SERVICE DISCOVERY IN A DYNAMIC OFFICE ENVIRONMENT

OOI SOO TECK

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY UNIVERSITY OF MALAYA
KUALA LUMPUR

2007

SERVICE DISCOVERY IN A DYNAMIC OFFICE ENVIRONMENT

OOI SOO TECK

DISSERTATION SUBMITTED IN FULFILMENT OF THE REQUIREMENTS FOR MASTER DEGREE OF COMPUTER SCIENCE

FACULTY OF COMPUTER SCIENCE AND INFORMATION TECHNOLOGY UNIVERSITY OF MALAYA KUALA LUMPUR

2007

ACKNOWLEDGEMENTS

The motivation, encouragement, advice and support provided by many individuals are appreciated in producing this thesis. To those individuals who directly and indirectly involve in finishing this thesis are highly appreciated.

Firstly, I would like to thank my supervisor, Dr. Rosli Salleh who had helped me in supervising this thesis. His guidance, enthusiasm and knowledge have contributed so much in this research work. I would like to express my gratitude to him.

I would like to give my deepest thanks to my parents for their caring support, motivation and being there for me during my most trying moments.

ABSTRACT

The raising of pervasive computing environments of heterogeneous network has provided a major impact to replicate the network-based service discovery technologies in ad hoc networks and mobile networks. The ability to interact and control network devices with different modalities within the home and office environment could be very beneficial to many users. The Service discovery in computers and handheld devices enabled them to interact with one another through heterogeneous wired and wireless networks. These services advertise their existence in a dynamic way and the devices may automatically discover them together with their properties.

Service discoveries were used in home and office environment where the devices in homes and offices can be monitored and controlled over the Internet by handheld devices such as laptop and mobile devices. These devices which are based on different technologies are able to communicate and be discovered over the internet with the service discovery technologies. However, there are few issues that prevent these to be realized due to the complexities of the ad hoc environment.

This thesis focuses and provides a preliminary study and analysis of the service discovery technologies in an ad hoc network along with the requirements that need to be met. The challenges are to communicate devices of different communication protocols in a heterogeneous network and platform and the devices are able to be accessed over wide-area network which currently are poor in support.

An architecture of devices interaction is proposed based on the analysis of requirements. A new service discovery concept for dynamic office network is developed and presented in a scenario where a salesman, bringing his pocket pc into his client office which was enabled with UPnP support, able to discover a printer around and print his product details without any network setup and configuration. The service-oriented OSGi (Open Service Gateway Initiative) framework is proposed as the base platform of the architecture. A prototype is partially deployed on the knopflerfish OSGi framework to evaluate the possibility of the proposed architecture. This prototype demonstrates and satisfies the requirement that services could be discovered over different networks and communication platform such as UPnP and OSGi.

TABLE OF CONTENTS

CHAPTER 4 PROPOSED ARCHITECTURE DESIGN

CHAPTER 5 IMPLEMENTATION AND EVALUATION

LIST OF FIGURES

LIST OF TABLES

ABBREVIATIONS

- API - Application Programming Interface
- DHCP - Dynamic Host Configuration Protocol
- DMP - Device Message Protocol
- DNS - Domain Name System
- GENA - General Event Notification Architecture
- GPRS - General Packet Radio Service
- GW - Gateway
- HTML - Hypertext Markup Language
- HTTP - Hypertext Transfer Protocol
- HTTPMU - HTTP Multicast over UDP
- HTTPU - HTTP Unicast over UDP
- IrDA - Infrared Data Association
- ISM - Industrial, Scientific, Medical
- J2SE - Java 2 Platform, Standard Edition
- JLS - Jini Lookup Service
- JVM - Java Virtual Machine
- L2CAP - Logical Link Control and Adaptation Protocol
- LMP - Link Manager
- LAN - Local Area Network
- OBEX - Object Exchange
- OSGi - Open Service Gateway Initiative
- OSI - Open Systems Interconnection
- PAN - Personal Area Network
- PDU - Protocol Data Unit
- PPP - Point-to-Point Protocol
- RF - Radio Frequency
- RFCOMM - Radio Frequency Communication
- RPC - Remote Procedure Call
- SD - Service Discovery
- SDDB - Service Discovery Database

- SDK - Software Development Kit
- SDP - Service Discovery Protocol
- SIG - Special Interest Group
- SIP - Session Initiation Protocol
- SLM - Salutation Manager
- SLP - Service Location Protocol
- SOA - Service-Oriented Architecture
- SOAP - Simple Object Access Protocol
- SSDP - Simple Service Discovery Protocol
- UDP - User Datagram Protocol
- UPnP - Universal Plug and Play
- URL - Uniform Resource Location
- UUID - Universal Unique Identifier
- WAN - Wide Area Network
- WAP - Wireless Application Protocol
- XML - Extensible Markup Language

CHAPTER 1 INTRODUCTION

1.1 Introduction

In this new modern era, the usage of various gadgets which equipped with computing power, computer devices and network services at our homes or in offices becomes essential to facilitate our daily tasks. Besides classical services, such as printers, scanners, fax machines, there are others services at home such as television, air condition, radio and etc. This can be expressed as "ubiquitous computing". This term coined by Mark Weiser in 1988 [1], means that computers are present everywhere in our daily environment.

Users should be able to choose and make use of the services that are available to them. Ideally, they would like to obtain access to the right services immediately, without requiring them to reconfigure their device. This function should not be noticeable by the user. He should be able to interact with and manage all the devices and services whenever required without any difficulties. Wireless communication with service discovery protocol is important to achieve this goal because user prefers something that is user friendly.

In future, the ability to control home or office devices and appliances remotely using a single user device will be an advantage to user. In home environment, Infrared (IR) remote control is the traditional way to control devices wirelessly, while current technologies often use radio frequency in wireless communication. Radio frequency in remote control does not require line of sight to work and it is able to work both ways communication compare with IR. Communication of devices in home environment become more advances with the user interaction technology such as voice recognition and interaction, and hand gesture. Many sophisticated products and protocols are designed to solve this dilemma in home and enterprise environments.

Service discovery is also relevant in the car environment [2]. Passengers could bring enabled devices such as mobile phone, pocket pc and laptop into the car, connect them to the car area network and use the equipment that is installed inside the car such as CD player or amplifier. Service discovery also play an essential role in ad hoc communications. The mobile phone, pocket pc and laptop could form an ad hoc network via Bluetooth links. In such network without administrative control, the device must be self organizing. For example, the laptop

may offer a translation service to mobile phone and the phone may offer internet access service via its General Packet Radio Service (GPRS) interface. Since this network is dynamic, there is a need of dynamic and automatic service discovery functionality.

However, the existing service discoveries are limited to their own domains and specific devices. Therefore, a service discovery solution to interoperate over different network technologies and platforms is needed. This thesis presents a proposed solution of service discovery over heterogeneous network.

In this thesis, a service discovery concept is presented in a scenario where a salesman bringing his pocket pc which was enabled with UPnP support into his client office. The pocket pc is able to discover printer service in the office and print the product details without any network setup and configuration to be done. To implement this concept, there are few issues which require our consideration:

- Service Discovery - Allows user to discover services within the range of the network
- Self-organization - Devices in the office network are able to be organized together with trusted connection.

1.2 Problem Definition and Motivation

There are few different service discovery technologies exist such as Bluetooth SDP, Jini, Salutation, SLP and UPnP. Each service discovery technology forms a domain on a network where the services of the same technology are available for clients of the same technology or network [3]. However, there is a need for service discovery solution to interoperate over different network technologies and platforms. If service discovery solutions are not deployed on a multiple technology platform in a network, services which are available in a domain are not visible to clients of the other domains. This problem becomes more obvious in an ad hoc network.

The problem with current solution is that those service discoveries are restricted to specific device, service or technology. It is necessary to have solution to enable multiple services to be discovered over multiple communication networks. This should be allowed to be done in home, office or vehicle network regardless of what type of underlying service discovery technology. A resident gateway is needed in practice to integrate the communication.

There is a need to bring about a change in paradigm from dynamic discovery being purely device-based to a service-based approach. This service referred to applications developed using different computing models. Several directory-based discovery services are built around the publish-subscribe model where the services publish their interfaces in a central directory and clients discover the services by contacting the directories. However, almost all these directory services do not assume the presence of other models and is limited to the underlying hardware platform. The problem that needs to be solved is to provide a service discovery model that is dynamic and encompasses services developed in diverse distributed computing models.

1.3 Objectives

The objectives of this thesis are as follows:-

- To propose and illustrate an architecture design of interaction of devices and services in heterogeneous environment.

- To specify the designed of service access scenario and a service discovery concept for an office network. This service discovery is able to provide service availability and service information to the requesting device. The process of service discovery should be completely unknown to the user.

- To demonstrate a simulation of service discovery over different technologies, Universal Plug and Play (UPnP) and OSGI platform as the proposed service discovery model.

1.4 Thesis Scopes

The project scopes help to focus on the important part of the system implementation of this thesis. The scopes of this thesis are to:

- Illustrate the architecture of the service gateway which runs on OSGi platform.
- Focus on the service discovery over different technologies, which are UPnP and OSGi platform.
- Run a simulation of an UPnP device to discover another UPnP service and a simulation of a OSGi device to discover an UPnP service.

After the scopes have been determined, the expected outcomes of the result were produced. Not only the device in the UPnP network is able to discover a service in UPnP network but also a device in the OSGi platform is able to discover services in the UPnP network. Figure 1.1 shows overview of the implementation scope.

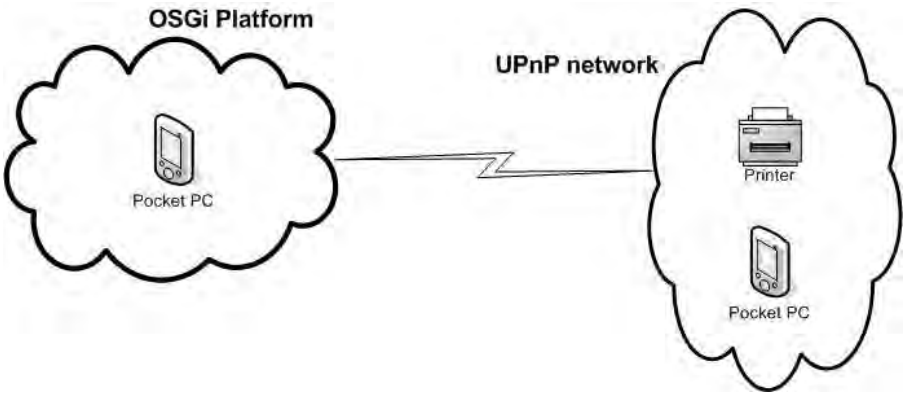

Figure 1.1 Overview of the thesis scope

1.5 Research Methodology

In order to achieve the objectives of the thesis and the solution to the problem statements, the following methodology is adopted. First, a study of the current related technology was carried out in three parts. They are the wide-area communication technology, gateway server technology and the service discovery technology. Technology for the communication in the wide-area was studied and two protocols were analyzed. Then, the suitable gateway designs and technologies were studied and selected. OSGi and UPnP were discussed in Section 2.4. The discovery technologies reviewed are the Bluetooth Service Discovery Protocol, Jini, Salutation and Service Location Protocol.

Secondly, an architecture design of the service discovery was proposed according to the requirements to meet the objectives. The design is used to illustrate the scenario of service discovery in a dynamic office environment. Thirdly, some studies were carried out on the tools to carry out the implementation. It was taken into account already at the time of the literature search in order to find the right tools and interesting technologies for the prototype implementation. Then, an implementation of the service discovery was carried out in a simulated environment and some tests were conducted. Finally, the results are evaluated.

1.6 Thesis Organization

This thesis is organized as follows:

Chapter 1 begins with introduction of service discovery technology and its limitation. It also briefly introduced the reasons for proposing this research.

Chapter 2 explains the related technologies such as service discovery technologies, gateway server, and wide-area technologies that are able to meet the defined requirement.

Chapter 3 illustrates the overview of the scenario, functional and non-functional requirement for proposed architecture design.

Chapter 4 describes the proposed architecture design of service discovery in a heterogeneous environment and the solution to the presented scenario.

Chapter 5 presents the implementation and evaluation of the scenario in a simulated environment to fulfill the concept of the proposed architecture design.

Chapter 6 summarizes the conclusion and achievement of this research and discusses future enhancement.

CHAPTER 2 LITERATURE REVIEW

2.1 Introduction

Traditionally, a network consists of electronic devices such as computers, printers, and scanners. These devices are linked together with fixed local area network. The computer would have a static network configuration or would have one using the DHCP protocol. Once properly configured the computer would own a valid network address and able to know and reach the available services.

