, such applications should also consider the existence of resource-constrained devices such as PDAs and other mobile devices. For the combination of mobile devices and other devices to participate in a home network, users need a standard with an interface that handles interaction among wired and wireless devices accurately and efficiently[1].

Figure 1.1 shows a typical home network environment comprising many devices such as controlled lights, self-monitoring refrigerator and television. These devices can not only perform the basic functions, but they also can intelligently communicate with each other to coordinate the execution of automated applications such as turning lights on or off at appropriate times, etc.

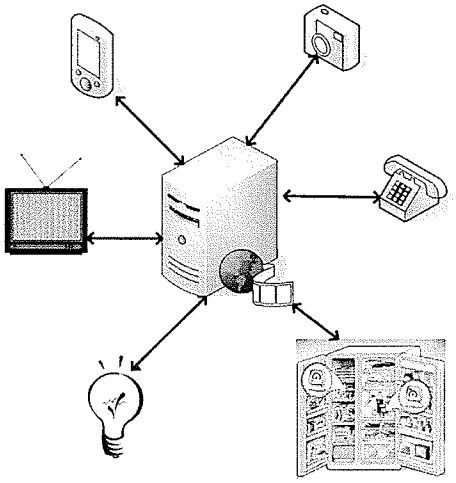

Figure 1.1: A Typical Home Network Architecture

[1]In this thesis, the terms devices and services are used interchangeably.

1.1 Assumed Home Environment

Typically, a home network consists of several devices and appliances networked with one another using different connection technologies (wired or wireless) and protocols. A networked device is considered to be a consumer device with embedded processor and network connection. These devices can configure themselves and discover other devices within the network with which to share their services. In the assumed environment, we envision a home gateway device capable of making the home devices that are built with different specifications interoperate. In my research, the home gateway device is an embedded computer that can detect the presence of devices and configure them. I assume the home gateway device to be a relatively powerful device with adequate storage and processing ability. In the future, the home gateway device may reside either on an embedded device or on, for example, a DSL modem to provide the home owners with automated control of their home network and the devices on it.

1.2 Example Scenario

Inter-communication among multiple devices within homes can be better illustrated with the help of scenarios. The following scenario demonstrates the capabilities of devices that are being used in a home to minimize necessary human intervention.

It is now 6 AM and Tom is awakened by the alarm clock that turns off the air conditioner and turns on the lights. He is getting ready to go to work and his television turns on and gives him the updated traffic news on the Trans Canada Highway which he normally takes to work. He steps out of the house and the sensors turn off all the lights and turn the security camera and alarm on.

During lunch, Tom wants to inspect his house remotely using his PDA. The PDA connects to the security camera rendering the video and displays all the entrances. After lunch, his wrist watch displays the name of the medicine he should take for his high blood pressure.

He has to send the readings to his doctor. He opens the sphygmomanometer and checks his blood pressure. After the reading, the sphygmomanometer automatically sends the results to his doctor.

It is now 4 PM and Tom is heading back home. His PDA buzzes and displays a file with the high blood pressure result analysis sent by his doctor. When Tom gets back home, the sensors turn on the lights and the air conditioner. He is now near the audio system and his home phone plays the unread voice messages on his audio system. At 6 PM, the television turns on and plays his favorite channel. Further, Tom has set his favorite channels on the television and if he is not available to watch any of his preferred channels at his selected times, the television records all the programs onto the VCR and informs when he gets back home about the recording done.

This scenario motivates the necessity for devices to communicate among themselves to coordinate particular tasks to automate the functioning of homes. These homes have a variety of devices compliant with a wide variety of protocols. Thus, we require an all-in-one standard that can understand a variety of home device protocols to support this communication.

An interoperability standard for home networks must satisfy the following requirements [54]-

- Physical connectivity between the devices in the home network
- Standard framework for device discovery, configuration and control
- Common formats and streaming protocols for media
- Standard protocols for media management and control
- Compatible authentication and security mechanisms for users and devices
- Common user interfaces for devices

Ideally, the devices in the network must be intelligent enough to discover each other without having to be manually configured. However, there are also many mobile devices that may

periodically be a part of the network. This means that interoperability standards must be able to discover the devices and the resources they offer dynamically. Currently, there are many such interoperability standards including Universal Plug and Play (UPnP) [49], Jini [41], and Home Audio Video Interoperability (HAVi) [12]. However, these standards alone are not capable of communicating between devices built with different standards. Moreover, each are accepted by only a few member industrial partners. Thus, we need a common platform for providing overall interopearability.

The Open Services Gateway Initiative (OSGi) is a technology that works above the UPnP, Jini and HAVi standards. OSGi is gaining momentum because of its capability to integrate multiple standards including UPnP and Jini [30]. OSGi is a set of specifications achieved with the collaboration of various industry partners that have agreed upon a common platform. However, direct communication between UPnP and Jini through OSGi is not possible with the current specifications of OSGi. OSGi can communicate with UPnP and UPnP can in turn communicate with OSGi. Similarly, OSGi can communicate with Jini and Jini can communicate with OSGi. However, communication between UPnP and Jini through OSGi is not feasible with the release 3 specifications of OSGi [29].

Generally, an interoperability framework must also satisfy several other requirements. With advances in home network research, more and more interoperability standards are arising. Thus design of an interoperability framework must also make integration of future standards easier. Further, an interoperability framework must be self-configuring in integrating a variety of devices and standards. A framework must also be reliable in providing QoS guarantees. Since an interoperability framework may also communicate via the Internet, security is one of the main issues that should be addressed effectively. Considering the above constraints and requirements, this thesis focuses on integrating home network devices and allowing them to communicate with each other effectively to automate the functioning of the homes. However, the additional above-mentioned requirements such as extendibility, self-manageability, security and QoS are not addressed in my research.

A prototype implementation of interoperability framework is a part of a bigger home

technologies project at TRLabs. The core of the research involves developing an interoperability framework to integrate UPnP and Jini using an OSGi-based home gateway. Our OSGi gateway is compliant with the release 3 specifications of OSGi. This thesis contributes by developing a UPnPLUS bundle for OSGi to integrate UPnP with Jini and a Jini-to-UPnP Translator bundle to integrate Jini with UPnP. I have also ported my framework to a Pocket PC to demonstrate the capability of running on a resource-constrained device since future home gateways will likely run on an embedded device.

1.3 Thesis Organization

The remainder of this section is organized as follows. Chapter 2 reviews the work related to this thesis which ranges from low-level cabling issues to home automation projects. Chapter 3 explains the characteristics of the interoperability layer in home networks needed to integrate a wide variety of devices and motivation for my thesis. In Chapter 4, my design and implementation are explained in detail. Chapter 5 discusses evaluation strategies used to assess my framework. Finally, Chapter 6 provides conclusions and discusses areas for possible future work.

Chapter 2

Related Work

Designing a framework for interoperability in a home network involves hardware and software based interoperability standards. Working from the bottom up, I have divided my related work into four distinct parts. In the beginning, I review the low-level, hardware based home network standards including HomeRF, X10, HomePNA and HomePlug followed by software based interoperability standards for home networks. Further, I explain the current research towards designing home network architectures and frameworks, followed by middleware systems built for home networks. Finally, I discuss current academic and industrial projects in building home automation.

2.1 Hardware/Wiring Standards for Home Networks

There are a number of low-level hardware-based standards for home network connectivity including HomeRF, X10, HomePNA and HomePlug that can be used in homes to automate

device inter-communication and sharing of resources.

2.1.1 HomeRF

The HomeRF Working Group [15] was formed to create a specification standard for interoperable home devices by establishing an open industry specification for wireless digital communication between computers and home devices anywhere in a home environment. HomeRF was created to enable wireless interoperability by using the Shared Wireless Access Protocol (SWAP). SWAP is a specification for wireless voice and data networking in the home. The limited roaming required in home network environments enabled this new technology.

A SWAP based network can operate as an ad-hoc Peer-to-Peer network or as client server network that provides data networking or as a managed network under the control of a connection point. A SWAP network can consist of up to three types of devices.

- Control Points
- Isochronous Voice Devices
- Asynchronous Data Devices

Using a contention-based protocol such as TCP/IP, the connection point can perform data transfers among data devices. SWAP acts as a client server protocol between the control point and the voice devices but is Peer-to-Peer between the control point and data devices. The MAC layer is designed to carry both voice and data traffic and to interoperate with the Public Switched Telephone Network (PSTN). The MAC Layer provides very high data throughput and data security. It also supports high quality voice traffic by using the TDMA access mechanism [15].

2.1.2 X10 Protocol

X10 is a communication protocol [50] that utilizes the electrical wiring in houses as the medium to identify and control electrical appliances in the home. Each of the devices will have its own specific address and responds when addressed using the address. To control specific devices, X10 allows 256 addresses on a home power system. The address consists of module number and house code. When you wish to turn on an X10-controlled lamp, you have to tell the Lamp Module controlling that lamp to turn on. The base station is a simple controller that is programmed to transmit the commands issued.

There are several X10-based devices on the market that can be used to automate home functions. Since X10 makes use of the default electrical wiring, configuration is easier, not involving complex instruments and applications. However, X10 suffers from signal attenuation, noise, and sometimes may cause spurious on or off signals. Some X10 controllers may not work with low powered devices and X10 can transmit only one command at a time.

2.1.3 HomePNA

The Home Phone Networking Alliance (HomePNA) [4] is the largest alliance promoting home networking over existing telephone cabling. The initial version of HomePNA had a data rate of 1Mbps and the recent upgrade of the earlier version provides bandwidth up to 10Mbps. These standards are designed to work with the wide range of protocols found in the telephone network. Phone calls, cable modem and the home devices can all simultaneously work over the existing phone lines. The third version of HomePNA offers 128Mbps of bandwidth with an option to extend it up to 240 Mbps. Quality of service has also been included in the third version. As the only home networking industry specification capable of reaching above 100 Mbps and with inherent deterministic Quality of Service (QoS), HomePNA technology provides an excellent high speed backbone for a home multimedia network that requires a fast and reliable channel to distribute multiple, feature-rich digital audio and video applications throughout the home network.

While QoS parameter in HomePNA 2.0 enables equipment manufacturers to prioritize telephone voice data over computer data, HomePNA 3.0 greatly enhances the version 2.0 capabilities by incorporating deterministic QoS support for real-time data. This technology allows users to assign specific time slots for each stream of data guaranteeing that the real-time data will be delivered when it is required with predetermined latency and without interruption. It also allows HomePNA to transport data with inherent IEEE1394 [18] QoS requirements.

2.1.4 HomePlug Powerline Alliance

HomePlug is a standard for powerline communication where power distribution wires are used for data transfer. It is a technology supported by several leading companies including Intel and Sony that helps devices connected with each other through power lines to communicate. HomePlug devices must be directly connected to wall outlets without using surge protectors or extension cords. HomePlug strictly enforces compliance and provides a certification program to make all hardware manufacturers producing HomePlug devices adhere to the standard specifications thereby ensuring interoperability among different HomePlug devices manufactured by member organizations. The current HomePlug specification allows for speeds up to 13.78 Mbps suitable for High Definition (HD) TV and VOIP [14].

HomePlug AV

HomePlug AV is the next generation of powerline technology designed to support entertainment applications, such as HDTV and Home Theater [13]. HomePlug AV provides a cost effective method of distributing HDTV in the home without new wires. The objectives for the HomePlug AV specifications include providing solution for HD quality video distribution, with secure connectivity and built-in Quality of Service (QoS). HomePlug AV can co-exist with HomePlug 1.0.

2.2 Interoperability Technologies for Home Networks

An interoperability standard should be self-configuring and support a wide range of devices and industry sponsors. There have been many efforts in the past to define interoperability technologies applicable to home networks. In this section, some of the most popular interoperability standards are explained.

2.2.1 Universal Plug and Play (UPnP)

The UPnP device architecture specification (UPnP) defines interoperability standards that enable self-configuring devices to create an ad-hoc, self-discovering system of interoperable network devices [49]. This specification defines mechanisms for automatic address configuration, device discovery as well as service discovery, control, and event handling. Figure 2.1 shows the UPnP protocol stack. It makes use of both UDP and TCP/IP for multicast and unicast messaging.

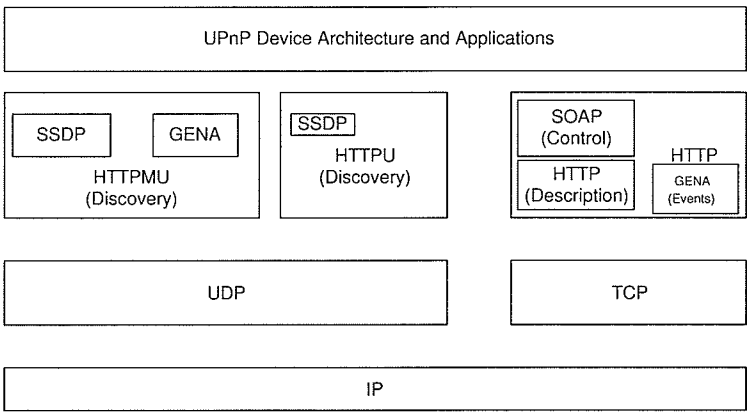

Figure 2.1: The UPnP Protocol Stack

UPnP Specifications

The protocol architecture shown in Figure 2.1 describes the protocol structure used in UPnP. UPnP technology uses existing protocols and technologies such as TCP/IP, HTTP, Simple Service Discovery Protocol (SSDP) [53], Simple Object Access Protocol (SOAP) [37], General Event Notification Architecture (GENA) [49], and XML [47]. These protocols define the communication infrastructure of the UPnP architecture. Although the UPnP architecture consists of a peer-to-peer network, nodes on the network communicate with each other in a client-server manner. Clients are called Control Points (CP) and typically provide a User Interface (UI) for end-users. Servers are called Controlled Devices (CDs) and they define a set of functions called services.