In the present time, this is still true for many networks, but new highly dynamic environments have appeared. A networked electronic device can be as well a mobile phone, PDA, laptop or home devices. This could form a dynamic wireless ad hoc network to discover available services.

Network devices should be able to advertise their services in the network while the user device should be able to search for the available services. A user is not too interested in the underlying protocols used to communicate with networked devices. What the user concern are the services provided by the devices. The networked devices should be capable of advertising their services automatically due to the lack of administration in home and car network environment. In general, service discovery protocols function by advertising the available service and providing information about the details and capabilities of the services requested. After discovering the service, the user will register to use that service.

Some of the possible scenarios for ad hoc networks are as follows:

- The ability to discover all nearby restaurant and make food ordering using a wireless mobile device.
- The ability to discover and use a nearby printer while in a hotel using a wireless laptop.
- The ability to discover and share entertainment resources such as music and photos on handheld device.

There are some discovery protocols and products that provide service discovery mechanisms in pervasive computing environments. Service discovery of available services becomes an important requirement in such environments. The goal of all service discovery protocols provider is to find needed services dynamically without manual configuration. Some of the most important service discovery protocols including Bluetooth Service Discovery Protocol (SDP)[6], Jini[7], Salutation[8], Secure Discovery Service (SSDS)[9], Service Location Protocol (SLP)[10] and Universal Plug and Play (UPnP)[11]. They all allow a client to find a server that provides the sought service. A client can be any device or software application and a service can be any hardware equipment, function or application server. UPnP provide discovery which targets in home network environments while Bluetooth SDP allows a Bluetooth enabled device to discover services of another Bluetooth enabled device. SLP and JINI provide discovery mostly for enterprise environments while both UPnP and Salutation are device oriented protocols.

This chapter will discuss the meaning of services, the technologies for wide area network, the technologies for resident gateway and the prominent discovery protocols. .

2.2 The Internet alarm clock

Many studies have been carried out for several years on the idea of home devices communication through a network. Telcordia Technologies has studied the market for this type of devices and was one of the first to propose the idea of networked appliances. Their first case study was on an Internet alarm clock. This study was based on the implications of the networked devices had on the network design and requirement.

Telcordia Technologies identified two features that could be added to a traditional alarm clock [39]. One of the improvements was on configuring rules on the alarm clock which the device would use to program itself given a target time. For instance, the alarm clock programs itself to manage the time given by the user for the user to wakeup, get dressed, take breakfast and drive to office, if the user wants to reach work at a given time. The second improvement was by providing external network connectivity to the alarm clock to use external factors to calculate the suitable wake up time besides the configured rules.

The alarm clock was created with an Internet interface board with clock driver and clock controller and a LCD display. The alarm clock was connected to web servers. These web servers which contain user profile are used to interpret the service rules. The user just need to key in the time for them to reach office, the system will then calculate the time to set the alarm depend on the external condition such as traveling distance and weather.

Figure 2.1 below shows the entire network communication sequence to support the transaction [39].

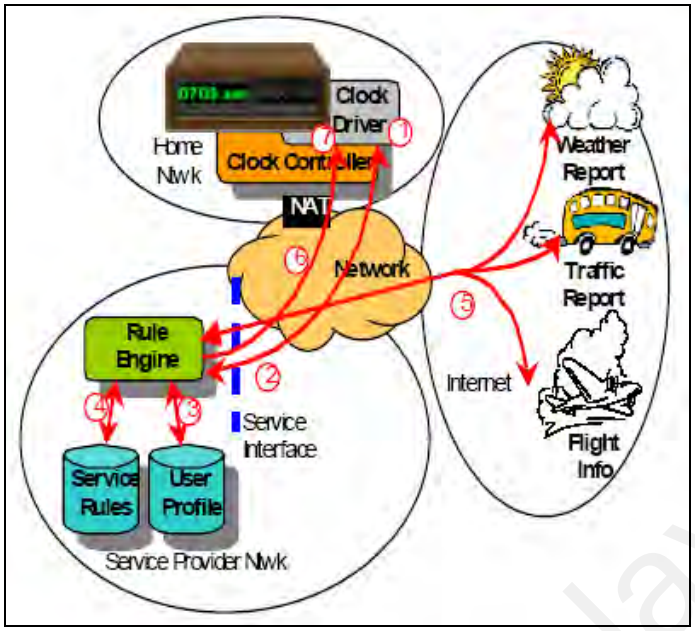

Figure 2.1 Internet Alarm Clock Functional Architecture [39]

1. User presses one of the buttons on the clock to start the transaction. The Clock Driver opens a communication session with the Clock Controller. Then, the driver and controller interact with the user via the buttons and display.
2. The Clock Controller requests the Rule Engine for the default Alarm Time and Check Time of event that the user selected.
3. The Rule Engine retrieves the user profile which contains all users' alarm rules.
4. The Rule Engine retrieves service rule information and inserts user rule where appropriate.
5. The Rule Engine calculates the Alarm Time and Check Time by executing the service rules. Execution of these rules will issue in HTTP request to the web service.
6. The Rule Engine returns the calculated Alarm Time and Check Time to the Clock Controller.
7. The Clock Controller sets the Alarm Time and Check Time on the Clock Driver. Then, the clock display the current time and date.

2.3 Smart Home Network

The diagram below describes a high-level view of a smart home network.

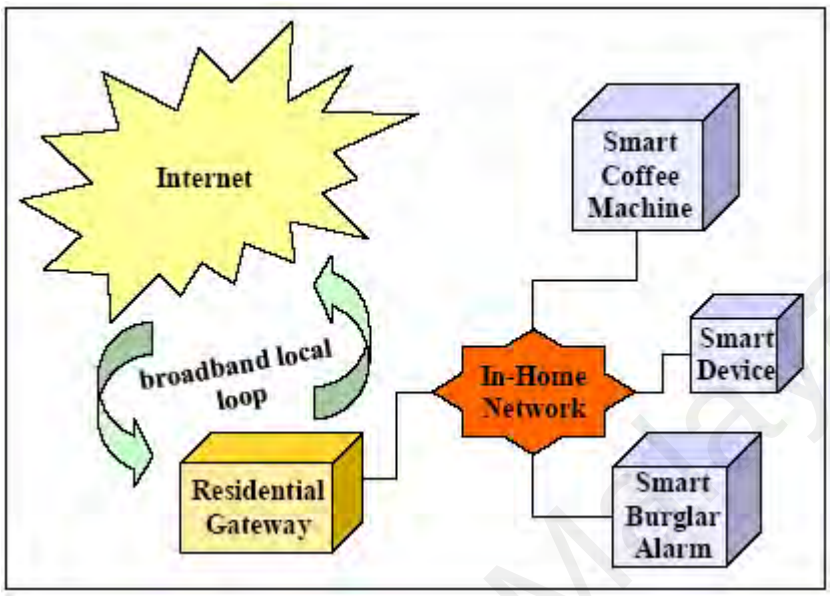

Figure 2.2 High-level View of Home Networking Architecture [58]

This architecture has an interaction between the Internet, a broadband local loop and the in-home network. In the whole network, there is a residential gateway that links the outside with the network of smart devices or appliances. The broadband local loop is provided by technologies that give a permanent connection such as cable, satellite or xDSL. The residential gateway with the OSGi specification has the LAN and WAN interfaces. The important factor to use this architecture is the increase of the number of homes with broadband access. The very obvious benefit of a home network will be to solve the problem of sharing the Internet connection and the diverse computer peripherals in a household. There are other benefits when others devices are shared in the home network.

There are other candidate technologies for the in-home network. There is the possibility to mix those different home networks to get the complete network, a multi-layered home network [58]. The house requires to be rewired with category 5 UTP (Unshielded Twisted Pair) cable. High-speed powerline and phoneline technologies can act as a backbone network. The wireless and Radio Frequency technologies, HomeRF, Bluetooth, IEEE 802.11b could solve the ubiquity requirement and be considered as a “mobility network layer” [58]. Another type of layer should consist of the low-speed powerline automation

technologies such as X10. This X10 technology communicates between transmitters and receivers by sending and receiving signal. These signals involve short RF (Radio Frequency) bursts which represent digital information. This is considered a low-speed control network. Technologies such as infrared-based technology, old European home automation network (Batibus), Japanese evolutionary system are technologies for controlling small appliances not able to support a TCP/IP stack [58].

HAVi is another technology, which is based on the IEEE 1394 standard. It is also called FireWire. HAVi is dedicated toward entertainment systems and high-level functionalities. All this technologies rely on different service discovery protocols. The architecture above presented a framework for a home network and they encompass more than a stand-alone discovery protocol.

2.4 Introduction to Services

Services are ubiquitous and there are many services use in our daily environment. These services can be for instance physical services, social services and electronic services. There are many different definitions for the term *service* [12, 13, 14]. *"A service is a software entity that performs an action on behalf of another entity."* [12].

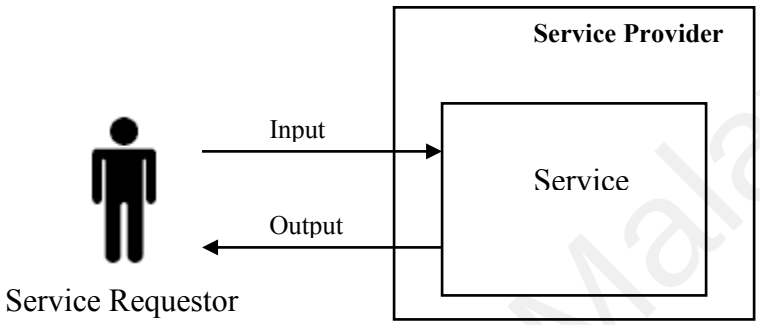

Figure 2.3 Service

A Service has functional properties such as the person who perform the action and the action it performs. Refer to Figure 2.3. It has non-functional properties such as cost, performance, quality of service and security. These properties can be used to describe a service. A service is also an entity provided by a Service Provider. It performs an action (input) on behalf of a Service Requestor and provides a result (output).

A service has an operational lifecycle which consists of three phases as shown in figure 2.4. These phases are Advertisement, Discovery and Delivery. The service requestor and service provider are unassociated with each other before the delivery phase. In the advertisement phase, the service provider creates a service description to advertise the service. This description is based on the service properties. In the discovery phase, the requestor searches for service that satisfies his need. When a service is found, it is provided in the delivery phase. In the delivery phase, the service requestor and the service provider are associated with each other.

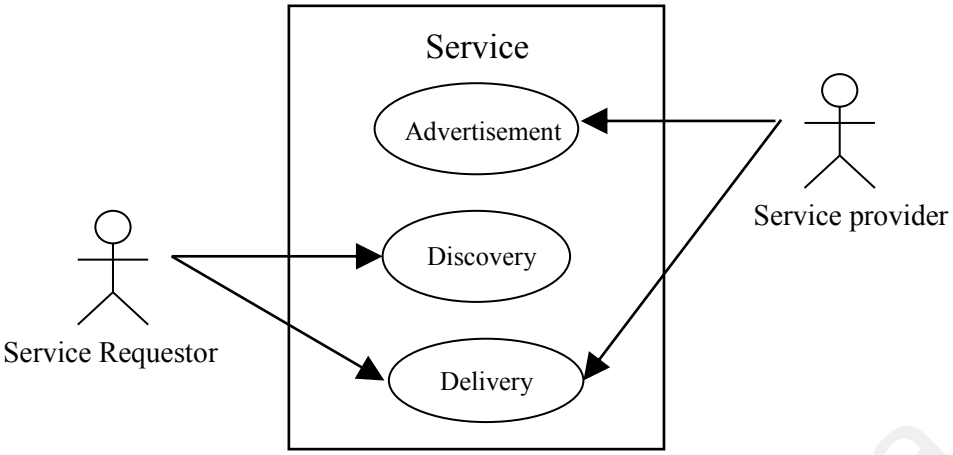

Figure 2.4 Service Lifecycle

2.4.1 Background on Service Discovery

A service has to be found before it can be used. This is often difficult for a service requestor because services are distributed and heterogeneous. Services are offered by different providers using different operating systems and transportation technologies.

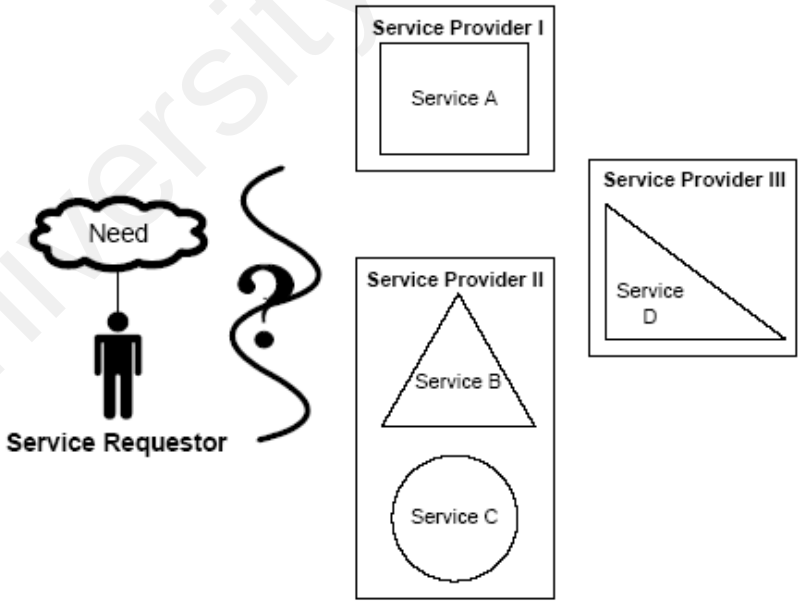

Figure 2.5 Distributed and Heterogeneous Services [4]

There is no obvious link between the service provider and service requestor. Remote Procedure Calls (RPC) may be one of the foundations of the vast majority of middleware platform. It was introduced in the early 1980's and provided a way to transparently call procedures located on other machines [4]. First binding has to be established before an RPC can be made. The association is hard-coded with static bindings. It is a simple and efficient mechanism and it is tightly couples between server and client. However the flexibility is reduced. To overcome this issue, naming and directory servers were created to enable client to dynamically locate service location.

2.4.2 Processes in Service Discovery

Service discovery phase consists of three sub-processes. They are the Discovery request handling, Matchmaking and Discovery result handling. Figure 2.6 shows these sub-processes.

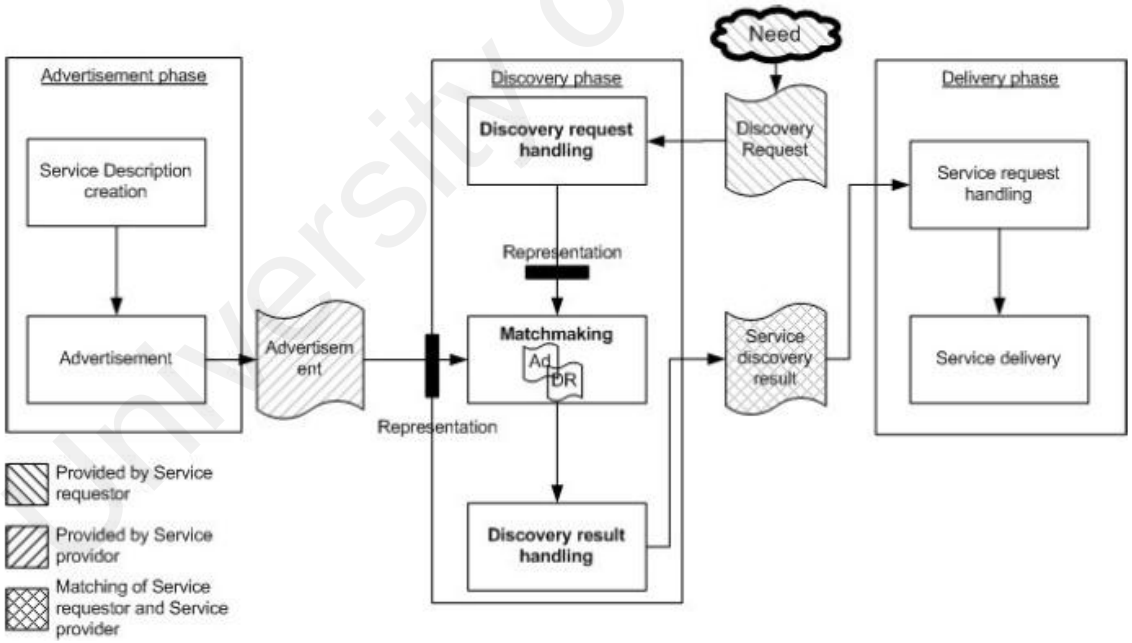

Figure 2.6: Service discovery [4]

- ***Discovery Request Handling***

  This process performs knowledge acquisition. It retrieves request of the service requestor and formats it so that it can be used by the matchmaking process. In this

step, contextual information like location and time of request, that cannot be handled are ignored.

- ***Matchmaking***

  The matchmaking process compares service advertisements with a discovery request and tries to match them. In this process, the most accurate match between information from service provider and service requestor has to be provided.

- ***Discovery Result Handling***

  This process transfers the matchmaking result to the service requestor so that it can associate with the service provider.

2.5 Wide-area Communication Technologies

Wide-area communication is needed to communicate devices over the Internet. There are two most common solutions for communicating devices over the wide-area network. They are the Session Initiation Protocol (SIP) and Hypertext transfer protocol (HTTP) [56, 57]. The following section explained the two protocols.

2.5.1 Session Initiation Protocol (SIP)

The Internet Engineering Task Force (IETF) defines the Session Initiation Protocol (SIP) as *An application-layer control that can establish, modify and terminate multimedia sessions such as Internet telephony calls* [40].

SIP works independently of the transport protocol and type of session established. It supports name mapping and redirection services, which supports personal mobility [40]. SIP should be used together with other protocols in order to provide complete services to the user.

SIP can works with existing protocols by enabling user agents to discover other user agents and agree in the type of session they would like to share. The type of session is specified using the Session Description Protocol. Session Description Protocol is not a true protocol but a text-based description language. A user agent is the entity implemented on single device which can be a SIP client or SIP server.

SIP provides primitives that may be used to implement different services. A single primitive is typically used to provide several services depending on the type of session being established. The facets used to establish and terminating multimedia communication are as below [40].

- ***User location*** - Determination of the end system to be used for communication
- ***User availability*** - Determination of the willingness of the called party to engage in communications
- ***User capabilities*** - Determination of the media and media parameters to be used
- ***Session setup*** - Establishment of session parameters at both parties
- ***Session management*** - Including transfer and termination of sessions, modifying session parameters, and invoking services

2.5.1.1 SIP Messages

SIP is a text-based protocol with messages request and response. A SIP transaction consists of one request, one or more provisional responses and a final response. SIP requests contain field called method. IETF defines six types of requests.

- ***INVITE*** - Invite other user to participate in a session. It contains the description of the session.
- ***ACK*** - Provides three-way handshake between user agents by acknowledge the reception of a final response to an INVITE
- ***CANCEL*** - Cancels pending transactions
- ***BYE*** - Terminate a session between two parties.
- ***REGISTER*** - Indicate a SIP server the current location of the user agent.
- ***OPTIONS*** - Query a server about which methods and which session description protocols it supports as well as other capabilities.

SIP request and responses contain headers that provide information. Below are some of the headers.

- ***From*** - Request sender with SIP address
- ***To*** - Recipient of the request
- ***Cseq*** - Command Sequence header consists of a numerical and a method name.
- ***Via*** - Routing mechanism.
- ***Content-Type*** - Provide information about the message contained in the body of SIP message.
- ***Content-Length*** - Provide the length in bytes of the SIP message body.

The body of the SIP messages depends on the application. This type of flexibility makes SIP an ideal protocol for controlling networked devices.

2.5.1.2 SIP Extension

The methods mentioned above are the foundation of SIP, but none of them has the capability of carrying control messages to a networked device. To allow SIP to be used in the networked devices it has been enhanced with the following modifications and extensions. The extended version of SIP, designed for communicating with network devices [41]. There are few methods added to Extended SIP. They are DO, SUBSCRIBE and NOTIFY methods. Messages or requests for networked devices are carried in the body of the DO request and are delivered to the PAN environment [42]. SUBSCRIBE enabled user devices to subscribe to certain events within the network and NOTIFY allows devices to notify subscribers of events occurring. The interaction of devices using the extended SIP solution is presented in [44, 45]. LOCK and UNLOCK are to allow devices to interact with each other without interference. These methods are useful for home alarm system.

An XML based format, Device Message Protocol (DMP) is a suitable format for the message body. XML is independent of the transport protocol and it's important for communicating networked devices.

The advantages of using SIP for wide-area communication are presented below [43].

- ***Simplicity*** - New services can be deployed easily by service providers using SIP.

- ***Scalability*** - SIP works well in LAN and WAN conditions and over variety of transport protocols such as UDP and TCP.
- ***Flexibility*** - SIP allows extensions to be added to support new features. The protocol is defined in a way that any providers can easily defined extensions to the existing grammar to add features. SIP specification mechanisms ensure that any new extension does not break an existing SIP aware node.
- ***Registration*** - A device can register its current location with its registrar. The exact location will be resolved by the proxy in conjunction with the registrar.
- ***Personal Mobility*** - SIP provides the concept of personal mobility at no extra cost.
- ***Security*** - SIP able to perform authentication and encryption using schemes such as Pretty Good Privacy (PGP) for security purpose.
- ***Transport Independence*** - SIP messages can be sent through various heterogeneous networks.
- ***Event notification*** - Extended SIP introduce SUBSCRIBE and NOTIFY methods which enable entities to subscribe to certain events and to be notified when occur.
- ***Addressing*** - SIP uses The Uniform Resource Identifier (URI) addressing scheme which can encompass wide range of addressing requirements.
- ***Integration with existing SIP service mechanisms*** - SIP allows networked devices to exploit the rich infrastructure that SIP provides
- ***Session based and non-session based communication*** - SIP provides session based and non-session based communication which is ideal for networked devices with different types of services.

2.5.2 HyperText Transfer Protocol (HTTP)

Hypertext Transfer Protocol (HTTP) is a ubiquitous protocol for data connections between Web browsers and servers. This protocol is the current standard for transferring HTML documents, although it is designed to be extensible to almost any document format like XML. HTTP version 1.1 is documented in RFC 2068 [46]. It operates using port 80 over TCP connection. When the client request, a message is sent to the server, and a reply message is send back. HTTP has some security features such as Secure Sockets Layer (SSL)

There are few types of HTTP request like GET, HEAD and POST request. A client sends a GET request for a specific document to the server. If the server does not respond, it is up to the client to wait for the timeout and request the same document again. POST request are used for HTML forms and other operations that require the client to transmit a block of data to the server.

However, HTTP is not suitable for networked devices communication because HTTP does not provide good support for mobility or notifications and HTTP must run over TCP, and a TCP stack is larger and more complex than a UDP stack. This can be an issue in small devices with stringent memory and processing power.

2.6 Gateway Server

Middleware is a software layer that stands between the networked operating system and the application, providing well-known, reusable solutions to encountered problems like heterogeneity, interoperability, security and dependability. Middleware were introduced in the early 90s were based on the object-oriented infrastructures (eg. CORBA, Java-RMI and DCOM). Using object-based middleware infrastructure, distributed application and higher-level middleware functionalities may be developed in terms of distributed objects.
Although this significantly eases the development of distributed applications, developers still need the solutions for the enforcement of non-functional properties like dependability and persistence management.

Thus, this led to enforce of non-functional properties like transaction and security management. Middleware technologies have further evolved towards service-oriented computing in early 2000 to support open distributed application over the Internet. This allows applications as services to be accessed with other applications over the Internet.

This section introduces the middleware architectures for the office environment, addressing based service oriented architectures, discovery protocols and their interoperability. It will provides an overview of software technologies complying with Service-Oriented Architecture (SOA) such as OSGi, and UPnP architecture.

2.6.1 Open Service Gateway Initiative (OSGi)

The OSGi Alliance [30] is an open standard organization formed by Sun Microsystems, International Business Machines, Ericsson, IBM and others in March 1999. For past few years, it has specified a Java programming language-based service platform that can be remotely managed. The core part of the specifications is a framework that defines an application life cycle model and a service registry. Based on this framework in figure 2.7, a large number of OSGi services have been defined: Log, Configuration Management, Preferences, HTTP Service running on servlets, XML Parsing, Device Access, Package Admin, Permission Admin, Start Level, User Admin, Jini, and UPnP. Specifications are developed by the members in an open process and made available to the public free of charge and without licensing conditions. The OSGi Alliance has a compliance program that is open to members only. Currently the available specifications are R1, R2, R3 and R4.

The original focus of OSGi was on Service Gateways. In 2003, the well known Eclipse IDE selected OSGi as the underlying runtime for their plug-in architecture. The Equinox project experimented with this idea, and built the runtime for Eclipse R3 that has been available since December 2003 [30]. In October 2003, Nokia, Motorola and other OSGi members formed a Mobile Expert Group (MEG) that will specify a service platform for the next generation of smart mobile phones, addressing some of the needs that MIDP cannot manage [30]. The application areas of the OSGi Service Platform are currently as various as service gateways, cars, mobile telephony, industrial automation, building automation, PDAs, grid computing, entertainment, and IDEs.