In the UPnP architecture, the CPs call the services when invoked and devices respond to provide the services that are necessary. Within the UPnP architecture, device functionality is invoked using a set of services, each of which corresponds to one or more of the devices in the network. Each device defines a set of functions that are associated with the services and these functions allow CPs to obtain the most active state of the devices and to control the operation of the devices. To enable interoperability among the devices that participate in the network, developers of UPnP constructed a set of device and service definitions which can be used to model various common devices. This is the abstraction state of the devices in the network. These abstract data types hide the underlying complexity of the devices in the network. Since the behavior of these device and service functions are well defined, a CP can interact with any device that implements the supported services. In this manner, CPs and devices can be built independently by different hardware vendors so that they interoperate according to the functionalities defined by the corresponding UPnP device service functions.

All the devices in the UPnP standard need to have an IP address. To obtain an IP address for each of the devices, a DHCP [7] client must be installed on each of the devices and a DHCP server must provide an IP address as soon as a new device enters the network. If a DHCP server is not installed, then the AutoIP [49] protocol can be made use of to provide IP addresses to the devices that enter the network. AutoIP is a protocol that uses the Address Resolution Protocol (ARP) to provide an unused IP address in the network to

new devices. However, one of the major drawbacks of the AutoIP protocol is that the IP addresses can be used only in the same network. Since data packets cannot cross routers making Internet access impossible.

Once a UPnP device gets its IP address, the discovery and advertisement of the services it can offer begins. The services advertise themselves at a known port and the CPs listening on that port start the discovery process. Both the service advertisements and the discovery are implemented by the Simple Service Discovery Protocol (SSDP) [49]. When services are available, they broadcast their availability in the network by sending an initialization message. To discontinue their presence in the network, a UPnP device sends a termination message to the CPs.

In UPnP, services have three attributes associated with them: service type, service name and service location. These attributes are vital for querying device information. During querying, the CPs multicast a UDP message with the type of the service needed. Services that match the request reply with a unicast message. UPnP does not have any centralized servers that handle discovery requests in the network. The replies from the services consist of a URL, which points to an XML document. These XML documents describe the services. This description of the services contains specific information about them, such as vendor information and IP addresses of devices. The XML document also contains a list of various commands that can be performed on the services and the variable(s) associated with the service. For example, a camera in the UPnP network may be represented as devCamera (autofocus). Here, the devCamera represents the abstract data type of the device and a variable associated with that device is autofocus. Queries for cameras may contain the values of the state variables. The state variable autofocus can have the values, autofocus = on/off. Once the Control Point has obtained the necessary XML document, it can issue commands to the corresponding services. These messages are then delivered using the SOAP protocol.

The commands issued by CPs may change the values of the state variables that are associated with the services. These services publish the values of the variables when they

change. To advertise new values, the services send messages to the CPs that are subscribed to those services. These messages are expressed in XML and formatted using GENA. The final step in UPnP discovery is presentation. From the XML document, the control point can retrieve the address of a presentation URL. This URL identifies the presentation web page for the service which can be fetched by the CPs. The CP uses a browser to display the web page at this URL.

Advantages of using UPnP are multiple. All the devices are IP based. Since, most devices, at present, come with built-in IP capabilities, it is easier to incorporate these devices readily into the network. UPnP was also designed in such a way that the upcoming IPv6 [20] can be easily integrated into the technology without further modification in the design. Further, in UPnP, no code has to be moved around or downloaded unlike many other standards. However, UPnP suffers from lack of security and other issues. Since legacy devices may not have IP capabilities, we also need a different standard architecture running in all the devices or an architecture that can integrate the legacy devices. The main drawback of UPnP is that it does not allow vendor developed media formats and protocols. UPnP also suffers from remotely exploitable buffer overflow, denial of service, and distributed denial of service attacks.

2.2.2 Jini

Jini [41] was invented at Sun Microsystems for use in pervasive computing environments. Jini is based on Java and uses Remote Method Invocation (RMI) for establishing communication among the services and the clients. The Jini architecture consists of three components: service providers, clients and lookup services. Available services on the network are defined by the service providers, the clients are the end-users of the services, and the lookup services provide centralized repositories of information about the different services registered by the service providers. Jini requires reliable and stream-oriented communication. Stream-oriented communication means that Jini devices establish a unicast connection with the LookUp

Server using UDP instead of multicasting the services to the other devices within the network. The discovery protocol in Jini uses multicasting with TCP and UDP where the LookUp Server listens on a standard port and the clients then register their services with the LookUp Servers. Since all the services and clients are written in Java, Jini imposes a requirement that all the devices must be able to support the Java Virtual Machine (JVM) [28]. However, the "weaker" devices that do not have the capabilities to run the JVM can also be supported using the surrogate architecture of Jini [41]. Service providers and the clients communicate with each other using proxies that establish the connection for service invocation. Mobility of code is the key concept in Jini. There is no direct communication between the clients and the service providers in Jini instead proxy code that makes RMI calls for services is distributed to clients.

The Jini architecture is built on a concept of a federation between a group of service providers, clients and the lookup services. Jini uses three protocols for service discovery and communication between the services and the clients: discovery, join and lookup protocols. The discovery protocol is used by the service providers and clients to find the lookup services. The join and lookup protocols are not network protocols. They define specific rules that the Jini services and clients must follow to join a Jini federation. Service providers and the clients multicast a request on the network. The lookup services respond to the proxy of the lookup service. The service providers and the clients use this proxy for further communication with the lookup service.

The discovery protocol which permits the service providers and clients to discover the lookup services makes use of three protocols: the Multicast Request Protocol, the Announce Request Protocol and the Unicast Discovery Protocol. The lookup services multicast a message at regular intervals announcing their presence in the network using the Announce Request Protocol. Service providers and the clients looking for the lookup services listen to these messages and request the lookup service for the service. The service providers or clients then unicast a message to the lookup service and receive the proxy code in return. This proxy code is used by the service providers to register their services and by the clients

to communicate with the lookup service using Java RMI to search for other services. Jini also supports distributed events.

Service providers in Jini can be found in many ways. They can be obtained using a Jini browser or they can be attached to applications as a device drive. To run a Jini service or client, a device must implement TCP/IP and UDP with DHCP or any other IP providing server. Both the client and the server must be able to run Java with the Jini extension installed. Figure 2.2 shows the architecture of Jini and the different components in the Jini architecture.

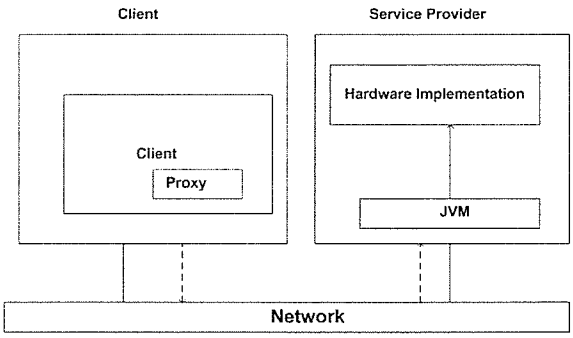

Figure 2.2: Jini Standard Architecture

The primary difference between Jini and UPnP is that UPnP focuses on interaction between the devices in environments using network protocols whereas Jini focuses on objects that move in a network to connect different devices together easily. Jini was also developed with a set of specifications to allow devices that cannot execute the JVM to work with the Jini environment. This allows legacy devices to actively participate in a Jini enabled network. Though Jini offers flexibility in integrating legacy devices, it has to define function specific interfaces for each of the devices. When a new device is brought into a home network environment, these interfaces will have to be developed to use the different services offered by the device.

Use of Jini to automate homes offers several advantages. Jini has a simple lookup service mechanism to discover, and allow devices to share services on the network. Jini does not

require configuration of the devices or installing drivers for new devices. Jini offers mobility of code which can be invoked by the requesting clients instead of communicating with the service providers directly. This is useful since low-end devices offering the services do not have to communicate with the clients directly thereby reducing the overhead on these low-end devices.

2.2.3 Home Audio Video Interoperability (HAVi)

HAVi is a new architecture for developing applications that run on networked home entertainment devices. HAVi is essentially a distributed programming environment that provides mechanisms allowing applications to communicate with each other using IEEE 1394, firewire technology [18]. HAVi specifies a set of system services that form a foundation for building distributed applications. System services include messaging, events, device discovery, lookup functionality, and configuration of streaming connections.

HAVi was created by major industry players including Sony, Hitachi, and Toshiba, who agreed on a unifying architecture to facilitate interoperability among devices in a home network. The focus of HAVi is on audio and video transmission in home networks. The HAVi standard defines a set of APIs that allow devices to configure and integrate themselves into a home network. The implementation of HAVi, however, is left to the manufactures as it does not define how the different APIs should be implemented.

The HAVi architecture includes Controller Devices and Controlled Devices. The Controller Devices control other devices in the network which are called Controlled Devices. The Controller Devices control the other devices using a proxy termed the Device Control Module (DCM) of the Controlled Device. The DCMs are platform dependent. By exploiting Java, HAVi restricts these devices to be those that can run the JVM. However, interoperability between devices that support the JVM and ones that do not can be achieved by using a JVM installed device to provide proxy service to the non-JVM devices. The services that are available in the network are called Software Elements. HAVi has its own software elements

that the HAVi enabled devices can implement including a 1394 Communication Media Manager (CMM), Messaging System (MS), Registry, Event Manager, Stream Manager, Resource Manager, Device Control Module Manager, Self Describing Device (SDD) Data, and Java Runtime Environment (JRE) as shown in Figure 2.3. User interfaces to provide control of the services are called havelets in HAVi. HAVi applications can have many user interfaces deployed.

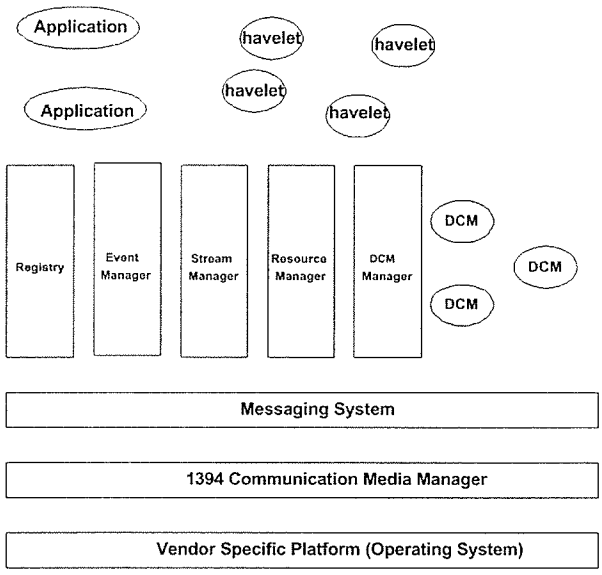

Figure 2.3: HAVi Standard Architecture

The communication Media Manager in the HAVi architecture deals with the physical layer of the network. The Messaging System manages communication among the devices. The registry maintains a database of device information. When a new device enters the network, information about it is registered in the registry. The Event Manager handles the events which relate to communication among the various devices in the network. The DCM Manager installs the DCMs onto the devices to control them.

HAVi devices are of four types.

- *Full AV Devices (FAV)*— Devices in this category actually implement all HAVi software

elements and are responsible for running the JVMs.

- *Intermediate AV Devices(IAV)*— These devices do not have a JVM running on them. Their purpose is to control other devices in the network.
- *Basic AV Devices (BAV)*— Devices in this category are always Controlled Devices.
- *Legacy AV Devices (LAV)*— These devices do not have a JVM installed on them. Since these devices are not HAVi enabled, HAVi commands must be translated into legacy commands for them.

HAVi provides a lot of features to automate audio/visual devices within homes by allowing entertainment products from different manufacturers to communicate with each other when connected to a HAVi network. HAVi compliant devices automatically announce their presence and capabilities to every other device on the HAVi network, greatly simplifying installation and setup. All the devices that come into the network environment are detected and registered. Upgrading functionalities can be easily done by downloading the functionalities from Internet.

Unfortunately, HAVi also has some disadvantages. Jini is targeted more generally, for computer accessories including printers and scanners whereas HAVi is more focused on the home entertainment arena. HAVi also works only on IEEE 1394 firewire networks. While this provides high performance, not all devices support firewire.

2.2.4 Salutation

Salutation is a network independent interoperability technology and thus can be run in different network environments including TCP/IP and IrDA network. Unlike Jini, Salutation is not tied to any programming language. Salutation is thus operating system and platform independent. The protocol stack for Salutation is shown in Figure 2.4. It also shows the different layers in the architecture of Salutation. Salutation Managers act as intermediate components between clients and devices assisting in managing services offered by salutation.

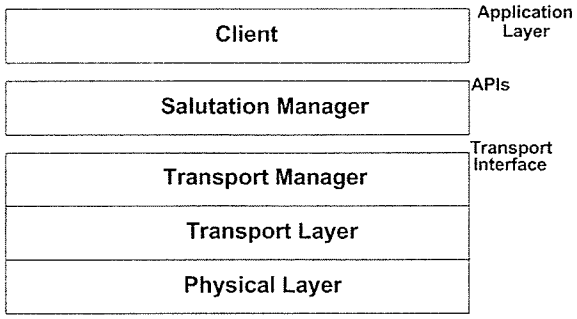

Figure 2.4: Salutation Protocol Stack

All devices talk to the local Salutation Manager which can either reside on the same device or on a remote device. Salutation Managers coordinate with one another and exchange information. They are the agents for the devices in the network. All data transmissions, compatibility in media formats and protocols are handled by the Salutation Managers.

When a new device enters the network, it is registered automatically by the nearest Salutation Manager. The local Salutation Manager then exchanges the registration information with any other Salutation Managers. A service description in Salutation is a collection of functional units representing a different feature. For example, each unit may represent some essential features such as fax and print. Service description records can be queried and matched against a request for existing services during the service discovery process. A discovery request is routed to the local Salutation Manager which in turn directs the request to other Salutation Managers if needed. Inter-Salutation Manager communication is implemented using Remote Procedural Call (RPC). Salutation defines APIs for the clients to invoke services and gather the results.

Communication among Salutation clients and services can happen in many ways. The clients and the services can use the native supported protocols when they are in "native mode" to establish the connection without the use of the Salutation Managers. In "emulated mode", the Salutation Managers deliver the messages to other components. In "salutation mode", the Salutation Managers, not only carry the data, but also the data formats associated with the data. All clients which would like to use a certain service, say fax, must use

the data structure for fax.

Salutation does not address self configuration of devices. In the case of networks with IP and non-IP based devices, Salutation is not flexible because it cannot configure and make the devices belonging to different networks communicate. However, when used in IP environments, network protocols like AutoIP or DHCP can be used to incorporate the devices into the network. Salutation allows user authentication using a user-id and password scheme. Salutation does not require proxies since a client establishes direct communication with devices.

2.2.5 Obje

The Obje framework [51] can be implemented in any digital device, and because the framework can easily adapt to new functionality, it can support virtually any protocol or media format as long as the underlying hardware has enough computing resources to support their execution. The Obje framework requires the ability to execute mobile code similar to the JVM. However, the mobile code need not necessarily be in each and every device. Instead, the mobile code can be stored in a centralized device and this device can provide proxy services to the requesting clients.

To make this technology interoperable with legacy devices, the Obje architecture allows computationally rich devices to act as proxies for the legacy devices. Therefore, in a typical scenario, a desktop computer might act as proxy for other resource-limited devices, such as speakers, digital cameras, printers, scanners, and so on. This allows the Obje platform to be deployed in networks in which only one device is computationally capable of executing mobile code.

The key difference between the Obje technology and other interoperability standards is that Obje does not require specific APIs to be already developed. Other technologies, including UPnP, Jini, HAVi and Bluetooth, each define specialized interfaces for each new type of device: Bluetooth defines profiles for phones, headsets and printers; Jini defines APIs

for devices including printers and cameras; UPnP defines interfaces for audio/visual devices, scanners and home appliances; HAVi defines APIs for DVD and CD players, and TVs. With Obje, services and applications are written once, against a small, fixed set of Obje meta-interfaces, which allow them to acquire any needed communication and control capabilities dynamically. Obje meta-interfaces provide support for the most basic functions that provide a complete context with which devices can interoperate even if they have no prior knowledge of each other. For example, when two Obje enabled devices want to communicate with each other, an Obje meta-interface establishes the connection based on the network capabilities of the devices and handles the data transfer. These Obje meta-interfaces acquire the protocols and communication standards used by specific devices so that the communication among the devices can be easily achieved with no reprogramming of existing services. This allows users and manufacturers to combine devices and services instead of creating a bridge to combine different standards.

The architecture of Obje is shown in Figure 2.5. Meta interfaces and the Mobile execution code are the components that deal with dynamically creating services without having to explicitly program them. Unfortunately, Obje does not provide the details of dynamically creating new services. All Obje devices or services, which are called "components", implement and make use of one or more of the meta-interfaces. Applications are developed using these meta-interfaces.

2.2.6 Service Location Protocol (SLP)

The Service Location Protocol (SLP) is an Internet Engineering Task Force (IETF) standard for service location [52]. SLP requires TCP/IP, multicast and UDP/IP support for the network to be formed.

The SLP architecture shown in Figure 2.6, consists of three entities: service agents, user agents and a directory agent. The service agents represent the services that are available within the network. User agents represent the clients that request the services and the

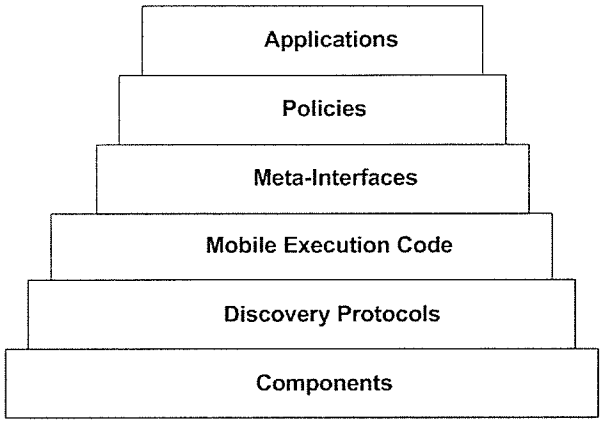

Figure 2.5: Obje Architecture

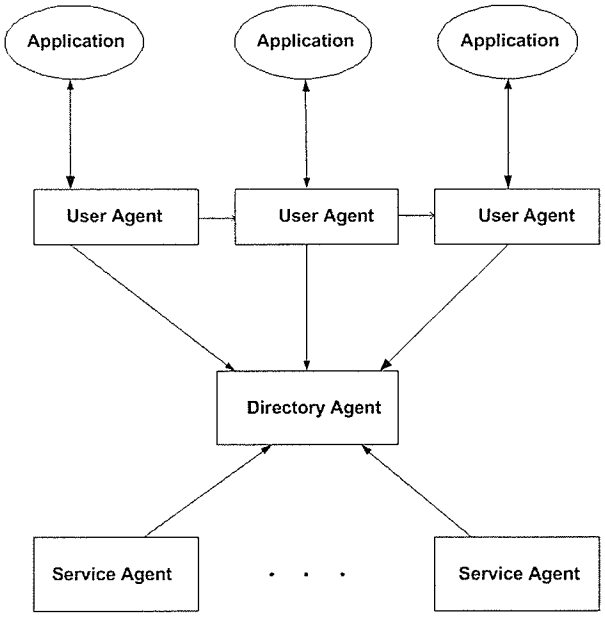

Figure 2.6: Service Location Protocol Architecture

directory agent is the centralized storage host which stores the services that are registered. Services in SLP are similar to services in any other interoperability standards. The user agents look for services on behalf of their applications using the directory agent.

If a directory agent is available in the network, then the all the service agents have to register their services in the directory agent. Multicast messages are not allowed to pass through the gateway to avoid flooding the network with messages. Service discovery messages also have a constraint which specifies how many hops a particular message is allowed to take.

SLP can also work without the directory agent. If a directory agent is not available in the network, then the user agents simply broadcast requests in the network. The service agents respond with URLs that contain the IP addresses of the services that match the request. In "passive discovery" mode, the directory agent broadcasts the available services and the user agents that require the services unicast a message with the request. In the "active mode" of service discovery, the user agents multicast a request message for services and the directory agent checks if the services are available. If the services are available, then the directory agent replies to the user agent using a unicast message.

The SLP protocol makes use of digital signatures that can be added during the registration of service agents to allow the directory agents to verify the signatures and hence ensure their authenticity. These signatures are also passed to the requesting user agents. The directory agents can also attach the signatures to service advertisements.

SLP is operating system independent and is easy to implement. However, SLP also has some disadvantages. SLP does not specify such essential implementation details as how services are created. It provides only a method to discover the services, but not how to use them. Thus some implementation details are left to the manufacturer which may add complexity when services implemented by different manufacturers try to cooperate.

2.2.7 JetSend

JetSend [16] is a protocol defined by HP which can be used in environments where devices are compliant with TCP/IP, IEEE 1394 and infra red and allows such devices to communicate with each other to share services. JetSend allows two devices or information appliances to negotiate the best way to exchange information. JetSend only addresses the communication among the devices, but does not explain how the devices locate each other. JetSend is focused on devices like cameras, printers, scanners, and computers.

The main component in JetSend is the surface. In JetSend, all the information is stored as electronic media in areas called surfaces. The surfaces are the source of information. The surfaces include a name, description and content. A surface may contain instances of electronic media or links to the location of other surfaces. The device with the original copy of a surface creates copies of the surface on other devices. Other devices can then make use of the copied surface. Surface negotiation is the main exchange of information. In negotiation, the sender sends a brief description of the service and then the receiver invokes the required components to make use of that service. For example, if a certain service is encoded in a particular way, the message sent may contain information on the type of encoding. The receiver can then use that information to invoke the appropriate decoder to decipher the message.

The JetSend architecture can be implemented on any physical layer as shown in Figure 2.7 but requires a reliable Transport Protocol. If a pre-existing reliable Transport Protocol is not available, then RMTP [35] can be used.

The JetSend Session Protocol (JSP) defines the messages for setting up sessions between devices. The JetSend Interaction Protocol (JIP) defines messages to deliver surface descriptions, transfer their contents and do updates to reflect any changes that occur in the surfaces. The Interaction Policies define how the surfaces are implemented. The data contained in a surface is called electronic material (E-material).

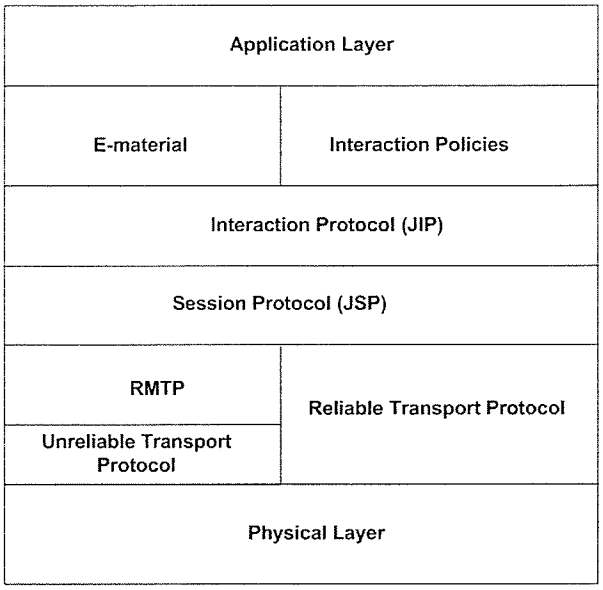

Figure 2.7: HP JetSend Protocol Stack

2.2.8 JXTA

JXTA [42] started as a research project at Sun Microsystems and is now one of the major standards meant for distributed devices. JXTA is an open source standard that is built on Peer-to-Peer protocols. It defines a set of open protocols that allow any connected device on the network ranging from cell phones and wireless PDAs to PCs and servers to communicate and collaborate in a Peer-to-Peer manner. JXTA peers create a virtual network where any peer can interact with other peers and resources directly even when some of the peers and resources are behind firewalls and NAT boxes or are using different network transports.

JXTA is mostly used in enterprise applications. However, it can also be easily integrated into a home network for automation purposes. It provides interoperability across different Peer-to-Peer systems and can be run on any platforms, systems and networks. It also makes it easier to share computational power among multiple devices in the network.

In JXTA, any device can communicate with other devices irrespective of location, type of device, and type of environment. Hence, JXTA can be used to create interhome network architectures where devices in one home can share the services offered by the devices in other

homes.

2.2.9 Home Networking with Zigbee

Galeev [26] describes how many wireless, remotely controllable devices can be incorporated into home networks using the Zigbee standard. Zigbee is a software middleware layer building on the IEEE 802.15.4 standard. It is cost-effective and does not consume much power to provide a reliable and secure network. The use of Zigbee in the home can also be considered as a replacement for X10 protocol.

Zigbee uses direct sequence spread spectrum in the 2.4 GHz band. The frame structure of Zigbee provides four basic frame types- Data, ACK, MAC and beacon. The data frame has a check sequence that ensures the arrival of the packets are in sequence. Error checking in data transfer is provided in the Data frame. ACK frame provides two way communication to ensure that the packets reach intact. MAC frames provide features to control the clients remotely. A centralized network manager can thus configure the clients at the MAC layer. Beacon frame awakens the sleeping clients to provide the address. If the clients do not get the address, they go back to sleep. Thus, Zigbee reduces the power consumption by not making clients listen to the address all the time. Channel access in Zigbee is contention based. However, the network manager can allocate up to seven dedicated time slots.

A Zigbee network uses three types of devices.

- The network coordinator which manages the entire network.
- Full function devices (FFD) which support all the functions and specifications of the 802.15.4 standard. These devices can also function as the network coordinator. They possess large memory capacity and computing power.
- Reduced function devices (RFD) are devices with limited computing capabilities.