OSGi has established a set of principles, which are guidelines through the planning and definition processes of the standard. The principles [24] are as follow:

- ***Platform independence*** - OSGi platform can be implemented in various different platforms.
- ***Application independence*** - The capabilities of the available software can be used in any computing environment through OSGi platform.
- ***Multiple service support*** - OSGi is capable of hosting different applications offered by any service provider.

- ***Service collaboration support*** - Services implemented on OSGi platform are able to adapt, cooperate with and find other available services dynamically. This guarantees easy expandability of the system.
- ***Security*** - Because OSGi supports concurrent execution of applications offered by different service providers, taking care of security issues between the services is an important task.
- ***Multiple network technology support*** - OSGi supports several networking technologies both at Local Area Network (LAN) and at Wide Area Network (WAN) levels.
- ***Simplicity*** - Service providers and gateway operator handle the most of the complexity and the administration of the platform.

Networking at homes and offices is steadily speeding up. More personal computers are being interconnected in the home. Also home appliances and devices, such as televisions, air-conditional, refrigerators, alarm systems, electricity meters and lighting are interconnected and control over the Internet. Such a smart home environment that contains the technology that allows devices and systems to be controlled automatically and remotely, has been hot in the research community [25]. There is a need of some kind of coordination due to the variety of the home appliances, devices and network technologies. These different architectures can be coordinated via a central control point or a gateway. OSGi provides a specification for such service delivery.

2.6.1.1 OSGi Framework

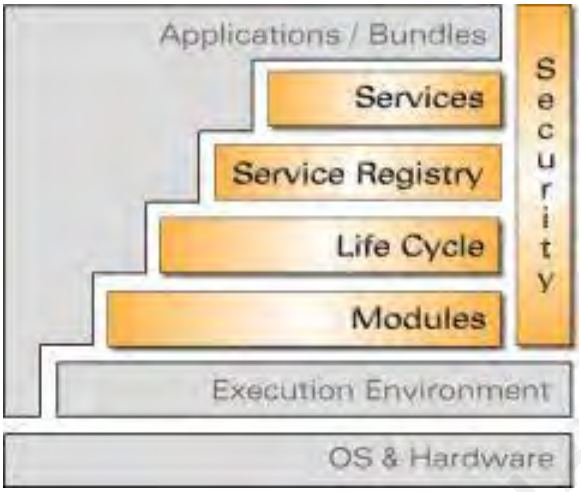

Figure 2.7 The OSGi Framework [30]

The OSGi organization is the leading standard for next generation Internet services to home, cars, mobile phones, desktops, small offices, and other environments. The OSGi framework forms the core of the OSGi Service Platform Specifications. OSGi provides a general-purpose, secure, and managed Java framework that supports the deployment of extensible and downloadable applications known as bundles. OSGi-compliant devices can download and install OSGi bundles. The framework manages the installation and update of bundles in an OSGi environment in a dynamic and scalable way.

OSGi framework manages the dependencies between bundles and services in detail. It provides the bundle developer with the necessary resources of Java's platform independence and dynamic code-loading capability. This is for them to easily develop services for small-memory devices that deployed in a large scale.

2.6.1.2 OSGi Architecture

OSGi architecture [26] and related standards are shown in figure 2.8. OSGi gateway has obviously a key role in the architecture. It acts as a central point managed by the gateway operator between local area and wide area networks. It is capable of supporting and integrating variety of services. One of its main functions is to operate as an execution environment for the services offered by service providers.

Service platform [24] is an application server, which enables delivery of services much better than normal plain Web browser could offer for the client. It acts as an execution environment and service platform to facilitate operation of services for different service providers in the customer home local networks. Interoperability of the services is also made possible by the service gateway. OSGi service platform contains a specification [23] for a service framework, which is the core of the environment. Specification portrayed APIs that address life cycle management, services interoperability, data management, device management, client access, resource management and security.

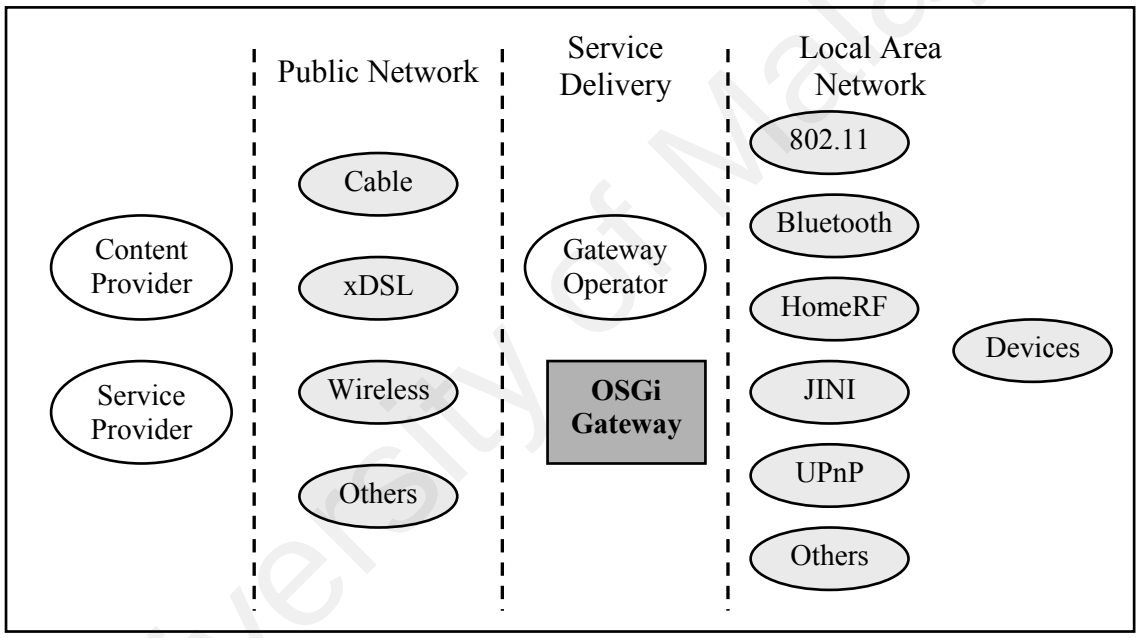

Figure 2.8 OSGi Architecture and Related Standards [29]

2.6.1.3 OSGi Bundles

A bundle is a Java JAR archive including Java class files and any other necessary data the service might need, including possibly native code. A special *Manifest File* provides information about resources needed or provided by the bundle. Bundles contain an *Activator* which allows the bundle to be started and stopped as an application. A bundle without an *Activator* must be considered as a library that provides list of packages. OSGi specifies a strict class loading policy so that classes provided by a bundle cannot be used in another bundle unless specified in the import-export clauses of both bundles. Life cycle management

is one of the most prominent features of the OSGi framework. It provides the necessary mechanisms to allow remote management in a wide variety of management models and is also accessible via an API that allows bundles to manage other bundles.

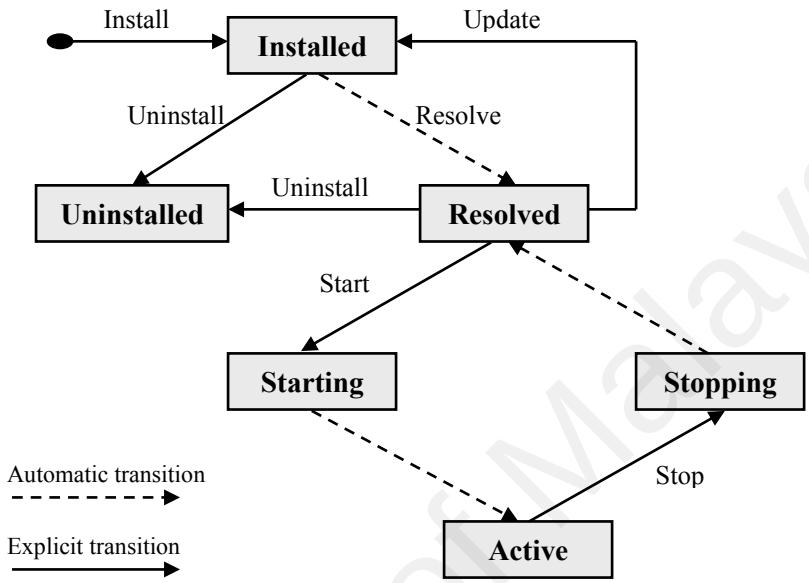

Figure 2.9 The OSGi Bundle Life Cycle [29]

2.6.1.4 The Bundle Life Cycle

A bundle has a life cycle, where it can be installed, activated, updated, deactivated and uninstalled. Figure 2.9 shows the OSGi bundle life cycle. The steps of the life cycle are as below [50]:

1. Installed - The framework reads the contents of the bundle and assigns it a bundle ID. Framework also caches its location and state persistently. A dedicated class loader is created to access the bundle's resources.
2. Activated - The framework checks whether the Java classes required by the bundle have been exported by other bundles. If yes, the framework calls the start method and registers the bundle's services. Then, the service will start to run.
3. Deactivated - The framework calls the bundle's stop method and unregisters the service. Then the service will stop.
4. Uninstalled - The bundle is removed from the framework.

If an error occurs during the life cycle, the framework throws a *BundleException*, a standard exception defined by OSGi specification in *org.osgi.framework*.

2.6.1.5 Types of OSGi

Oscar [31] is an open source (GPL) implementation of the OSGi framework specification, currently compliant with a large portion of the R3 specifications. It proposes few interesting features like *Service Binder* and the *Oscar Bundle Repository.*

The Knopflerfish project [32] is based on the Gatespace GDSP OSGi framework which has been in development since 1999. Knopflerfish is absolutely R3 compliant. Moreover, it comes with a visual desktop, which allows the management of the whole framework. Knopflerfish is available under a BSD style license.

Physalis [33] is an OSGi implementation for the .NET (Compact) framework. This project started its activities in August 2004, so the status is yet pre-alpha and there is no documentation or download available. However, this is the fact of being the first open source OSGi implementation for the .NET framework.

JEFFREE [34] stand for Java Embedded Framework Free. The last version 0.91 was released on 3 March 2003 provides almost complete OSGi v2.0 specification. An interesting feature is that JEFFREE is compatible with personal Java, whereas Oscar and Knopflerfish require Java 2. This allows JEFFREE to be installed on a large variety of devices. However, the project development seems to be stopped for the moment. JEFFREE is available under the open source license.

2.6.1.6 Interoperability with other devices

The OSGi Service Platform and the UPnP Device Architecture specification provide complementary functionalities. The OSGi Service Platform specifies an execution point and does not define what protocols or media are supported. In contrast, the UPnP Device Architecture specification is a data communication protocol that does not specify where and how programs execute. That choice is made by the implementations. The UPnP specification and the OSGi specifications are fully complementary and do not overlap. It is the same with

JINI and OSGi. OSGi release R3 defines standard ways of incorporation UPnP and JINI technologies on OSGi platforms.

For example, the JINI Driver module in the OSGi framework is responsible for the JINI-to-OSGi and OSGi-to-JINI communication. Using this API, OSGi services from the framework can be exported easily to the JINI network and JINI services from the JINI network can be imported into the OSGi framework.

Concerning UPnP, the OSGi framework defines how an OSGi bundle can implement a service that is exported to the network via the UPnP protocols and how to discover and control UPnP devices that are available on the local network.

2.6.1.7 Knopflerfish OSGi

Knopflerfish OSGi [32] and Oscar OSGi are the open source active project OSGi implementations with the same goals. Knopflerfish was chosen to in this thesis because it is more actively developed and supported by Gatespace Telematics Inc. Gatespace Telematics released a product called Ubiserv. The business concept behind Ubiserv is to make the freely available Knopflerfish a fully supported product. Gatespace was previously selling its own OSGi implementation, but now they provide a better concept of combining open source software and commercial support services.

Knopflerfish includes the OSGi framework, the standard OSGi services and additional components, such as graphical desktop and console utilities. The components that are implemented in the OSGi R3 specification have passed in some tests defined by the OSGi Alliance. However, Knopflerfish cannot claim to be OSGi R3 certified, because that is granted only for the OSGi Alliance members. Some test was done shown that it is stable and compliant with OSGi specification. Therefore, Knopflerfish OSGi is chosen to be used in this implementation.