Zigbee can only be used in wireless networks thus restricting its use in controlling wired devices. Moreover, Zigbee cannot incorporate legacy devices into the network. In terms of

service discovery and interoperability, Zigbee can only be used to detect and control services compliant with Zigbee. It does not provide any ability to control other devices.

2.3 Open Service Gateway Initiative (OSGi)

OSGi is one of the latest middleware standards and is supported by a vast number of major industry players including Sun Microsystems, Cisco, IBM, and Oracle. OSGi acts as an intermediate layer in handling the interaction between LAN/WAN and UPnP, Jini, and HAVi, etc. OSGi middleware specifications provide an open architecture for service providers, appliance vendors and manufacturers.

The service Gateway is the central component of the OSGi architecture. Devices register the services they offer with the dictionary component of the gateway. The functionality of the gateway is to provide descriptions of registered services in response to client requests. Registration and removal of services on the network is managed by the gateway.

One of the key benefits of using OSGi as the gateway device in a home network is its capability to understand both UPnP and Jini. A gateway device running OSGi can recognize both UPnP and Jini devices in the network. OSGi provides specifications to develop interface layers for UPnP and Jini. Bundles in OSGi refer to Java applications deployed on the gateway. The "UPnP bundle" installed on the gateway exports OSGi services to UPnP networks and imports UPnP services to OSGi. Likewise, the "Jini bundle" acts as the coordinator for OSGi and Jini services making them available to each other.

OSGi is an open specification meaning any protocol implemented using Java can be deployed on OSGi. OSGi bundles/applications are packaged as jar files. These jar files contain an activator class that implements start and stop methods to start and stop the bundle. Each jar file also contains a meta-interface file which stores the name of the bundle, the author of the bundle and bundle descriptions. Integrating new middleware standards into OSGi requires driver bundles that can handle the device interactions between OSGi and

the new standards. However, the driver bundles cannot communicate directly with other standards such as UPnP and Jini.

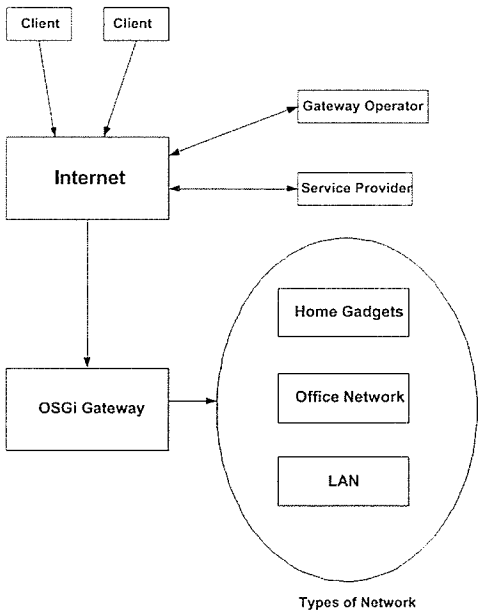

Figure 2.8: Open Services Gateway Initiative

In Figure 2.8, the major components of the OSGi framework are shown. As illustrated, an OSGi gateway can also connect to the Internet to communicate with Service Providers remotely. Service Providers are entities that can deploy useful application bundles on the gateway. For example, a simple application that requires drivers to operate, can download the drivers from the Service Providers. The gateway operator is the owner of the gateway who can perform manual configuration that may be warranted by applications. For example, the home owner may have to configure an application to instruct which lights need to be turned off at 11 PM every night.

The advantages of OSGi are several in addition to its focus on supporting interoperability. Since OSGi is built on Java, it is platform independent. OSGi also offers digital signatures like SLP to authenticate bundles that are to be installed on the gateway.

2.4 Framework/Architectural Design for Home Networks

Rasheed et al. [54] present an interoperability framework based on UPnP that discusses various interoperability issues, requirements of home networks, and the standards that are currently followed. They explain design issues in the device connectivity, application and service layers of the framework, and the benefits of such a framework, considering future home networking scenarios. Although the authors explain the details of designing a framework for UPnP, the design does not consider the existence of other standards such as Jini, HAVi and OSGi.

Friday et al. [3] have designed a new API-based scheme to develop applications that provide useful services in home networks. They discuss the issues related to service discovery, querying and interaction in a home network computing environment. Their API is designed to suit heterogeneous environments. They consider the existence of UPnP and HAVi compliant devices in the network. However, this effort is only an initial prototype design and is targeted exclusively for mobile applications. Moreover, OSGi despite being a very important standard, is not considered in the prototype design. Unfortunately, the authors do not explain how Jini can be integrated into their prototype architecture.

Bettstetter and Catterall [5] provide a very useful comparison of service discovery protocols and introduce an implementation of the Service Location Protocol (SLP). They compare different standards including UPnP, Jini, and SLP [43] with respect to service discovery.

Using event-based principles, Bates et al. [22] show that components that are not explicitly designed to interoperate can be made to work together quickly and easily. Events, in their research, correspond to the arrival/removal of devices/services within the network. For example, the parameters of an event may be matched to a specific service registration, and then transmitted to a remote client. A component notifies a distributed client of an event if the parameters of that event match with the parameters of a particular service registra-

tion. Their research was published before many of the current middleware standards were introduced, and the concept of such event handling is now used in OSGi and Jini.

Pietzuch et al. [31] designed an event-based component detection mechanism using a publish/subscribe model as an extension to generic middleware architectures. In a publish/subscribe model, services that communicate require the sending services (publishers) to publish messages without specifying the recipients. Similarly, receiving services (subscribers) must receive only those messages that the subscribers have registered for. The publish/subscribe model works on the basis of events that allow recipients to subscribe to services of interest. Their design works on pre-defined distribution policies that control event detections within the system and can be used on existing middleware architectures such as CORBA.

These research efforts have identified some of the necessary components required for designing a middleware layer in home networks. They also discuss the requirements and complexities involved in developing a middleware layer standard for home networks. However, none of the above architectural designs consider the integration of existing standards.

2.5 Middleware System Designs Related to Home Networks

Nakajima et al. [45] discuss an event-based middleware architecture that addresses the various issues in designing middleware for home networks. The authors provide an excellent analysis of the issues in building middleware for home network environments and stress the need to integrate user preferences when designing such middleware.

Ishikawa et al. [10] describe building an appliance integration middleware on top of OSGi. The authors used OSGi to develop an inference engine to generate context-based events for

creating intelligent home networks. The devices can communicate among themselves based on the inferences drawn by the inference engine. They also have experimented with dynamic user interface generation based on user preferences and context.

Nakajima et al. [44] describe a middleware design to evaluate a HAVi-based system that can work with legacy devices that do not have a Java Runtime Environment (JRE) installed. For such devices, the design proposes running small modules written in the C programming language to integrate them into the network environment. The authors also discuss customizing user interfaces as per the users' requirements. The systems allows home owners to select the type of inputs and outputs to control the intended devices. This design trims the complex user interfaces and displays only the features that are known to the home owners.

Nakajima et al. [46] have prototyped a home network middleware using Java on Linux. They compared developing HAVi-based middleware standard with others based on UPnP and concluded that developing HAVi for home devices is complex. Complexity in their research refers to the amount of effort that is required to develop HAVi-based devices. Developers need to implement HAVi middleware specifications and also HAVi devices. Currently, there are no open source implementations of HAVi. Further, HAVi only specifies the way HAVi-based devices should be implemented and the implementation details are vendor-specific. Thus, an HAVi device developed for specific HAVi middleware may not be recognized by a different HAVi middleware.

Turcan [48] explains different issues involved in querying within smart homes. He discusses scalability, sharing of services, and inclusion of mobile devices in the network. Scalability involves sharing of resources in an area of connected homes which involves a large number of heterogeneous devices. This work only mentions those characteristics that are important for Peer-to-Peer operations.

Okamura [11] has designed an adaptive resource management system for home networks that addresses the presence of resource-limited devices that are aided by functionally rich devices which donate CPU, memory and storage. The authors have implemented a prototype

of this adaptive resource management system that can detect connection and disconnection of the low-powered devices. Their simulation model shows low overhead in integrating these resource-limited devices.

Raatikainen et al. [24] discuss interoperability and location issues for mobile and distributed applications to develop a middleware layer for home networks. They also describe specific applications like mobile healthcare and home entertainment. Their research outlines the specific requirements for mobile healthcare and home entertainment systems. The authors do not discuss the architectural details of such systems.

Preuss and Cap [36] address the security issues associated with home networks. They also apply available methodologies for designing a security framework and discuss the UPnP standard with respect to security in home networks.

The middleware standards described in the research reviewed above are important, but do not address integration of existing standards. Mobile, home entertainment and security applications are also very important for home networks and highlight specific challenges. Mobile applications must integrate resource-constrained devices within the network. Home entertainment systems can integrate audio visual devices to automate the functioning of homes but require QoS guarantees. Security is an important feature since network communication is essential for all devices and is open to external observation. The only existing research addresses the integration of existing standards is by Allard et al. [21].

2.5.1 Jini Meets UPnP

Research done at the University of New Orleans focuses on achieving interoperability between Jini and UPnP [21]. The authors have implemented a Jini/UPnP interoperability framework that allows Jini clients to use UPnP services and UPnP clients to use Jini services, without modifying the services or clients.

The architecture consists of UPnP service framework that manages the UPnP network related services, Jini service discovery that handles Jini service registrations, a proxy service

that translated UPnP services to Jini and vice-a-versa. The main motivation of this research is to create a design that supports rapid proxy development and to reduce the amount of code to be written to support new services. The implementation consists of a reference design where the UPnP services are made available to the Jini network **manually** and vice-a-versa.

2.6 Comparison of Interoperability Standards

Table 3.1 shows a comparison of different interoperability standards with respect to various features. These include features that can be implemented in certain standards but have not been implemented yet. Those standards have been marked "possible".

Service integration refers to the ability to automatically integrate newly added services into the system. HAVi does not perform well because of the differences in vendor developed custom devices and services. Device integration refers to the automatic integration of devices into the network without having to manually configure the devices. Jini does not offer any specifications for developing Jini devices. Many of the standards are not portable. These standards require configuration to be made when deployed in new locations. Many of these standards cannot retain the configuration when they crash. HAVi offers flexibility in fault tolerance. Standards such as Jini, HAVi, OSGi and UPnP are platform independent. These standards are not bound to specific operating systems. The major drawback in these standards is they cannot communicate with each other. OSGi is the only specification that integrates the UPnP and Jini standards. Standards such as HAVi and UPnP are network specific. HAVi runs on IEEE 1394 firewire technology while UPnP is IP based. Security is one of the most important features in implementing a middleware layer. Security features such as digital signatures are implemented in OSGi specifications.

As seen from the chart, OSGi is more flexible than most other standards. The newest middleware system, Obje, has not been included because detailed specifications have not been released by the manufacturer, Xerox PARC. Since HomeRF, X10, and HomePNA are

Table 2.1: Comparison of Different Interoperability Standards

	OSGi	UPnP	Jini	HAVi	Salutation	JetSend
Service Integration	Good	Good	Good	Partial	Good	Not Relevant
Device Integration	Good	Possible	No	Good	Possible	Partial
Adaptability	Good	Partial	Good	Partial	Partial	Not Relevant
Fault Tolerance	No	Possible	Partial	Good	Possible	Not Relevant
Platform Independence	Good	Good	Good	Good	Good	No
Interoperability	Good	Partial	Partial	No	Partial	Partial
Network Independence	Good	No	Possible	No	Good	Good
Security	Good	Poor	Good	Good	Partial	Possible

hardware-based standards, it is difficult to achieve interoperability with them.

2.7 Home Automation Research

This section briefly overviews some of the notable industrial and academic research on home automation systems. The network environments in all these projects use one of UPnP, Jini or OSGi standards. However, none of these research projects focus on integrating these standards to make home devices interoperate.

2.7.1 Easy Living

Easy Living [27] is the brainchild of Microsoft Research. It is an intelligent environment designed to assist users with information and services stored in home devices. The home network environment in Easy Living consists of motion sensors, PCs, projectors, and light devices. The system can detect and configure devices compliant with UPnP technology. Components of Easy Living include middleware to facilitate inter-communication between sensor and other UPnP devices, physical space modelling to provide location-oriented context, sensing to collect information about the location and state of devices and people, and service description to support device control.

When a person enters a room, a motion sensor detects his presence and turns the light on if its dark. If the person wishes to use a PC, he has to log in to authenticate himself. Once the user is authenticated, his information is sent to a centralized context broker. The context broker provides a centralized repository of personal histories. If the same person moves from the PC to any other devices within the network, his movement is detected by a motion sensor and the information is sent to the context broker. The context broker re-authenticates the user on the new device based on his previous login and then facilitates access to other devices. The context broker also provides intelligence to the network by user session monitoring. Users can search for services, select services, remotely configure the system, and also share and transfer media easily with the help of devices such as PCs.

2.7.2 Philips Ambient Intelligence

The Ambient Intelligence project from Philips deals with integrating people and electronic appliances in day-to-day life [32]. Electronic devices are used to assist people with decision making based on their preferences. One of the scenarios described in their design discussion is as follows: a user is awakened by a clock and while brushing his teeth, the intelligent mirror informs the user if he will be late for work. The mirror also displays the user's favorite television channel according to the preferences set.