2.6.2 Universal Plug and Play (UPnP)

Universal Plug and Play (UPnP) [20] is a technology and industry consortium driven by the UPnP Forum, including more than 700 companies from consumer electronics, computing, home automation, home appliances and related industries. The UPnP Forum defines UPnP Device and Service Descriptions, which are based on open Internet-based communication standards for interoperability, according to a common Device Architecture contributed by Microsoft. Similar to the Internet, UPnP is based on wire protocols, not APIs. Therefore, it is independent of the underlying OS, programming language and physical medium.

UPnP enables devices to join automatically to a network, find and use networked devices and services provided by one another without manual configuration. It describes and supports discovery and communication between devices to find and use services. The communication specifically focuses on PCs, Internet gateways, consumer electronics and PC peripheral devices.

Recently, UPnP has been gaining acceptance from consumer electronic and PC peripheral makers, and is becoming the common way for these devices to interact with more powerful computing devices such as PCs, servers or handhelds. There are also specific industry efforts such as Digital Living Network Alliance (DLNA) that use UPnP as a base and ensure that specific device interaction scenarios are reliable enough for the home market.

UPnP [20] has the following characteristics:

- ***Media and device independence*** - UPnP technology can run on any medium including phone line, power line, Ethernet, RF, and 1394.
- ***Platform independence*** - Vendors can use any operating system and any programming language to build UPnP products.
- ***Internet-based technologies*** - UPnP technology is built upon IP, TCP, UDP, HTTP, and XML.
- ***UI Control*** - UPnP architecture enables vendor control over user interface and interaction using the browser.
- ***Programmatic control*** - UPnP architecture also enables conventional application programmatic control.

- ***Common base protocols*** - Vendors agree on base protocol sets on a per-device basis.
- ***Extendable*** - Each UPnP product can have value-added services layered on top of the basic device architecture by the individual manufacturers.

The UPnP architecture is composed of three elements: Control Points, Devices and Services [55]. These elements are shown in Figure 2.10.

1. ***Device*** - is a container of services and characterized by a XML device description. The description includes the device properties and a list of pointer URLs to the services it contains.
2. ***Service*** - is the smallest unit of control in UPnP. It is modeled by state variables, and allows executing actions. It consists of a state table that contains the state variables of the service, a control server that receives and executes action requests, and an event server that alerts subscribers when the state of the service changes.
3. ***Control point*** - is a controller capable of discovering and managing other devices. A control point can discover a device, retrieve its description, invoke actions to control a service within the device, and subscribe itself to the service's event source. To enable true peer-to-peer connectivity, devices should incorporate control point functionality.

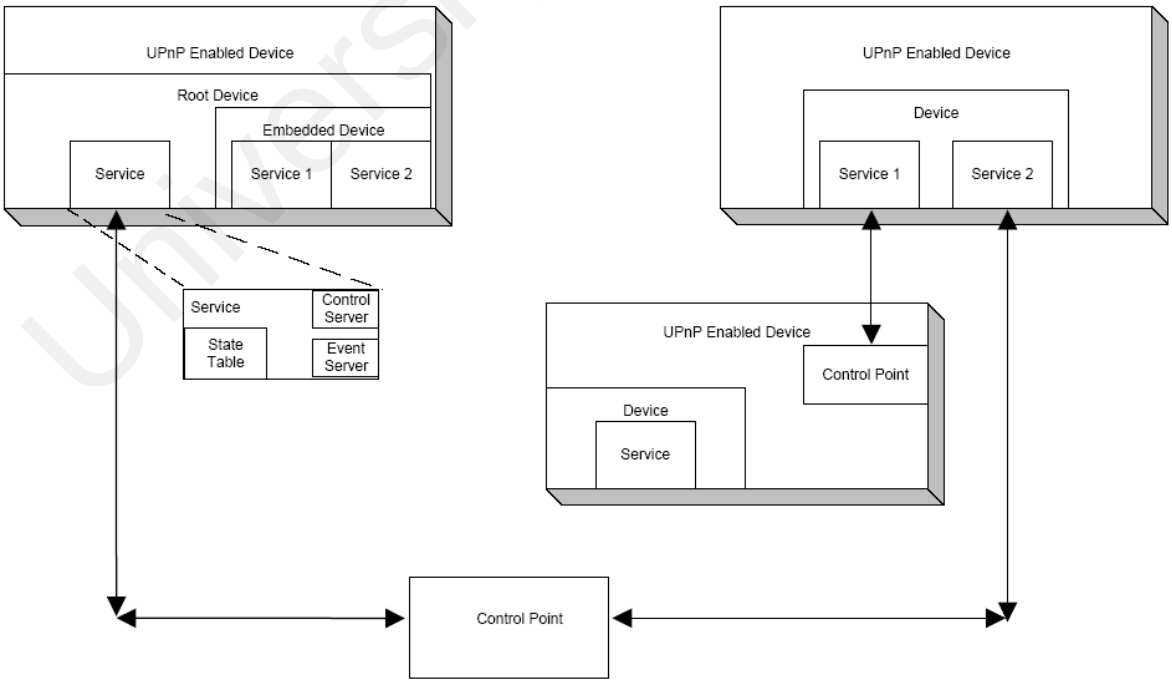

Figure 2.10 UPnP Control Points, Devices and Services [20]

UPnP Devices handle the discovery and control requests from Control Points and produce events to inform Control Points. A device can host several services and other embedded devices. For example, a printer/scanner built-in device would host a print and scan embedded device, which in turn would consist of a print service, a scan service, a color type service, and a picture resolution service. Services provided by a device are standardized by different working groups. This information such as device manufacturer, device type and version number as well as services provided by the device is stored in an XML device description file.

UPnP Services are the smallest units in the UPnP network, which show actions and model state through state variables. These actions and state variables are stored in an XML service description file. The state of a service is modeled in the state table through state variables. Action requests of the Control Points are handled by the Control Server, and status changes are forwarded to interested Control Points through the Event Server.

A Control Point is used to discover and control the devices providing services on the network. After discovering a device on the network, a Control Point can:

- Retrieve the device description and get a list of associated services
- Retrieve service descriptions for interested services
- Invoke actions to control the service
- Subscribe to the service's event source. Anytime the state of the service changes, the event server will send an event to the control point.

2.6.2.1 UPnP Protocol Stack

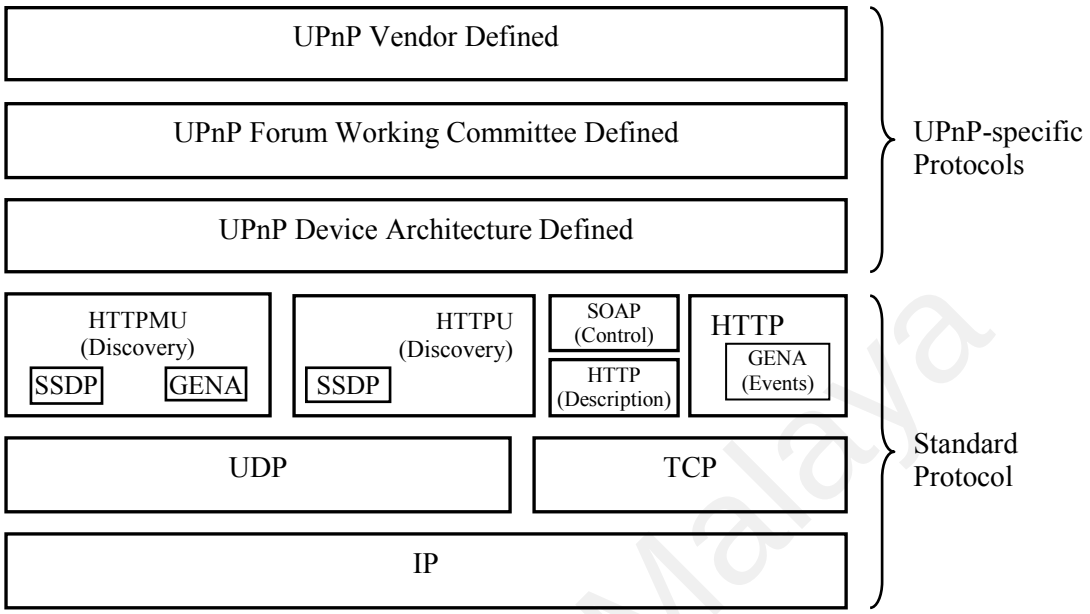

Figure 2.11 UPnP Protocol Stack [21]

The figure above shows the UPnP protocol stack. Above the protocol stack is the vendor-specific APIs where vendors can choose their own programming model on top of UPnP. Below the protocol stack is the vendor-specific OS and hardware platform where vendors can choose their own OS and hardware to implement UPnP. The UPnP protocol stack is divided into two parts: UPnP-specific protocols and standard protocols.

2.6.2.1.1 UPnP Specific Protocols

UPnP Device Architecture Defined

It defines basic device architecture and is contributed by Microsoft.

UPnP Forum Working Committees Defined

The UPnP Forum is a group of companies and individuals from various industries. It defines standards for describing device protocols and XML-based device schemas. Its goals are to enable devices to connect seamlessly in the home and office network. The UPnP Forum Working Committees are set up to define device-specific and domain-specific format of data based on UPnP Device Architecture, such as service types and device types.

UPnP Vendor

UPnP Vendors specify their own extensions based on what working committees define. The vendor-specific information contains data specific to some kind of devices. Examples of data specific are the device description URL and device identifier URL and argument values.

2.6.2.1.2 UPnP Standard Protocols

TCP/IP & UDP/IP

TCP/IP and UDP/IP networking protocol stack serves as the base on which the rest of the UPnP protocols are built. UPnP leverages TCP/IP and UDP/IP protocol to enable network connectivity over different physical media and interoperability over multiple UPnP devices. UDP is used for discovery. TCP is used for description, control, eventing and presentation.

HTTP, HTTPMU, HTTPU

Hypertext Transfer Protocol (HTTP) is a core protocol of UPnP. HTTP Multicast over UDP (HTTPMU) and HTTP Unicast over UDP (HTTPU) are the variant of HTTP. All aspects of UPnP build on top of HTTP or its variants. The two variants are defined to deliver messages on top of UDP/IP instead of TCP/IP. These protocols are used by SSDP. The basic message formats used by these protocols are required for multicast and unicast communication.

GENA

Generic Event Notification Architecture (GENA) was defined to send and receive notifications using HTTP over TCP and using HTTPMU over UDP. GENA also defines the concepts of subscribers of notifications to enable events. GENA formats are used by the devices to create presence advertisement by sending using SSDP, and also for the devices to signal their changes of service state variables.

SSDP

Simple Service Discovery Protocol (SSDP) was built on top of HTTPU and HTTPMU. SSDP defines methods on how network services can be discovered on the network in two mechanisms; device advertises its presence by using *ssdp:alive* and Control point discovers interest resources by using *ssdp:discovery*. SSDP also provides a way for devices and services to leave the network gracefully; device leaves the network by using *ssdp:byebye*.

Control point send SSDP search request to discover devices and services on the network. UPnP devices listen to the multicast port. Upon receiving a search request and a match is found, a unicast SSDP response is sent to the control point.

SOAP

Simple Object Access Protocol (SOAP) uses XML and HTTP to execute remote procedure calls (RPCs). SOAP can also make use of Secure Sockets Layer (SSL) for security and use HTTP's connection management facilities. UPnP uses SOAP to invoke actions on devices and return results or errors to the control points. Each SOAP request message contains the action to be invoked and associated parameters. The SOAP response message contains the status, return value and return parameters.

XML

Extensible Markup Language (XML) is the universal format for structured data on the Web. UPnP selected XML to describe device and service descriptions, control messages, and event messages because of its characteristic. XML place nearly any kind of structured data into a text file. Tags and attributes are used in HTML to define the meanings globally. However tags and attributes used in XML are within the context of their usage and are able to develop schemas for document types.

2.6.2.2 UPnP Networking

There are six steps in the UPnP networking; *addressing, discovery, description, control, eventing,* and *presentation*. The first three networking steps, addressing, discovery, and description, should be done in order; while the last three are independent of each other.

Addressing

Addressing is the foundation for UPnP devices to be used to identify devices uniquely. IP addressing is needed by UPnP control point and devices to obtain addresses to communicate with each other. UPnP devices apply two addressing mechanisms, Dynamic Host Configuration Protocol (DHCP) and Auto IP. Every device required to have a built-in DHCP client. This device will require an IP address from DHCP server when it connects to the network. If the DHCP server is unavailable, the device must use Auto IP to get an address.

Discovery

When the device obtains address appropriately, discovery can take place. Devices advertise their services and discovery enables control points to search for interesting devices. There are two scenarios in discovery networking phase [53].