Intelligent systems know who people are and in which context people operate different devices within a network. When a PC is turned on, for example, it has no way of knowing the mood of the people. For an Ambient Intelligence system, however, that greets us when we get home, selects suitable background music and lighting, or advises us on the state of our health, the system must know when to keep quiet and when to speak up. Ambient Intelligence systems are envisioned to be supportive, because they are aware of the physical presence of the users, and can adapt to their habits and wishes. Such systems need to include methods to discover the identity and location of users, devices and objects.

2.7.3 Gator Tech Smart House

The Gator Tech Smart House is a research project conducted at The University of Florida [2]. This laboratory house is built for elderly people to live in independently. Dr. Sumi Helal, the creator of the smart house, has instigated various related research projects such as tracking and monitoring elderly people and an intelligent meal planner for them.

The physical layer of the system consists of various devices and appliances such as a TV, radio, set-top-box, etc. The Sensor platform layer can communicate with a variety of devices and represent them to the middleware layer. The middleware layer is OSGi-based and keeps track of the deployed sensor devices. Application developers can browse the existing services to create new services and deploy them in the home. A "Knowledge" layer uses a reasoning engine to determine if certain services are available. A "Context management" layer is

responsible for using certain services based on events. For example, reminding people to take medicines once they have their lunches.

Easy Living makes use of UPnP technology to automate functionalities of the homes. It does not focus on the existing popular technologies such as Jini and HAVi. Ambient Intelligence research focuses on creating an ambience that can be automatically customized based on user preferences. The emphasis in this research is more on customizations than the technology. Since Ambient Intelligence research is more focussed on the network-based devices, this alone cannot perform interoperability among other devices that may exist in homes. Gator Tech Smart House is an application-oriented approach using OSGi as the gateway device to assist elderly people. It only makes use of the sensor devices to track and monitor the movements of the people. Intelligence in this research is merely an application that is deployed on the gateway. Further, this research does not focus on home devices built with different specifications.

Chapter 3

Problem Description

A home network consists of devices such as television sets, video cassette recorders, DVD players, computers and many other devices that are rich in functionalities. However, the capabilities of these devices are not being utilized to the fullest. These devices are usually operated manually and do not interoperate to facilitate automated functioning of the home. The underlying problem in establishing communication between many of these devices is interoperability. For example, a UPnP television might have to record a program on a Jini VCR if the home owner misses his favourite program.

A typical home network consists of devices compliant with different protocols. Interoperability is the key to achieve automation in homes containing many devices with different protocol compliance. Major industry players involved in bringing out these functionally rich devices do not follow a standard protocol in their designs. This lack of a unified standard has resulted in numerous standards such as UPnP, Jini, HAVi, and OSGi. Typically, a home user buys devices based on the functionalities offered by the devices, rather than selecting devices adhering to a single standard. Thus, to combine these various devices under one

home network, a universal home network standard is required. This standard must be able to seamlessly integrate all types of home network devices. It should also be able to discover, configure and advertise the functionalities of these devices within the network.

Currently available protocols/systems such as UPnP, Jini, and HAVi are becoming increasingly popular as "standards" in home networks. However, a quick review of these individual standards reveals their shortcomings in developing applications in heterogeneous home networks. Some standards such as UPnP are network protocol dependent, while others deal with only audio and video devices (HAVi). OSGi is an open specification that allows integration of multiple standards. OSGi currently provides support to partially integrate UPnP and Jini. Thus a home network comprised of both UPnP and Jini devices can be easily controlled and made to communicate with each other through OSGi. Moreover, since these standards are end-user specific, home users need not be interested in the underlying technology, but only in the functionalities offered by the individual devices. Home users require effective, 'all-in-one' technology that provides the advantage of allowing all types of devices to interoperate without having to manually configure them to make them compatible with each other. Since evolving applications will likely want to combine many heterogeneous devices supporting UPnP, Jini and many other disparate standards, a unifying interoperability standard must be capable of handling device communication and sharing of the resources offered by them.

The aim of this thesis is to build a proof of concept framework for interoperability in home networks. The framework will briefly address the following issues:

- Heterogeneity of the network environment
- Collaboration of various services offered by these heterogeneous home devices
- Inclusion of resource-constrained devices into the home environment
- Extendibility of home gateway in integrating future home network protocols
- Security issues associated with home gateways. I make use of security features of OSGi and Java.

- In future, increase in interoperability standards and network traffic will make QoS imperative.

However, my framework does not address quality of service (QoS) parameters typically associated with gateways. In addition to QoS, my framework does not focus on fault tolerance and manageability of the home gateway.

Chapter 4

Design and Implementation of

Interoperability Framework

My interoperability framework runs on OSGi and is assumed to execute on the home gateway device which os further assumed to be a reasonably powerful device persistent storage capability.

Jini Meets UPnP, though comes very close to my research, is not home network oriented. However, this research provides a detailed description on how to create proxy services to handle interoperability between UPnP and Jini. Jini Meets UPnP concludes that automatic conversion of new services between UPnP to Jini is not possible thus warranting significant development effort. Thus, in all the above industrial and academic projects, achieving interoperability automatically among various standards has not been addressed at all. These individual projects restrict the users to a particular standard.

My interoperability framework integrates UPnP and Jini through OSGi. UPnPLUS bundle handles the service translations from UPnP to Jini automatically and Jini-to-UPnP translator handles the service conversions from Jini to UPnP. With the integration of my interoperability framework into OSGi, more useful home automation applications can be developed involving a variety of devices. These two interoperability bundles can readily be re-used in any home networks to integrate UPnP and Jini. I extended OSGi to build a gateway service that can communicate with a variety of devices compliant with technologies such as UPnP and Jini. I used two open source implementations of OSGi, namely, Oscar [33] and Knopflerfish [25]. In my prototype, the interoperability framework bundles developed for Oscar can be directly used with Knopflerfish and vice-a-versa, without having to modify the framework.

4.1 Overview of the Prototype Home Gateway

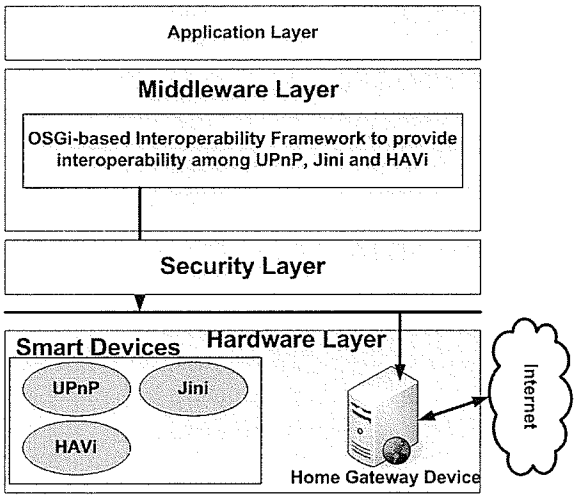

Figure 4.1: Home Gateway Architecture

The home gateway, the core device of a home network, is assumed to have a large capacity

storage device (where operating system, framework code, and applications are stored) and is responsible for control, management, and connection of all home devices within a home. In addition, the system must provide a method for communicating and sharing services among different devices.

Figure 4.1 shows the assumed architecture of a home network. It consists of four layers: hardware, security, middleware and application.

The hardware layer consists of the gateway and other home devices compliant with UPnP, Jini and HAVi and all associated cabling and wireless equipment. The gateway device is also accessible through the Internet so that users can control the gateway remotely.

The security layer prevents the home gateway from attacks. OSGi offers three level of security [29]. Admin level security provides administrator access to the gateway thereby letting the user take full control of the gateway and the applications deployed on the gateway. Package level security allows users to make controlled use of the packages exported by the deployed applications. Service level security grants the users access to the services offered by the applications. Service level users cannot make modifications to the gateway. OSGi also provides digital signatures wherein bundles can only be deployed once the signatures are authenticated thus, only bundles from known, trustworthy providers will be installed on the gateway. In addition to security measures offered by OSGi, my framework also makes use of Java 2 security features. Java offers policy files, using which authorization rights can be described by users [40]. However, my framework does not implement any additional security.

The middleware layer hosts the components that integrate a variety of home devices built with UPnP, Jini and HAVi specifications. The middleware layer also handles device configuration, service discovery and sharing of resources among devices. This middleware layer is a component that is built into the home gateway device thus giving control to the gateway to monitor and manage the home devices.

Above the middleware layer is the application layer where useful applications can be deployed to automate the functioning of homes. Additional components to manage the home gateway can also be deployed as bundles at the application layer. For example, a

bundle that can provide useful context information to monitor device behaviour to better automate homes can be deployed. Further, a QoS bundle can be deployed to offer QoS guarantees to traffic handled by the gateway on the gateway. Useful bundles to make the gateway fault tolerant and self-configuring could also be deployed at the application layer to support the home gateway. All these useful features may be implemented by other home gateway projects at TRLabs but my thesis does not focus on implementing any of these features.

4.2 Architecture of the Home Gateway

My prototype home gateway is essentially a computer running the Oscar [33] as well as the Knopflerfish [25] OSGi implementations. The purpose of using both Oscar and Knopflerfish is to test the functioning of these two open source implementations. A sample application that I developed specifically for Oscar was also successfully deployed on Knopflerfish. This verified the intended behaviour of the application on both the OSGi implementations.

Services are deployed as bundles on the gateway. Bundles are Java jar files implemented according to OSGi specifications [29]. These bundles include an activator Java file that implements start and stop methods to start and stop the bundle once deployed on the gateway. As described earlier, bundles also contain a manifest file that provides metadata for the bundle such as the name of the bundle, the author of the bundle, the intended use of the bundle, the packages imported and exported, and the version number of the bundle all in the form of attribute-value pairs. The bundle version is a very important feature in OSGi. Different versions of the same bundle can be deployed at the same time for compatibility and testing purposes.

Once a bundle is deployed, it registers with OSGi's dictionary. The dictionary is merely a centralized repository of information for all services registered within OSGi. Each deployed bundle obtains a bundle context from OSGi, using which it can query the other deployed

bundles, use the services exported by other bundles, and offer its services to other bundles. My interoperability framework bundles also implement all these features. My UPnPLUS bundle provides interoperability from UPnP to Jini while Jini to UPnP translations handle the service translations from Jini to UPnP.

4.2.1 Basic Interoperability in OSGi

OSGi release 3 provides support to integrate some aspects of Jini and UPnP operations. The UPnP Base Driver bundle installed on OSGi listens to UPnP service advertisements. When the Base Driver detects UPnP services, it simply imports the services into OSGi. For example, if a UPnP light device is detected on the network (external to OSGi), the new UPnP service within OSGi is registered in the dictionary. Any calls to OSGi to invoke the actual services implemented by the UPnP light device are taken care of by the Base Driver. In summary, the UPnP Base Driver exports the UPnP services from OSGi to the UPnP network and imports the UPnP services from UPnP network to OSGi. Similarly, a Jini Driver exports all the Jini services from OSGi to the Jini network and imports all the Jini services from the Jini network to OSGi. However, Jini and UPnP services cannot directly call one another.

Currently, there are many OSGi-based home gateway architectures [30]. However, none of them focuses on achieving interoperability with other home network protocols using the gateway. With OSGi, it is possible to indirectly integrate Jini and UPnP, the two most popular standards as well as others. However, the underlying problem is achieving direct inter-operation between, for example, a Jini device and a UPnP device.

Jini is based on Java RMI while UPnP is built with network-related protocols such as the Simple Service Discovery Protocol (SSDP) and the Simple Object Access Protocol (SOAP). Service advertisement in UPnP is done using broadcast of XML description locations. The UPnP standard specification also supports only a limited number of data types for use by the implemented services. These services need to be converted to Java interfaces for a Jini

client to be able to invoke UPnP services. On the other hand, Jini is rich in data types and uses a centralized LookUp Server to register the proxy services. A Jini service with data types other than primitives such as integer, boolean, float, string, and character cannot easily be automatically translated to UPnP since UPnP does not implement such data types. Thus, automatic conversion of the data types from Jini to UPnP is complicated. However, with significant development efforts, conversion from Jini to UPnP can be achieved. UPnP, unlike Jini, does not require the proxy code to be downloaded to the client side. However, Jini requires mobility of code among clients. There have been efforts to achieve interoperability between UPnP and Jini, but only partially or per service where for each new services and a significant amount of manual development effort is required for service translation [21].