- When a new control point is plugged into the network, it multicasts a SSDP discovery message in order to look for devices of interest. All devices must listen to the standard multicast address (239.255.255.250:1900) and must respond directly to the requester if they match the search criteria.
- When a new UPnP device is plugged into the network, it multicasts SSDP discovery messages to a standard address and port (239.255.255.250:1900) to advertise its root device, embedded devices and services. Control points must listen to the port 1900 so that it can notice once new devices or services advertise their capabilities.

In the cases above, discovery processes are handled by the SSDP, and the exchange element is the discovery message which contains essential information about the device or service.

Description

Description is carrying out after discovery. Control point can learn more about device capabilities by retrieving the device description from the URL provided by the discovery message and from the device description. The UPnP description for a device is expressed in XML.

Control

When a control point gets the device description, it may control the device via the URL contained in the device description. Control point controls by sending control message to the control URL of the service provided in the device description and the service responds with results or errors. Control point also queries the value of state variables of the service by sending query message to the control URL. The service responds with the state value.

The state variables querying and actions invoking are the kind of remote procedure call (RPC). This RPC mechanism used is the Simple Object Access Protocol (SOAP) which uses XML as data representation and HTTP as the underlying transfer protocol to send messages.

Eventing

When a control point gets the device description, it may subscribe for update information by requesting to the eventing URL contained in the device description. Subscription initiated from a control point to a service by sending a subscription message or a renewal message or a cancellation message. Event initiated from a service to a control point for publishing changes to a service state. A service has eventing if and only if at lease one of its state variables is evented.

In the process of eventing, if the subscription is accepted, the service responds with an ID as well as duration for this subscription. This Subscription ID is unique and its used for renewing and canceling process. When the subscription is accepted, the service sends the initial event message to allow the control point to initialize the model of service state. Then, control point receives all events with latest status sent by the service. When the subscription duration expires, the service will stop sending event messages. A renewal message is sent to keep the subscription alive and a cancellation message to cancel the subscription. These messages (subscription, renewal, cancellation and event message are delivered via HTTP that has been extended using GENA methods. The HTTP messages are delivered via TCP.

Presentation

Control point may control the device and view device status via the presentation page from presentation URL. This presentation page is a HTML based user interface and contain device operational parameters, device statuses and actions on the services.

2.6.2.3 Standard Service in UPnP

UPnP Service Template [48] and UPnP Device Template [49] are defined by the UPnP Forum to standardize service types. These templates defined that each UPnP service is described by a service description which is written in XML by UPnP vendor. A standard UPnP service is defined by UPnP Forum working committees and specify by a UPnP vendor and A non-standard service is completely specified by a UPnP vendor.

The UPnP device is a container for UPnP services. Each service in an UPnP device may contain any number of actions. Each action also has a name, a value and a direction.

2.6.2.3.1 UPnP Service ID

UPnP device is uniquely identified by Universal Unique Identifier (UUID). The UPnP service ID has the following format:

- **urn:upnp-org:serviceId:***serviceID*
  It is used in the standard UPnP service.
- **urn:***domain-name***:serviceId:***serviceID*
  It is used in the non-standard UPnP service.

2.6.2.3.2 UPnP Service Type

The service type is defined based on the device type. From the UPnP Forum, the printer device could have *PrintBasic:1* and *PrintEnhancedLayer:1* service type. UPnP service has a service type Uniform Resource Identifier (URI) that uniquely identifies the service. The following are the kind of UPnP service type:

- **urn:schemas-upnp-org:service:***serviceType:v*
  It is used in the standard UPnP service.
- **urn:***domain-name***:service:***serviceType:v*
  It is used in the non-standard UPnP service.

2.6.2.3.3 UPnP Service Description

UPnP Service description is used to describe the service type. Service description allows devices to list the functionality they provide. The device description document contains device information such as manufacturer, make, model, serial number, a lost of services provided by the device, and a list of embedded devices.

One service description exactly represents one UPnP device's service. A device can have one or more services, therefore one or more service descriptions are required. Each UPnP service is made up of zero or more actions and one or more state variables. Below is the sample of the Service Description [48].

(Values in *italics* are placeholders for *actual elements and values*.)

```
<?xml version="1.0"?>
<scpd xmlns="urn:schemas-upnp-org:service-1-0">
    <specVersion>
        <major>1</major>
        <minor>0</minor>
    </specVersion>
    <actionList>
        <action>Description for action defined by the UPnP Forum or the Vendor </action>
        Other UPnP-Forum-defined actions go here, if any.
        Other vendor-defined actions go here, if any.
    </actionList>
    <serviceStateTable>
        <stateVariable sendEvents="yes">
            Desciption for state variables defined by the UPnP Forum or the Vendor
        </stateVariable>
        Other UPnP-Forum-defined state variables go here, if any.
        Other vendor-defined state variables go here, if any.
    </serviceStateTable>
</scpd>
```

Figure 2.12 UPnP Service Descriptions [48]

2.7 Service Discovery Technologies

Users usually are interested in the services provided by appliances or devices. They are not too concerned on the underlying technologies used in communication. Therefore, network devices should be capable of advertising their services to be discovered by user devices automatically. Service discovery protocols work by advertising the available service and providing information about the capabilities and details of the services. Then the users may register to use the service. The following section discussed the type of service discovery technologies.

2.7.1 Bluetooth Service Discovery Protocol (SDP)

Bluetooth [5] developed in Ericsson Mobile Communication Laboratory in 1994 and version 1.0 published by Special Interest Group (SIG) in 1999. The Bluetooth Special Interest Group (SIG) includes companies like 3Com, Ericsson, IBM, Intel, Lucent, Microsoft, Motorola, Toshiba, Nokia and more than 3000 associate member companies develop the specifications.

Bluetooth originally was used to communicate low-power wireless handheld device such as mobile phones, laptops and personal digital assistants (PDA) and can be use to connect wired devices like desktop computers and printers. Using this technology, several mobile and wireless devices in addition to general-purpose computers can be connected together in a small network in order to enable data transfer.

Bluetooth has the followings features:

- ***Low-cost*** - There is no fee for channel rent and cable consumption in Bluetooth communication because Bluetooth uses 2.4GHz-ISM-Band (Industrial, Scientific, and Medical) for communication. The ISM radio bands are reserved globally for the non-commercial and unlicensed use of Radio Frequency (RF).
- ***Low-power consumption*** - Bluetooth uses a power control scheme to reduce the power consumption of devices. The mechanism of the power control scheme is to adjust the RF output power to the minimum level which is mandatory to maintain the communication link.
- ***Short-range*** - The Bluetooth communicate two device in the range of 10 meters without caring about obstacle in between.
- ***No cable connection*** - Bluetooth is a RF-based wireless communication technology without the need of cable.
- ***Little configuration*** - There is not much need of configuration to be set for two Bluetooth devices to communicate except for some security authentication before the connection is established.
- ***Piconet*** - Two or up to eight active Bluetooth units form a piconet.
- ***Scatternet*** - Two or more piconets with overlapping coverage areas form a scatternet.
- ***Security Connection*** - Bluetooth defines three levels of security:
    a) Secure Mode 1 – Non-secure. No security procedure is applied.
    b) Secure Mode 2 – Service-level Security. Security scheme is initiated after establishing a connection.
    c) Secure Mode 3 – Link-level Security. Security scheme is initiated before establishing a connection.

Bluetooth Service Discovery Protocol

Figure 2.13 shows the Bluetooth protocol stack. The two physical layers of this stack are at the button and can be accessed through the Host Controller Interface (HCI) which provides a uniform interface method. The Link Manager Protocol (LMP) is the link layer and it provides authentication and encryption functions. The Logical Link Control and Adaptation layer Protocol (L2CAP) provide connection–oriented and connectionless data services with the upper layers. Other protocols such as TCP/IP, OBEX, SDP, RFCOMM (which emulate the RS-232 interface) can be set on the L2CAP layer, which is a proprietary protocol. Applications can be built on top of these protocols.

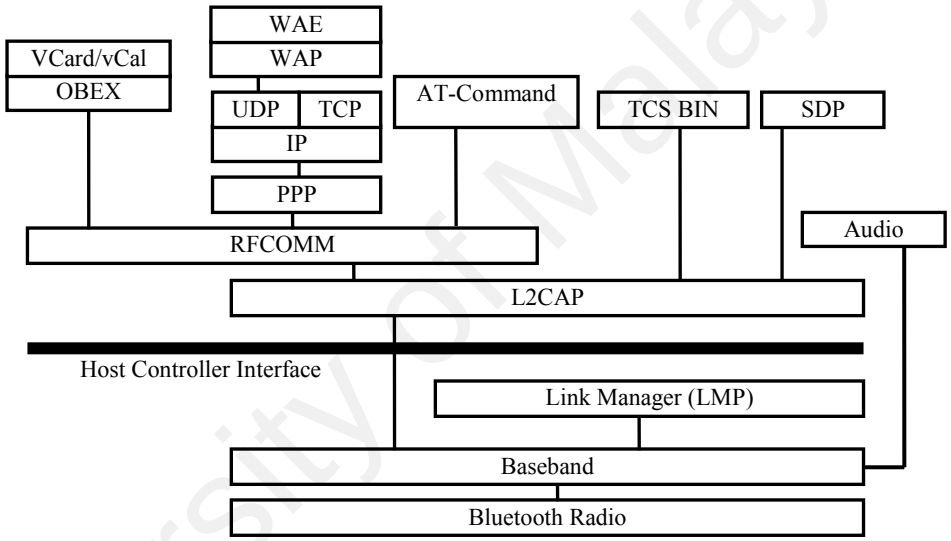

Figure 2.13 Bluetooth Protocol Stack

The Bluetooth protocol stack contains a Service Discovery Protocol (SDP) [6] that is designed to categorize and to advertise the services provided by each node. It enables the retrieval of information that can be used to configure the task to support user's application. Discovering could be done via searching for service by service attributes, service type and browsing without any service characteristics. SDP does not cover other mechanism such as selection and accessing of services.

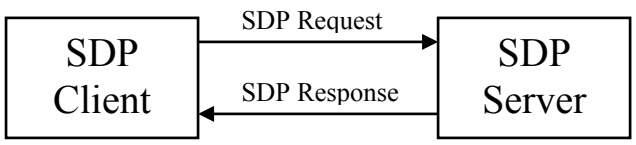

Figure 2.14 DSP Client-Server Interactions

SDP enables a client application to discover services and service attributes by issuing an *SDP request* to the server application refer to Figure 2.14. An SDP server maintains a list of service attributes and records that describe the characteristics of services. There is a maximum of one SDP server per Bluetooth device and there is no SDP server for Bluetooth device that acts only as a client. Each service attribute describes a single characteristic of a service. A Service Attribute consists of an Attribute ID which differentiates each service and Attribute value where its length is determine by the Attribute ID. The server issues an *SDP response* to the client. A client must open a separate connection to the service provider in order to use the service.

2.7.2 JINI

The Jini networking system [16] is a distributed infrastructure built around the Java programming language. The basic communication model is based on the semantic model of the Java Remote Method Invocation (RMI) system. Objects in one Java Virtual Machine (JVM) communicate with objects in another JVM by receiving a proxy object, which implements the same interface as the remote object. This proxy object deals with all communication details between the two processes, and it may introduce new code into the process to which it is moved. This is possible because Java bytecode is portable. It is also safe because of the built-in verification and security of the Java environment.

The Jini system adds some basic infrastructure and parts of a programming model to the underlying communication model. The infrastructure provides a mechanism by which clients and services can join into the Jini network, while the programming offers a mechanism to build reliable and distributed systems.

Jini creates a collection of networked devices, which represents a group of services. Jini service can be realized to represent hardware devices, software programs or combination of the two. Jini consists of the following types of entities:

1. ***Service Provider*** - It provides service to requester. This can be hardware device such as a printer, a disk space or projector. Services can be used by a user or another service.

2. ***Lookup Service*** **-** It displays available services and provides a mapping interface indicate the functionality provided by services and their devices. It can also contain directory services other than Jini by bridging mechanism.
3. ***Client*** **-** It looks up or request for services.

The interaction between various entities can be shown in Figure 2.15. The main purpose of discovery process is to locate lookup services.