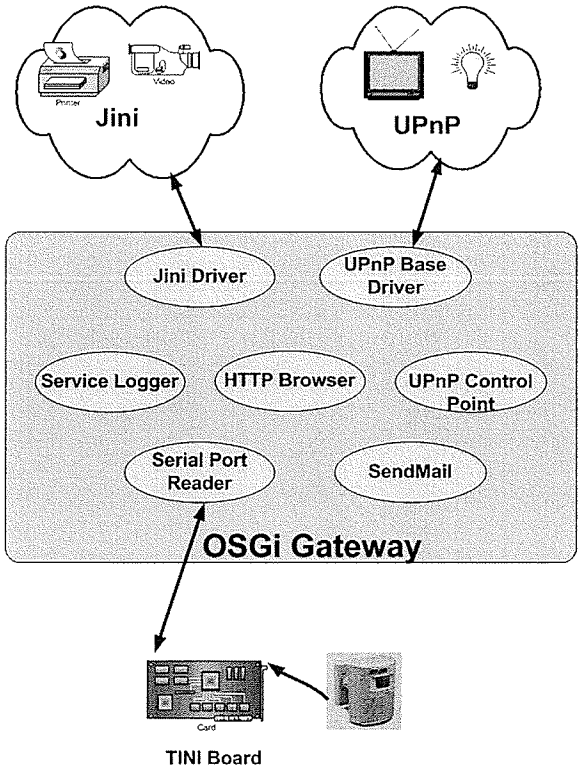

Figure 4.2: The Home Gateway Prototype

Figure 4.2 shows the assumed prototype implementation of the home gateway including UPnP, Jini and TINI boards [6]. The Jini Driver bundle exports its known services from

OSGi to the Jini network automatically and imports Jini services to OSGi. The Service Logger bundle logs all the services in the OSGi dictionary, the central repository of service information. The Serial Port reader bundle monitors the serial port on the PC for data from the TINI-based sensing board. As a simple door security application example, I connected a sensing board to the serial port of the PC. A sensor is deployed at the door that senses the light intensity. Light intensity values are transferred to the serial port of the PC through the sensing board. A threshold limit is set to this light intensity. When the sensor detects a movement at the door, the light intensity values go down. When this value goes below the threshold, the bundle turns on the camera stream player at the door to capture display from the webcam. Thus, security application example, demonstrates including non-networked devices into OSGi. The UPnP base driver performs similarly to the Jini Driver by importing and exporting UPnP services. The UPnP Control Point is an interface that can detect the presence of UPnP services within the network. Using this, a remote ISP can control a variety of UPnP devices within the network. Finally, the Sendmail bundle monitors the newly added UPnP and Jini services and sends an email to the home owner giving information about any newly registered services in the OSGi. This is particularly useful when ISPs deploy useful services in homes automatically.

4.3 Prototype Implementation of the Gateway

My interoperability framework uses the concepts used in the UPnP middleware layer discussed by Newmarch et al. in [23] to translate UPnP services to Jini. For every UPnP service, an equivalent Jini service interface is generated and any requesting Jini clients can then invoke the UPnP services using the created interfaces. My framework achieves UPnP to Jini and Jini to UPnP service translations using OSGi.

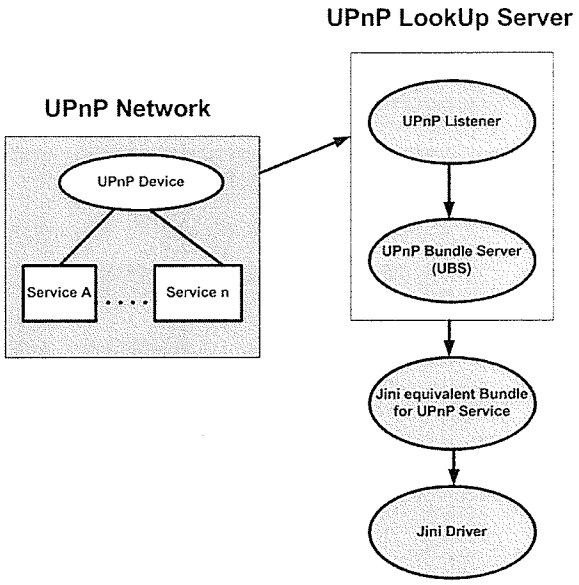

Figure 4.3: UPnP LookUp Server (LUS)

4.3.1 UPnP LookUP Server

My UPnP LookUp Server (UPnPLUS) bundle translates all UPnP services to Jini services. Figure 4.3 shows the architecture of UPnPLUS. A UPnP network is comprised of many UPnP devices and the services that they advertise. My UPnPLUS component is deployed on OSGi. UPnPLUS consists of a listener module that listens for service advertisements multicast by UPnP devices, and a UPnP Bundle Server (UBS) that creates equivalent Jini services for the UPnP services that are deployed on OSGi.

My UPnPLUS bundle is based on the Domoware UPnP Control Point [8]. UPnPLUS makes use of the Control Point to discover the newly added UPnP devices. UPnPLUS monitors service advertisements made by UPnP devices on the network and when a UPnP service advertisement is detected, UPnPLUS captures the XML description file and uses it to create the equivalent Java Interface file to be used by Jini clients.

A sample UPnP light device service described in XML, shown below, has two implemented services; SetPower and GetPower. Using SetPower, the UPnP device status can be altered and using GetPower, the current status of the device can be queried. These two services

receive boolean values as input and based on the received value, appropriate actions are taken by the light device. If SetPower receives boolean value 0, it turns on the light device and if it receives 1, it turns the device off.

```
<?xml version="1.0"?>
<scpd xmlns="urn:schemas-upnp-org:service-1-0" >
    <specVersion>
        <major>1</major>
        <minor>0</minor>
    </specVersion>
    <actionList>
        <action>
            <name>SetState</name>
            <argumentList>
                <argument>
                    <name>Power</name>
                    <relatedStateVariable>Power</relatedStateVariable>
                    <direction>in</direction>
                </argument>
                <argument>
                    <name>Result</name>
                    <relatedStateVariable>Result</relatedStateVariable>
                    <direction>out</direction>
                </argument>
            </argumentList>
        </action>
        <action>
            <name>GetState</name>
            <argumentList>
                <argument>
                    <name>Power</name>
                    <relatedStateVariable>Power</relatedStateVariable>
                    <direction>out</direction>
                </argument>
            </argumentList>
        </action>
    </actionList>
    <serviceStateTable>
        <stateVariable sendEvents="yes">
            <name>Power</name>
            <dataType>boolean</dataType>
            <allowedValueList>
                <allowedValue>0</allowedValue>
                <allowedValue>1</allowedValue>
            </allowedValueList>
            <allowedValueRange>
                <maximum>123</maximum>
                <minimum>19</minimum>
                <step>1</step>
            </allowedValueRange>
```

```
        </stateVariable>
        <stateVariable sendEvents="no">
            <name>Result</name>
            <dataType>boolean</dataType>
        </stateVariable>
    </serviceStateTable>
</scpd>
```

The UPnP XML service description file converted to a Jini interface file is shown below.

UPnPLUS translates the XML services to Java interface files. The UPnPLUS bundle parses the XML files to find the service descriptions and the input data types they expect. Based on the services, the UPnPLUS bundle creates a Java interface file with methods corresponding to the services. The SetState method in the example corresponds to the implementation of the UPnP service SetState and the GetState method corresponds to the implementation of GetState. SetState accepts two inputs, State and Result. State receives boolean values and depending on the input it either turns the UPnP light bulb on or off. GetState queries the current state of the UPnP light device and returns the appropriate result. The generated interface file is saved as a Java file and is compiled by UPnPLUS to create a bundle. urn-schemas-upnp-org-service- is a constant textual prefix for each name field used in the generated interface files. The actual name of the service is then appended at the end. For example, in the interface file generated, SwitchPower is appended to urn-schemas-upnp-org-service-. addEventListener is also a constant for all services to listen to Jini-based requests. All "in" and "out" variables listed in XML description files correspond to input and output variables. All output variables are created as type StringHolder to hold the output values. UPnPLUS then compiles the Java interface file, creates a bundle and deploys it on OSGi with a callback routine for the service pointing to the actual UPnP device. Thus, a Jini client can request and invoke required UPnP services.

```
import java.rmi.Remote;
import java.rmi.RemoteException;
import net.jini.core.event.RemoteEventListener;
import net.jini.core.event.EventRegistration;

public interface urn-schemas-upnp-org-service-SwitchPower extends
Remote {
```

```
    void SetState(String State, StringHolder Result) throws RemoteException;
    void GetState(StringHolder State) throws RemoteException;
    EventRegistration addEventListener(RemoteEventListener listener);
}
```

4.3.2 UPnP Bundle Server (UBS)

My UPnP Bundle Server (UBS) bundle creates a corresponding Jini service bundle using the Java interface files and deploys the bundle on OSGi. The Jini driver listens for Jini events and exports any newly discovered Jini services to make them available to the Jini clients. The translation process from UPnP to Jini is automatic without requiring any modifications to the existing code.

4.3.3 Jini-to-UPnP Service Translations

A UPnP device has a number of properties such as device name, manufacturer, etc. unique to each UPnP device that are used to identify them. Each service contains a set of actions to perform tasks such as turning on or turning off a light device. Actions take parameters as input or provide them as output, and each parameter is associated with a state variable. State variables are of a small number of types such as integer and string, and may contain any values of their corresponding types.

UPnP has defined a set of standard data types: ui1, ui2, ui4, i1, i2, i4, int, r4, r8, number, fixed.14.4, float, char string, date, dateTime, dateTime.tz, time, time.tz, boolean, bin.base64, bin.hex, uri, and uuid. State variables must be one of these data types. Further, data types represented by SOAP have a string format, so the string has to be converted to one of the above data types. This is required since translated Jini services have to provide SOAP invocation to requesting UPnP devices. Some UPnP data types can be represented by Java primitive types. Automatic service translations from Jini to UPnP, however, are

difficult to achieve because Jini is rich in data types. Automatic conversion of these complex data types to UPnP-specific services requires significant manual development efforts for each of the services.

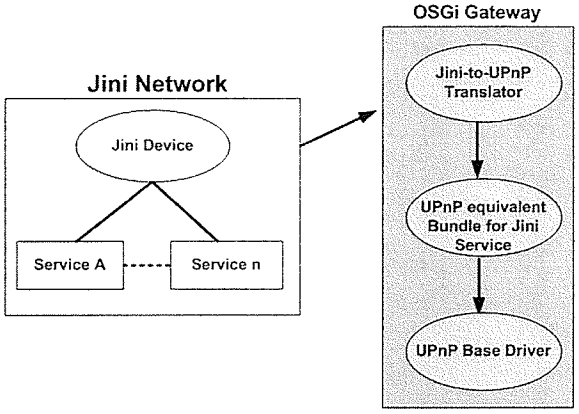

Figure 4.4: Jini-to-UPnP Service Translator

My prototype implementation consists of a Jini-to-UPnP Service Translator bundle as shown in Figure 4.4, that can handle the conversion of Jini services to UPnP for services with data types such as integer and boolean. My Jini-to-UPnP bundle listens for Jini services and works in the same fashion as UBS. When the Jini-to-UPnP bundle discovers Jini services within OSGi, it converts the service interface files to XML files and registers them as UPnP services within OSGi. The method names in the interface files are used as service names for UPnP with the data types used in those methods as inputs for the UPnP services. Intel's UPnP Device Builder [19] can generate UPnP devices based on the XML description files. Once, the translator provides an XML description file for the Jini service, I use Intel's UPnP Device Builder to generate the UPnP device bundle. Thus, the translator creates UPnP equivalent bundles for Jini services and deploys them on OSGi. The UPnP Base Driver then automatically exports the newly added UPnP services to the UPnP network where they can then be used by UPnP clients. In cases where the data types are complex, this bundle throws exceptions to request that such services be translated manually.

4.4 Integration of Resource-Constrained Devices

My purpose in integrating resource-constrained devices was to demonstrate that if the devices within homes that do not run Java could be included in a home network to offer useful services. In addition, I have ported my entire interoperability framework onto a resource-constrained device, a Pocket PC, to show that the framework was small enough to run on a device with characteristics similar to an embedded home gateway. To do this, I configured an HP iPAQ to run JVM using IBM's J9 [17] and ran Oscar on the iPAQ and deployed my framework bundles as shown in Figure 4.5. I also integrated TINI, an embedded network processor board [6] thereby demonstrating that resource-constrained devices within a home network could be integrated using such low cost network interface boards.

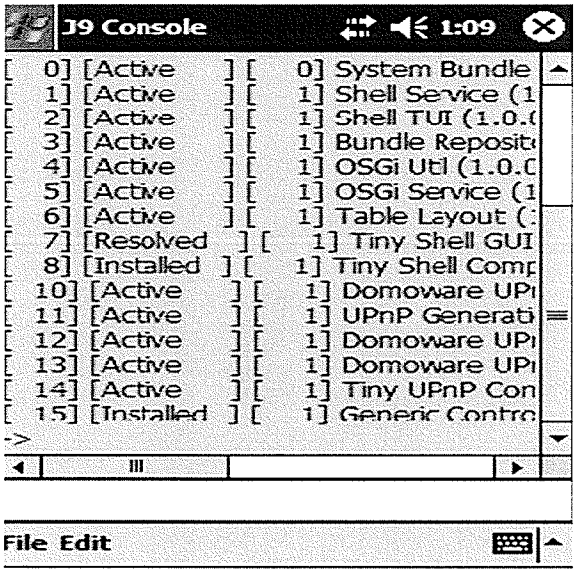

Figure 4.5: Oscar running on iPAQ 3850 Pocket PC

4.5 Essential OSGi Bundles

I also used several OSGi release 3 framework bundles such as HTTP admin, Log service, and OSGi services for various purposes. The HTTP admin bundle provides basic authentication of user logins. The Log service bundle enables logging of all OSGi events such as install/uninstall events. The OSGi service bundle provides support to export and import bundles to and from OSGi. The OSGi service bundle also provides bundle context to the deployed bundles. An application bundle can use any of the packages exported by already deployed bundles. I have also created bundles that implement various applications explained in the next section. These application bundles make use of HTTP Admin, Log Service, and OSGi service bundles.

4.6 Example Home Gateway Applications

I have also developed several application bundles implementing example home environment functions and a browser-based GUI for remote administration of the gateway, and a serial port listener bundle, based on the javaxcomm API [38] that can be used to monitor data coming in on the serial port of the PC.

One example application that I implemented was a simple security service using a webcam. I used the Java Media Framework (JMF) [39] API to read the capture device (webcam) and transmit the raw data (uncompressed) using RTP over an in-home local area network using a specific port. The home owner can use my application to check the camera over the Internet to monitor his home remotely. Figure 4.6 shows a captured frame from the streaming webcam using JMStudio player, which is part of JMF. Though the webcam is connected to the gateway with a different IP address, by specifying the last byte of the IP address of the stream to be 255, the stream can be accessed via "broadcast" by any computer within the local area network.

Figure 4.7 shows the browser-based management GUI for the home gateway. This browser

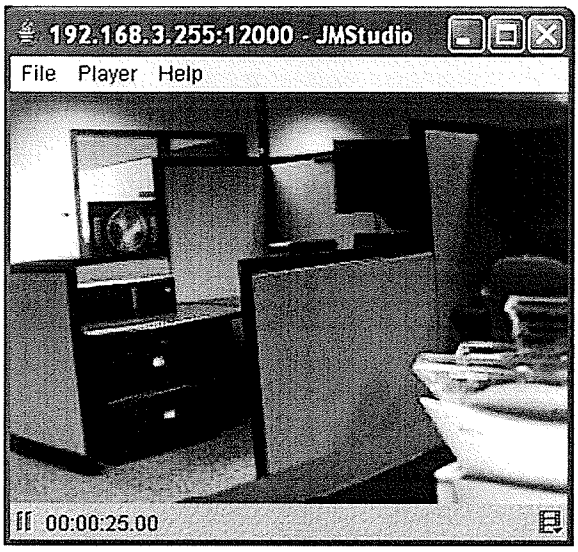

Figure 4.6: Surveillance Camera at TRLabs

bundle can be run on any device with support for web browser. Using this graphical interface, the home owner can install, uninstall and update the bundles. This browser bundle can also be used to remotely operate the gateway. The browser window refreshes itself every thirty seconds to display the current status of the home gateway. As a simple security mechanism, a sign-in page is displayed to users, requesting a username and password which are used to authenticate access.

An Email notification service is another useful application that I implemented for the prototype home gateway. Email notification is based on traditional sendmail technique. Whenever a new service is installed on the home gateway, this application detects the installation and emails the home owner telling them about the newly installed service.

The example applications and various management services were used in testing my implementation as described in the next chapter.

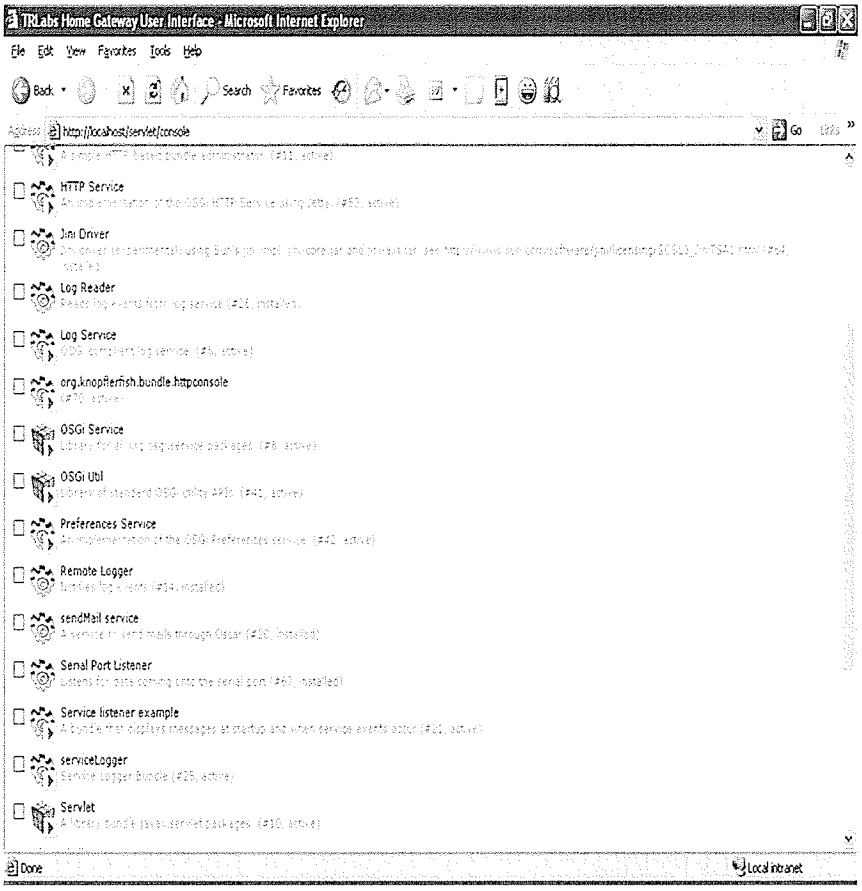

Figure 4.7: Browser-based Interface for the Gateway

Chapter 5

Assessment of the Interoperability Framework

Because of the nature of my framework, assessment was done only to ascertain the functional operation of key components such as UPnPLUS and Jini to UPnP translations. No performance assessment was done.

The prototype implementation of my interoperability framework was evaluated in two stages. In the first stage, I compared my work with the "Jini Meets UPnP" [21] and the "UPnP to Jini interoperability" research [23]. In the second stage, a few automated applications were developed to demonstrate interoperability between UPnP and Jini.

5.1 Comparison of Interoperability Frameworks

Jini meets UPnP [21] was the first work done in interoperability in home networks. In their work, Allard et al. implemented a bridge to integrate Jini and UPnP. However, the service translations from UPnP to Jini and vice versa are service-specific involving significant manual development efforts for each of the newly added services. In addition to the non-automatic nature of their work, there is no support for OSGi. My interoperability framework is advantageous in three ways:

1. Support for OSGi along with Jini and UPnP
2. Implementation of a centralized gateway service that can be used to configure and control the operation of UPnP and Jini
3. My framework is automatic in integrating UPnP and Jini services without having to manually create service-specific translations involving significant development overhead

To the best of my knowledge, the UPnP to Jini interoperability work by Newmarch, etc. [23] is the only other research on building an interoperability framework for UPnP and Jini. Their work builds a framework that can offer UPnP services to a Jini network automatically but not Jini services to UPnP. My proposed work distinguishes itself by integrating UPnP, Jini and OSGi technologies.

My interoperability framework is also a part of a larger project in the Home Technologies Group at TRLabs. My framework is a proof-of-concept implementation of a gateway service to integrate UPnP and Jini using OSGi. Hence, issues related to quality of service (QoS) and other issues will be able to be explored by other TRLabs researchers in the future.

5.2 Interoperability with Example Applications

In the first stage of application testing, I tested the home gateway running on Windows and Linux PCs involving UPnP and Jini networks. The entire implementation of the framework was tested on two open source implementations of OSGi, namely, Oscar and Knopflerfish. Although both Oscar and Knopflerfish are fully compliant with the OSGi 3 specifications, the purpose of testing on both the platforms was to assess the correct behaviour of my interoperability framework on both the implementations. I used emulated additional devices compliant with UPnP and Jini using the IncaX Jini builder [1] to test the correctness of my respective interoperability bundles installed on OSGi in importing and exporting multiple services. The presence of UPnP and Jini devices was correctly recorded by the Jini and UPnP Drivers, and then stored in the OSGi dictionary. The dictionary stores the name of the services with a special tag to identify where they come from. For example, a Jini service may be stored with a "Jini" tag. I tested the UPnPLUS and Jini to UPnP service translations to study the correctness of a UPnP device communicating with a Jini device through a OSGi-based gateway and vice-a-versa. An effective or successful test of the interoperability framework was when a UPnP service was made available to Jini network and vice-a-versa. The emulated devices were found to correctly interoperate. In the second phase, I have implemented a few more "realistic" and hence complicated automated applications to demonstrate the correct behaviour of the UPnP/Jini emulated devices and also the TINI boards with the interoperability framework.

Wake Up Service Application

This application makes use of UPnP clock communicating with a Jini light bulb. The UPnP clock monitors the time and when the clock strikes 6 AM, it searches for light devices. When the clock finds the Jini light device, it turns it on. The Jini light device is a very simple emulated Jini device with two services; turnOn and turnOff which accept boolean values as inputs. When turnOn receives 0, it turns the light device on and when the turnOff service receives 1, it turns the light off. When the Jini light device is detected by the network, its services are translated to UPnP services and the UPnP clock searching for light devices

correctly finds and invokes the appropriate turnOn service.

Time Announcement Application

My time announcement application demonstrates the integration of Jini and UPnP. *announceTime()* is a Jini service that returns the current system time (String) to the requesting client. The home owner would like the UPnP light device to turn off at 10 PM everyday. The UPnP light device fetches the system time and turns itself off. This application successfully notifies UPnP devices of the translated Jini service's availability. Thus the time announcement application establishes interoperability between Jini and UPnP.

Home Security Application

This is a proof-of-concept prototype of a simple home security application. It makes use of a non-networked door sensor deployed at the entrance to a home. The door sensor transmits light intensity levels to the gateway. The sensor application deployed on the gateway receives the light intensity levels and analyses if the value goes below a threshold value to indicate that an individual has arrived at the door. When the application detects the value below the threshold, it turns on the Jini-based security camera player on the gateway. Thus, the home owner can decide whether to answer the door or not. All the above test application executed correctly.

Chapter 6

Conclusions and Future Work

The use of home networking is growing with the development of more capable in-home devices and the release of newer interoperability standards. Due to lack of a single interoperability standard, however, the home gateway must be adaptable to accommodate multiple standards.

My framework designs and implements interoperability capabilities that enable communication and collaboration of services among devices belonging to different home network standards. My framework uses OSGi to build the home gateway which handles the basic interaction among these heterogeneous devices. The following are the contributions of my thesis:

1. Design of an interoperability framework and the implementation of framework components to integrate UPnP and Jini through OSGi. Specifically, I implemented UPnPLUS to translate UPnP services to Jini services and a Jini-to-UPnP translator to convert Jini services to UPnP services, for simple, well defined types.

2. I have installed the Oscar OSGI implementation and tested my interoperability frame-

work on low-end devices such as iPAQ and Dell Axim Pocket PCs to demonstrate OSGi's functionality on such devices. The home gateway will ultimately reside on an embedded device so my experiments in porting the framework to Pocket PCs help tp demonstrate the possibility of deploying the framework components on embedded devices as well.

3. Using a TINI board to allow a resource-constrained device to behave as Jini device and be a part of the home network shows how legacy, non-networked devices might be integrated.

4. I modified the existing browser-based GUI to authenticate gateway users with a sign in page to support secure and remote monitoring of the home gateway.

5. I developed a security camera application as described in Section 5.2 as a proof of concept interoperable application on OSGi.

6. My framework bundles have been tested on both Oscar and Knopflerfish frameworks. During the development of one sample application, the bundle performed well on Knopflerfish, but failed on Oscar. This shows that there are incompatibilities between the two OSGi implementations but I have yet to determine precisely why the application failed.

There is a wide range of possible future work. For example, my framework could be tailored to hospital, office and other environments. Currently, the medical instruments used in hospitals are not compliant with any of the home network standards but by integrating more and more standards into OSGi, a wide variety of devices can be automated. This would be particularly useful in hospital settings.

At present, HAVi enabled devices are being produced in the appliances market. As part of the future work, HAVi could be integrated into my framework. Similarly, as other standards arise they too could be added to the framework.

Useful applications such as monitoring the elderly in homes could be developed. Though my framework is more focussed on home networks, it could also be tailored to automate

functions in nursing care situations. Patients could be assisted in sending and receiving readings related to their health to the health practitioners.

Current applications that are developed on home networks, do not require strong security since many do not communicate with the Internet. However, with future applications, security will be a primary concern in protecting home area networks from attacks and illegal access. In addition, since devices compliant with different standards coexist in one network, an interoperability framework must be capable of integrating all the different security systems and compensating for the deficiencies of the weaker ones..

The home gateway device should also be capable of performing self-recovery after a crash. Current gateway devices do not implement fault tolerance. This would be useful since the gateway device must be highly reliable. At a minimum, the gateway should retain the last known good configuration and have a mechanism to boot from it, if needed.

More research can also be done to implement and assess the gateway's performance on an embedded device since future gateways will reside on an embedded device rather than a powerful PC. Further analysis could also be done on storing multimedia content on an available in-home PC and on using the embedded gateway device to install required bundles on demand to utilize the resources efficiently.

Bibliography

[1] Inca X IDE & JavaSpaces Starter Kit. IncaX. http://www.incax.com/. Retrieved on 05-05-2005.

[2] S. Helal, W. Mann, H. El-Zabadani, J. King, Y. Kaddoura, and E. Jansen. The Gator Tech Smart House: A Programmable Pervasive Space. *IEEE Computer Journal*, 38(3):50–60, 2005.

[3] A. Friday, N. Davies and E. Catterall. Supporting Service Discovery, Querying and Interaction in Ubiquitous Computing Environments. In *Proc. of the 2nd ACM Int'l Workshop on Data Engineering for Wireless and Mobile Access*, pages 7–13, May 2001.

[4] Home Phone Networking Alliance. HomePNA Technical Specifications. http://www.homepna.org/. Accessed on 3-04-2004.