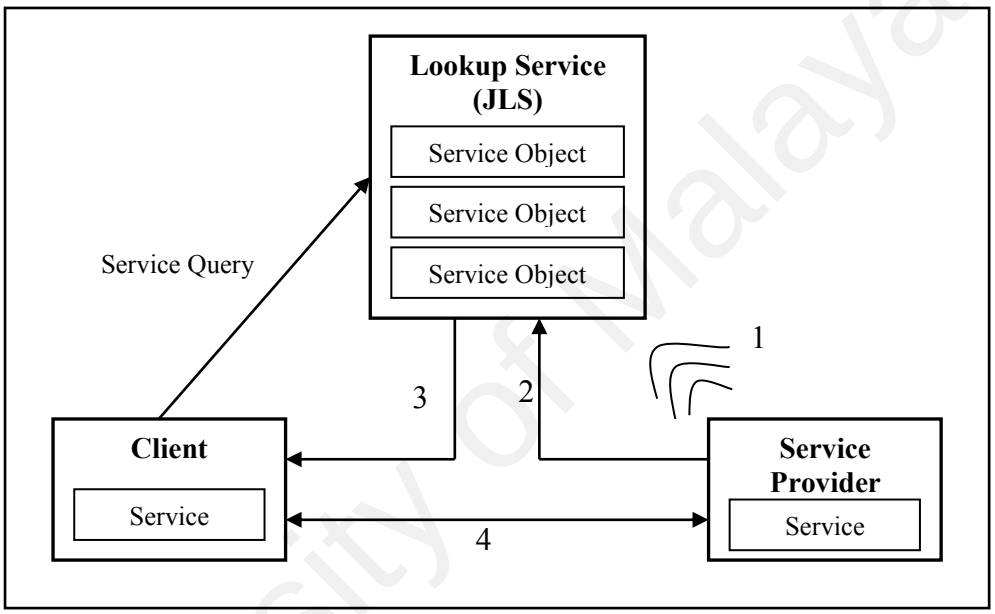

Figure 2.15 Jini architecture [17]

The architecture of Jini is based on the Jini Lookup Service component (JLS). The services must locate a JLS server using the discovery protocol. Refer to step 1 in Figure 2.15. Then, service provider registers their services in the JLS in step 2. The clients also discover a JLS and query it about the available services. The matching between queries and services can be made by comparing the list of characteristic attributes. Each service will be maintained by the JLS only for a certain period of time. Those services that are not registered are eliminated from the register.

When a client has discovered the desire service, it can communicate directly with the service provider by using the necessary code that the server previously uploaded during its registration. This is one of the main advantages of the Jini architecture. The clients do not need to own the specific drivers for a service; they can download it directly from the JLS in

step 3 and then invoke the service as in step 4. Besides that, the mobility of this code increases the traffic and latency of the transmissions.

Jini's client and server require to run on a JVM. Jini is platform independent but it is java-dependent. Therefore, everything involving Jini must be programmed in Java.

2.7.3 Salutation

The Salutation architecture [18] is an industry consortium's solution to the service discovery and utilization problem. The Salutation Consortium is developing it. The architecture provides a standard method for application, services and devices to advertise their capabilities or request the desired ones. It is claimed to be processor, operating system and communication protocol independent.

The Salutation architecture is presented in Figure 2.16. Its key piece is the Salutation Manager (SLM). Every Salutation network has a SLM, or uses a remote SLM by means of the Remote Procedure Call (RPC) protocol. The SLM provides services and clients with a transport independent interface (SLM-API). The different SLMs communicate among themselves using the Salutation Manager Protocol, which uses the Remote Procedure Call.

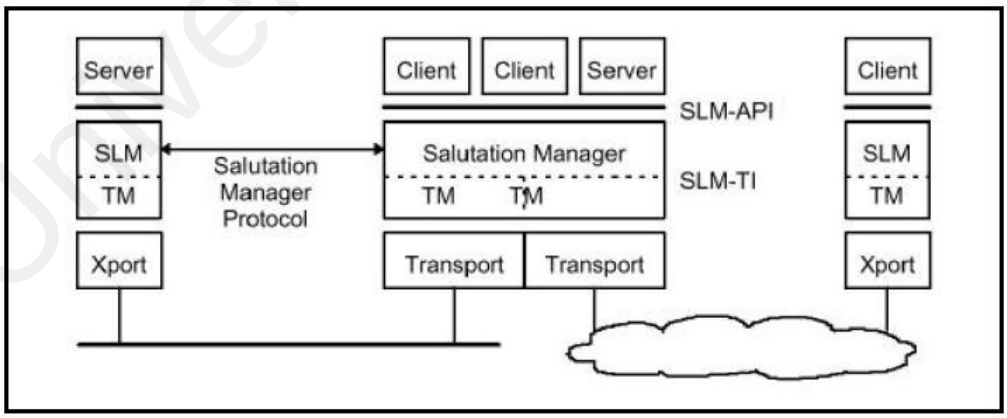

Figure 2.16 Salutation Architecture [18]

The SLM also presents a transport-independent interface (SLM-TI) to transport dependent entities, called Transport Managers (TM). Each Transport Manager supports one kind of transport, making thus the SLM independent of which one is used. The Salutation Manager

and the Transport Manager together perform the service of a service broker. The main tasks carried out by a service broker are the following:

- ***Service Registry*** - A SLM holds a registry of services. It hold the information about the service connected to it and also can store information about other services on the network. A SLM can act as a central directory for all Salutation equipment in a network.
- ***Service Discovery*** - The SLM can discover other SLM and their services. This discovery is based on the comparing of the service type and matching of specific characteristics.
- ***Availability Check*** - The SLM can periodically check the availability of the registered services.
- ***Service Session Management*** - SLM creates a service session between the client and service for the client to use the service.

There are several commercial implementations of the Salutation protocol exist. The Salutation architecture is rather mature and it can be used together with Bluetooth's service discovery protocol or SLP. When Salutation is used with SLP [19], Salutation obtains the scalability of directory-based service discovery.

2.7.4 Service Location Protocol (SLP)

The Service Location Protocol [15] designed by the Service Location Group (SRVLOC) of the Internet Engineering Task Force (IETF) for IP-based networks. SLP provides a scalable framework for the discovery and selection of network service or devices. It uses service URLs, which define the service type and address for a particular service. Based on a URL, users or applications can browse, select and use the available services in their domain. The SLP infrastructure consists of three types of agents: User Agent, Service Agent and Discovery Agent.

1. ***User Agent (UA)*** - UA is a client of services. UA sends a request about the desire service to DA or directly to SA to requests for services with particular characteristic.
2. ***Service Agent (SA)*** - SAs are services or resources that can be used by UA. These services could be printers, projectors, camera, or scanner. SA advertises their

presence by multicasting, broadcasting or unicasting and provides services. SAs advertise and register their presence with DAs. SAs also intercept and reply to request about the services they offered with an access point.

3. ***Discovery Agent (DA)*** - DAs are central databases for services. DAs intercept advertisements from SAs, accept registration from SAs and reply UAs on behalf of SAs.

Mode of Discovery [15]:-

- ***Static Discovery***

  SLP agent obtains an address to Discovery Agent via DHCP while connecting into the network.

- ***Active Discovery***

  1. ***SA and UA do not know the location of the DA***

     SA and UA use multicast to send Service Request message with a service type set to *service:Directory-Agent*. Every DA on the network replies to this request with a unicast message called Directory Advertisement message (*DAAdvert*) that contains the DA's URL. This is depicted in Figure 2.17.

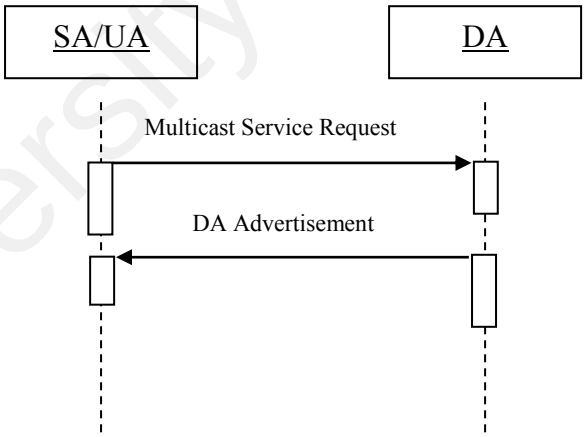

Figure 2.17 Active Discovery of DA using when SA and UA do not know location of DA.

  2. ***SA and UA know DA location***

     SA and UA send a unicast service request with a service type *service:Directory-Agent* to the DA which replies with a *DAAdvert* message. This is depicted in Figure 2.18.

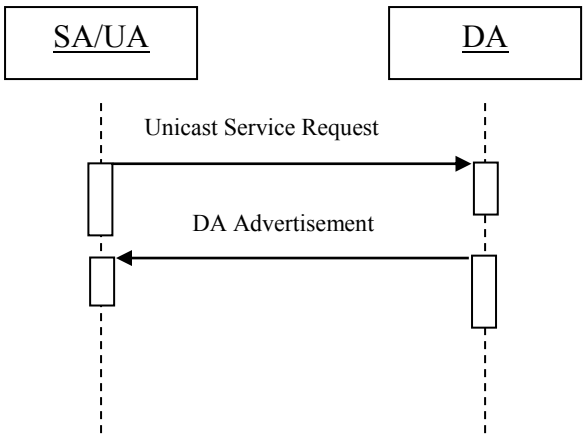

Figure 2.18 Active Discovery of DA when SA and UA know the location of DA

- ***Passive Discovery***

  DAs send periodically multicast *DAAdvert* message to notify its presence to UA and SA which listens to the advertisement message on port 427. Interested UA or SA extract the URL from the advertisement message and use it to contact DA.

After discovery, SA can register the service they offer by sending unicast Service Registration message to DA using UDP unicast. The DA responds by returning a Service Acknowledgement message. UA can request a service by sending a unicast Service Request message which contains the type of service, the predicate if the attributes and the scope to DA. If the Service Request matches one of the registered services, the DA replies with a Service Reply message, which contains the URL of the requested service.

UA requests for a service by sending a unicast Service Request message to DA. This message contains a set type of service *service:directory-agent*. The DA respond with DAAdvert message only if the predicate can be satisfied with the DA's attributes. The summary of the communication is depicted in Figure 2.19. The SA, UA and DA are member of the same scope.

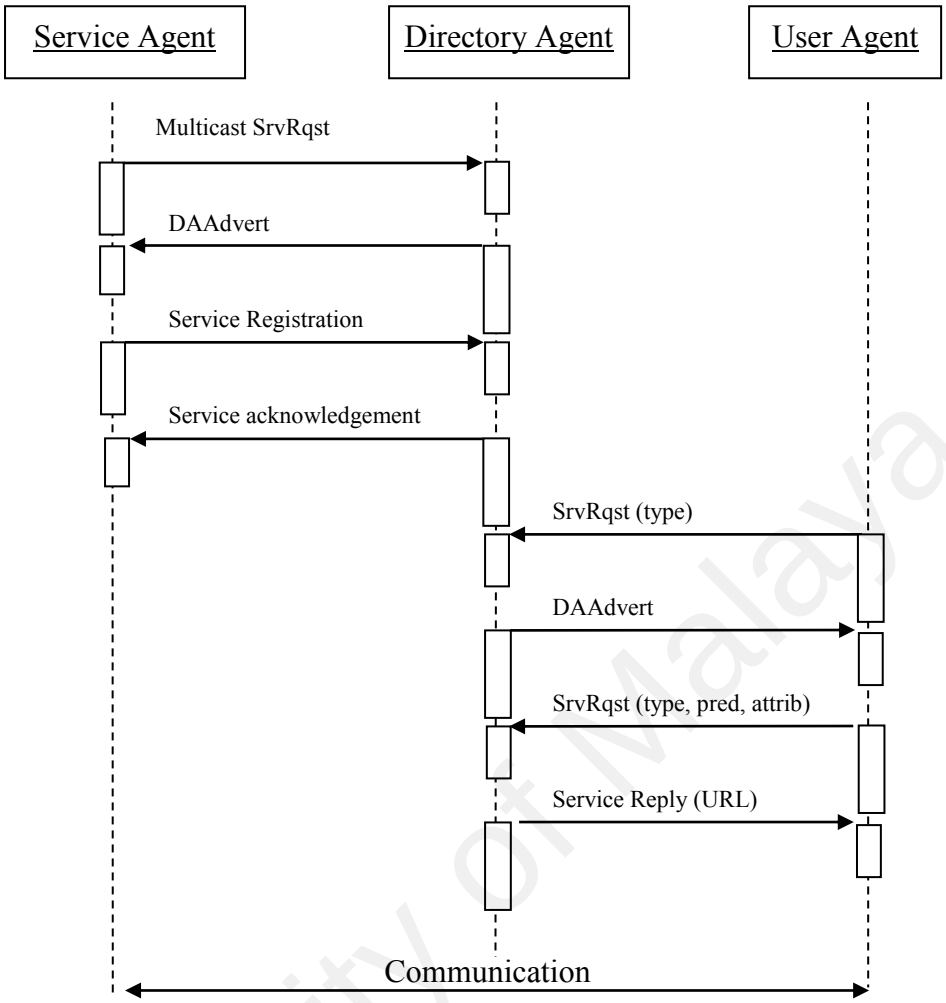

Figure 2.19 Complete SLP Operations

SLP messages use authentication mechanism through digital signature where it can be generated by using cryptographic technique.