[5] C. Bettstetter and C. Renner. A Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol. http://wwwtgs.cs.utwente.nl/Docs/eunice/summerschool/papers/paper5-1.pdf, 2000. Retrieved on 12-07-2004.

[6] Dallas Semiconductor. TINI- Tiny InterNet Interfaces. http://www.maxim-ic.com/TINIplatform.cfm. Retrieved on 10-10-2004.

[7] DHCP ORGANIZATION. Dynamic Host Configuration Protocol. http://www.dhcp.org/. Accessed on 27-04-2004.

[8] Domoware Team. UPnP Device Emulation. http://domoware.isti.cnr.it/. Retrieved on 17-10-2004.

[9] G. Kotsis. Performance Management in Ubiquitous Computing Environments. In *Proc. of the 15th Int'l Conf. on Computer communication*, pages 988–997, Aug 2002.

[10] H. Ishikawa, Y. Ogata, K. Adachi, and T. Nakajima. Building Smart Appliance Integration Middleware on the OSGi Framework. In *Seventh IEEE International Symposium on Object-Oriented Real-Time Distributed Computing*, pages 139–146, May 2004.

[11] H. Okamura. Adaptive Resource Management System for Home Area Networks. *Sony CSL Technical Report*, 06(04):16–19, Apr 2001.

[12] HAVi. Home Audio Video Interoperability. http://www.havi.org. Accessed on 05-03-2004.

[13] HomePlug Power Alliance. Audio Video (AV) Specifications. http://www.homeplug.org/en/faq/index.asp. Accessed on 13-07-2006.

[14] HomePlug Power Alliance. HomePlug 1.0 Technology White Paper. http://www.homeplug.org/en/docs/HP_1.0_TechnicalWhitePaper_FINAL.pdf. Accessed on 13-07-2006.

[15] HomeRF Working Group. HomeRF Technical Specifications. http://www.palowireless.com/homerf/homerf4.asp. Accessed on 09-04-2004.

[16] HP JetSend. JetSend Interoperability Protocol. http://search.hp.com/gwuseng/query.html?col=hpcom+ccen+ccenfor&qt=jetsend&la=en. Accessed on 26-06-2004.

[17] IBM. J9- Java Virtual Machine. http://www-306.ibm.com/software/wireless/weme/. Retrieved on 17-Oct-2005.

[18] IEEE 1394 Trade Association. IEEE 1394 Technology. http://www.1394ta.org. Retrieved on 26-05-2005.

[19] Intel Corporation. Intel UPnP Device Builder. http://www.intel.com/cd/ids/developer/asmo-na/eng/downloads/upnp/overview/index.htm. Accessed on 13-11-2005.

[20] IPv6. IPv6 Forum. http://www.parc.xerox.com/about/default.html. Accessed on 16-06-2004.

[21] J. Allard, V. Chinta, S. Gundala, and G. Richard, III. Jini Meets UPnP: an Architecture for Jini/UPnP Interoperability. In *Proc. of the 2003 Applications and Internet Symposium*, pages 268–275, Jan 2003.

[22] J. Bates, J. Bacon, K. Moody and M. Spiteri. Using Events for the Scalable Federation of Heterogeneous Components. In *Proc. of the 8th ACM SIGOPS European Workshop on Support for Composing Distributed Applications*, pages 58–65, Sep 1998.

[23] J. Newmarch. UPnP Services and Jini Clients. http://jan.netcomp.monash.edu.au /publications/jini_upnp.isng05.pdf. Accessed on 01-08-2005.

[24] K. Raatikainen, H. Christensen and T. Nakajima. Application Requirements for Middleware for Mobile and Pervasive Systems. *SIGMOBILE Mobile Computation Communication Review*, 6(4):16–24, Oct 2002.

[25] Knopflerfish. Knopflerfish OSGi Framework. http://www.knopflerfish.org. Retrieved on 17-10-2004.

[26] M. Galeev. Home Networking with Zigbee. http://www.embedded.com/showArticle.jhtml ?articleID=18902431. Retrieved on 13-07-2004.

[27] Microsoft Research. Easy Living. http://research.microsoft.com/easyliving/#links. Retrieved on 10-01-2005.

[28] N. Shaylor, D. Simon and W. Bush. A Java Virtual Machine Architecture for Very Small Devices. In *Proc. of the 2003 ACM SIGPLAN Conf. on Language, Compiler, and Tool for Embedded Systems*, pages 34–41, Jun 2003.

[29] OSGi Alliance. Open Service Gateway Initiative. http://www.osgi.org. Accessed on 06-05-2004.

[30] OSGi Alliance. OSGi Alliance 2005 World Congress. http://www.osgicongress.com/invitation.asp. Accessed on 28-03-2005.

[31] P. Pietzuch, B. Shand and J. Bacon. Composite Event Detection as a Generic Middleware Extension. *IEEE Networks Magazine, special issue on Middleware Technologies for future Communication Networks*, 18(1):44–55, Jan 2004.

[32] Philips Research. Ambient Intelligence. http://www.research.philips.com/technologies/syst_softw/index.html#ambintel. Accessed on 10-01-2005.

[33] R. Hall. Oscar- OSGi Framework Implementation. http://oscar.objectweb.org. Retrieved on 17-10-2004.

[34] R. Veldema, R. Hofman, R. Bhoedjang and H. Bal. Runtime Optimizations for a Java DSM Implementation. In *Proc. of the 2001 Joint ACM-ISCOPE Conf. on Java Grande*, pages 153–162, Jun 2001.

[35] Reliable Multicast Transport Protocol. RMTP. http://www.bell-labs.com/project/rmtp/. Retrieved on 26-05-2004.

[36] S. Preuss and C. H. Cap. Overview of Spontaneous Networking— Evolving Concepts and Technologies. In *Rostocker Informatik-Berichte*, volume 24, pages 113–123, 2000.

[37] SOAP Consortium. Simple Object Access Protocol. http://www.w3.org/TR/soap. Accessed on 16-08-2004.

[38] Sun Microsystems. Java Communications API. http://java.sun.com/products/javacomm/. Accessed on 28-06-2005.

[39] Sun Microsystems. Java Media Framework. http://java.sun.com/products/java-media/jmf/. Retrieved on 13-04-2005.

[40] Sun Microsystems. Java Security Archeitecture. http://java.sun.com/security/. Accessed on 28-03-2005.

[41] Sun Microsystems. Jini. http://www.jini.org. Accessed on 04-06-2004.

[42] Sun Microsystems. JXTA Project. http://www.jxta.org. Accessed on 07-06-2004.

[43] Sun Microsystems. Service Location Protocol. http://www.playground.sun.com/srvloc/slp_white_paper.html. Retrieved on 14-06-2004.

[44] T. Nakajima. A Middleware Component Supporting Flexible User Interaction for Networked Home Appliances. *SIGARCH Comput. Archit. News*, 29(5):68–75, Dec 2001.

[45] T. Nakajima, K. Fujinami, E. Tokunaga, and H. Ishikawa. Middleware Design Issues for Ubiquitous Computing. In *Proc. of the 3rd International Conference on Mobile and Ubiquitous Multimedia*, pages 55–62, Oct 2004.

[46] T. Nakajima, S. Oikawa, H. Ishikawa, K. Iwasaki, and M. Sugaya. Experiences with Building Distributed Middleware for Home Computing on Commodity Software. In *Proc. of the 3rd International Conference on Mobile and Ubiquitous Multimedia*, pages 424–429, Oct 2004.

[47] The World Wide Web Consortium (W3C). Extensible Markup Language (XML). http://www.w3.org/XML/. Accessed on 28-03-2006.

[48] E. Turcan. Peering the Smart Homes. In *First Int'l Conf. on Peer-to-Peer Computing (P2P'01)*, pages 27–29, Aug 2001.

[49] Universal Plug and Play . UPnP. http://www.upnp.org. Accessed on 26-03-2004.

[50] X10 Protocol. X10 Technical Specifications. http://www.x10.org/aboutx10.html. Accessed on 24-03-2004.

[51] Xerox. The Obje Software Architecture. http://www.parc.xerox.com/research/csl/projects/obje/default.html. Retrieved on 12-06-2004.

[52] Xlinx. White Paper on Home Networking. http://www.xilinx.com/esp/consumer/home_networking/pdf_files/white_papers/wp136.pdf. Retrieved on 04-06-2004.

[53] Y. Goland, T. Cai, P. Leach, Y. Gu and S. Albright. Simple Service Discovery Protocol. http://www.itl.nist.gov/div897/ctg/adl/sdp_projectpage.html, Oct 2002. Retrieved on 22-10-2004.

[54] Y. Rasheed, J. Edwards and C. Tai. Home Interoperability Framework for the Digital Home. *Intel Technology Journal*, 06(04):10–20, 2002.

Appendix A

Jini to UPnP Service Translation

```
 /**
 * ClockClient.java
 */
/*Jini service for clock device */

package client;

import java.rmi.RMISecurityManager; import
java.rmi.RemoteException;

import net.jini.discovery.LookupDiscovery; import
net.jini.discovery.DiscoveryListener; import
net.jini.discovery.DiscoveryEvent; import
net.jini.core.lookup.ServiceRegistrar; import
net.jini.core.lookup.ServiceTemplate;

import jiniupnp.types.StringHolder; import
jiniupnp.service.urn_schemas_upnp_org_service_timer_1;


/**
 * A client to test the UPnP Clock service. This is a service
 * created by CyberLink using the CyberGarage library. It has two methods,
 * GetTime() and SetTime().
 */
public class ClockClient implements DiscoveryListener {
```

```
Class cls = null;

public static void main(String argv[]) {
new ClockClient();

    // stay around long enough to receive replies
    try {
        Thread.currentThread().sleep(100000L);
    } catch(java.lang.InterruptedException e) {
        // do nothing
    }
}

public ClockClient() {
System.setSecurityManager(new RMISecurityManager());

try {
    cls = Class.forName("jiniupnp.service.urn_schemas_upnp_org_service_timer_1");
} catch (ClassNotFoundException e) {
    System.err.println(e.toString());
    System.exit(1);
}

LookupDiscovery discover = null;
    try {
        discover = new LookupDiscovery(LookupDiscovery.ALL_GROUPS);
    } catch(Exception e) {
        System.err.println(e.toString());
        System.exit(1);
    }

    discover.addDiscoveryListener(this);

}

public void discovered(DiscoveryEvent evt) {

    ServiceRegistrar[] registrars = evt.getRegistrars();
Class [] classes = new Class[] {cls};
urn_schemas_upnp_org_service_timer_1 clockService = null;
ServiceTemplate template = new ServiceTemplate(null, classes,
                           null);

    for (int n = 0; n < registrars.length; n++) {
    System.out.println("Lookup service found");
        ServiceRegistrar registrar = registrars[n];
    try {
    clockService = (urn_schemas_upnp_org_service_timer_1)
        registrar.lookup(template);
    } catch(java.rmi.RemoteException e) {
```

```
        e.printStackTrace();
        continue;
        }
        if (clockService == null) {
        System.out.println("ClockService null");
        continue;
        }
        //System.out.println("Service: " + clockService.toString());
        try {
        StringHolder timeResult = new StringHolder();
        clockService.GetTime(timeResult);
        System.out.println(timeResult.getValue());

        // CyberLink hasn't implemented SetTime, just get a stub answer
        String time = "Sun, Jun 15, 04";
        clockService.SetTime(time, timeResult);
        System.out.println("New time " + timeResult.getValue());
        } catch (RemoteException e) {
        System.err.println("Service call failed " + e.toString());
        }

        // success
        System.exit(0);
    }
    }

    public void discarded(DiscoveryEvent evt) {
    // empty
    }
} // ClockClient


/*Translated Jini Service to UPnP*/

<?xml version="1.0" ?> <root
xmlns="urn:schemas-upnp-org:device-1-0">
    <specVersion>
        <major>1</major>
        <minor>0</minor>
    </specVersion>
    <device>
        <deviceType>urn:schemas-upnp-org:device:clock:1</deviceType>
        <friendlyName>CyberGarage Clock Device</friendlyName>
        <manufacturer>CyberGarage</manufacturer>
        <manufacturerURL>http://www.cybergarage.org</manufacturerURL>
        <modelDescription>CyberUPnP Clock Device</modelDescription>
        <modelName>Clock</modelName>
        <modelNumber>1.0</modelNumber>
        <modelURL>http://www.cybergarage.org</modelURL>
        <serialNumber>1234567890</serialNumber>
        <UDN>uuid:cybergarageClockDevice</UDN>
```

```
        <UPC>123456789012</UPC>
        <iconList>
            <icon>
                <mimetype>image/gif</mimetype>
                <width>48</width>
                <height>32</height>
                <depth>8</depth>
                <url>icon.gif</url>
            </icon>
        </iconList>
        <serviceList>
            <service>
                <serviceType>urn:schemas-upnp-org:service:timer:1</serviceType>
                <serviceId>urn:schemas-upnp-org:serviceId:timer:1</serviceId>
                <SCPDURL>/service/timer/description.xml</SCPDURL>
                <controlURL>/service/timer/control</controlURL>
                <eventSubURL>/service/timer/eventSub</eventSubURL>
            </service>
        </serviceList>
        <presentationURL>/presentation</presentationURL>
    </device>
</root>
```