SLP defines eleven messages that UAs, SAs and DAs interchange with each other. Some of them are required for every implementation and some are optional. All the messages in SLPv2 have the same SLP header followed by a specific body.

2.7.5 Comparison of Service Discovery Protocol

Bluetooth SDP, JINI, Salutation, SLP and UPnP presented above have different approaches for service discovery. They have different philosophy and discover resource or service in

different ways. Therefore, it is need to make a comparison between them on some features and issues that is related to the context of this thesis [22].

The Bluetooth Service Discovery Protocol is simple because it is designed to work in small devices and in ad hoc environments. This makes the protocol rather limited. It lacks the service agent's functionality such as service registration or aggregation and the lease mechanism, although clients may poll for the availability and estimated life time of services. Bluetooth SDP do not provides access to services, but only information about them. SDP is only restricted to the Bluetooth environment and not other platform. Thus, it is not expected to have full functionality compare with other service discovery mechanisms. It is rather collaborate with them.

JINI is a powerful architecture and able to interoperate with SLP using bridges. The main achievement of JINI is the code mobility possibility, but that also produces more traffic and the more latency of the transactions. It has a disadvantage that it needs the JINI Lookup Server to be used even in ad hoc networks. JINI is platform independent but everything must be programmed in Java and every device needs a JVM running on it. This is a drawback because it can be too expensive for some small devices.

The Salutation architecture is rather mature and is supported by an important consortium. Its major deficiency is defective scalability. This however, can be solved when interoperate with SLP. The possibility of operating with SLP, and Bluetooth, and its platform and transport protocol independency can make Salutation become the bridge that allows interoperability between different protocols.

UPnP is supported by many companies and great industries. UPnP are compatible standard for UPnP enabled Internet gateways. The drawback of UPnP are the possibility of having a central repository of services that makes scalability worse and the absence of better searching mechanism. Besides that, it allows some extra features, such as events notification, remote services control and devices auto-configuration.

The SLP is the standard of the IETF. SLP accepted by many developers but on the other hand not supported by any important company. The SLP architecture is very flexible because it can work with or without a central register of services and many of its features are

optional. The drawbacks of SLP are its dependency on TCP/IP and the lack of some added functionalities. SLP does not directly provide access to services or any of the UPnP extras such as events notification or remote devices control. These capabilities have to be provided by other protocols or applications.

Table 2.1 shows a comparative study of the various technologies by some features.

Feature	**SDP**	**JINI**	**Salutation**	**SLP**	**UPnP**
Developer	Bluetooth	Sun Microsystems	Salutation Consortium	IETF	Microsoft
License	Open source	Open license, but fee for commercial use	Open source	Open source	Open (only for members)
Programming language	Independent	Java	Independent	Independent	Independent
OS and Platform	Dependent	Independent	Dependent	Dependent	Dependent
Code Mobility	No	Yes	No	No	No
Security		Java Based	Authentication	IP dependent	IP dependent
Used in small devices	Yes	No	Yes	Yes	Yes
IP Based	No	Yes	Yes	Yes	Yes
To work without central database	Yes	No	No	Yes	Yes
Support Extended Attributes	Yes	Yes	No	Yes	Yes
Support Advertisement Message	No	No	No	Yes	Yes

Table 2.1 Service Discovery Protocol Comparison

2.7.6 Summary

From the comparison above, there are advantages and also drawbacks for each respective service discovery. Therefore, not single protocol can be considered as the best service discovery protocol for devices communication in home and office environment. Instead, the used of service discovery protocol depend on the need of the environment, the middleware software and related functionalities. These protocols will probably have to learn to collaborate with each other in the near future.

CHAPTER 3 SYSTEM ANALYSIS

3.1 Introduction

Chapter 3 describes the proposed scenario of the Service Discovery where services are discovered over heterogeneous networks. Before the implementation of this scenario, the requirements and specification of the design architecture are defined. This chapter discusses on the functional requirement, non-functional requirement needed for the development of the scenario.

3.2 Proposed Scenario Description

This scenario depicts that a salesman bringing his pocket pc into his client office. He needs to present his company product to the client. However, the product details are contained in the pocket pc. He uses his pocket pc to discover printer service in the office and send the file for printing. One of the purposes here is to facilitate communication of devices over heterogeneous network where the user able to discover and interoperate with devices with different underlying technology. Therefore, a device gateway is needed to sit in between these heterogeneous networks to do the translation. A resident gateway was proposed in this design.

For the salesman to use his pocket pc to discover the printer service, there are three entities involved performing the action. They are the user accessing the office network over the Internet (pocket pc), the gateway (resident gateway), and services within the office network (printer service). The proposed design for the above scenario is in Figure 3.1.

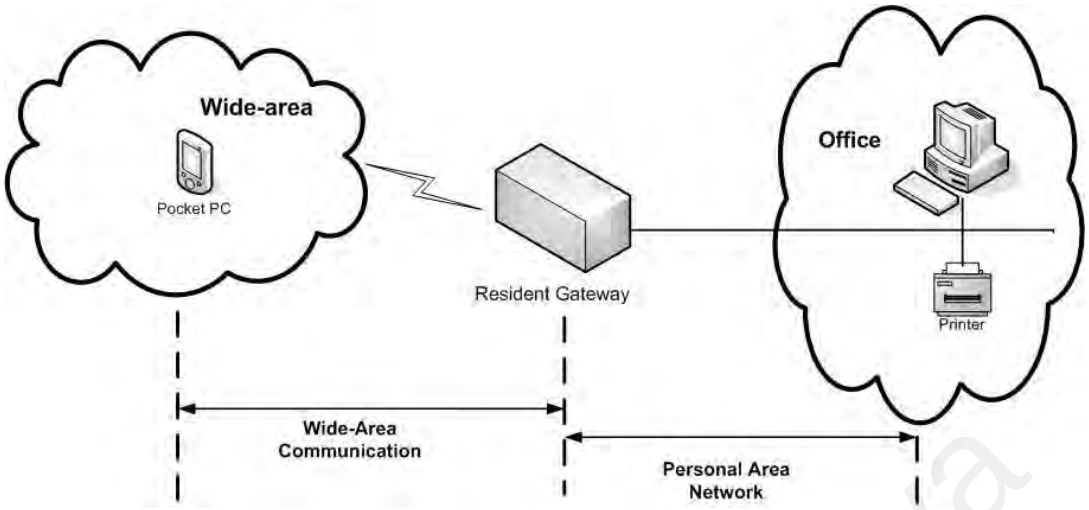

Figure 3.1 Proposed Scenario diagram

In this scenario, the gateway will discover the existence of the pocket pc. The pocket pc sends a service request to the resident gateway. The resident gateway checks the availability of the printer service and sends a result back to the pocket pc. If the gateway discovers the printer service, the salesman is able to send the file over to the printer to print. Then, the clients are able to get the product details in hardcopy format.

3.3 Functional Requirements

This thesis consists of three major categories of functional requirement. There are the service discovery and network communication, service discovery and interaction over Wide-Area, and service discovery within Personal Area Network.

3.3.1 Service discovery and network communication

There is a need of few different technologies to interoperate among themselves to allow communication and services to be discovered over heterogeneous network. First of all, there are a number of different discovery service protocols in use. These service discovery protocols are like Service Location Protocol (SLP), Jini, Bluetooth SDP, UPnP, Salutation, HAVi and etc. There is also little hardware, devices and transport medium operate among each other for the message to transfer from one domain to the other. These include

technology such as wired cable, wireless connection, Bluetooth, infrared, HomeRF, GPRS, X10 with the powerline and etc. These varieties of technologies with different capabilities and characteristics offer and enable devices to interact among each other.

3.3.2 Service discovery and interaction over Wide-Area

The second functional requirement is to enable remote access from Internet to the internal devices. The same discovery mechanisms for local devices to interact should apply here, where user outside the domain is able to browse and search services in the local domain. Therefore, a potential wide-area network solution should be used. The requirements for devices to interact over wide-area network are listed below [35].

3.3.2.1 Naming and Addressing Requirements

The naming and addressing scheme is one of the requirements that must be able to support both location and device independence [54].

- A networked device must be assigned a generic globally unique name so that any communicating entity can refer to it.
- There must be support for classification of addresses and selection between multiple instances.
- It must be possible to search for particular capabilities and to identify which devices possess those capabilities.

3.3.2.2 Wide-area Accessibility Requirements

- Network devices in the PAN environment (eg. home or office) must be accessible from outside.
- Only a subset of the devices within a domain may need to be addressable from outside. It should be able to query the domain and to discover the externally accessible devices.

3.3.2.3 Communication Protocol Requirements

- The communication protocol must provide a flexible payload that will allow the transport of commands to, and responses from, individual network devices.

- The communication protocol must support efficient control of messaging. It is expected that control messages for devices will be short and may or may not form part of an ongoing dialogue.
- The communication protocol must be able to encapsulate various network devices characteristics. For example, some device may act and respond immediately, while others may only respond after a non-determine amount of time.
- The communication protocol must able to support event notification of the status of the device.
- Support for the following communication modes is required:
    - Control - Turn on or off the light.
    - Queries - Request the temperature of the air conditioner.
    - Asynchronous events - Notify when the security alarm goes off.
    - Discovery - Find the device that can meet specific requirement.
    - Media streaming - view the babysitter camera.

3.3.2.4 Protocol Transparency and Independence

It is important that the wide area communication is independent of any particular or specific protocol implementation. It must be also able to work with different in-domain networking technologies transparently. This requirement applies to both physical networking and application networking technologies [35].

3.3.3 Service discovery within Personal Area Network

User interfaces should be made available to be retrieved by devices so that different devices are able to interact among themselves. The efficient and sensible approach is to use a generic user interface format. The generic user interface format can then be either transcoded on the server or in the client. The requirement for service discovery and communication between different model devices for the generic user interface format are as follow [36].

- The generic user interface representation should be independent of the technology transport protocols used over the network.
- There should be a natural separation of user-interface and non-user-interface code.

- The interface representation should be extensible. Adding new features will not affect the existing features.
- The control should be able to control the device remotely via provided connections. There should be no dependence on the actual connection technology. This means that there should be a way to communicate (user) events to the controlled device.
- The interface representation should independent from target format.
- The generic user interface should be able to be used to interact with the networked device internal and over the Internet.

Additional to that, another important requirement is the coordination and synchronization of different model devices. Below are the requirements for synchronizing multi-modal user interface [37].

- Services should be able to interact in parallel with the same networked services via a multiplicity of user and networked devices.
- Networked devices should present across user devices in a unified, synchronized and coordinated view.
- Devices should be able to interact in uniform and behaviors independent with other model of devices.
- There should be coordination of the user interfaces, behaviors and services.

3.4 Non-Functional Requirements

Because of the dynamic nature of home and office network with different network devices and appliances, the networked devices architecture is necessary to be scalable and robust and secure in real-time communication.

3.4.1 Security

Security is an important concern in these different technology networks. All communications require authentication, authorization, privacy, and replay protection. All the contents of the messages must be kept private to avoid eavesdroppers.

Security threats are divided into three main classes; availability, confidentiality, and integrity [38]. Availability is an assurance that the systems may carry out their intended functions when needed. Confidentiality is an assurance that services information is shared only among authorized person and Integrity is an assurance that information is authentic, complete and accurate for its purpose.

3.4.2 Performance

It is a requirement that all the messages sent within network devices are sent efficiently and without any delay. This protocol should work equally well in connection-oriented mode (TCP) as well in connectionless mode (UDP).

3.4.3 Scalability

The PAN network can be expended with the increase of new network devices. The gateway need to do most of the intensive processing to enable many devices with low memory and processing power able to communication among each other. As the number of clients and services in an environment increases, so the burden due to dynamic service discovery and interaction increases.

3.5 Specification

The requirements outlined above are described in the specification provided below. Basically, it describes the behaviors of the proposed architecture design include the processes involved.

The main purpose of the system is to realize the service discovery and interaction between devices over heterogeneous network. Three entities are involved in the process to carry out the functionality. They are the remote user accessing the local domain over the Internet, local user interact with devices and the residential gateway to control the communication between domains. Figure 3.2, figure 3.3 and figure 3.4 illustrate the use case diagram of the behaviors of the functionality of the modules.

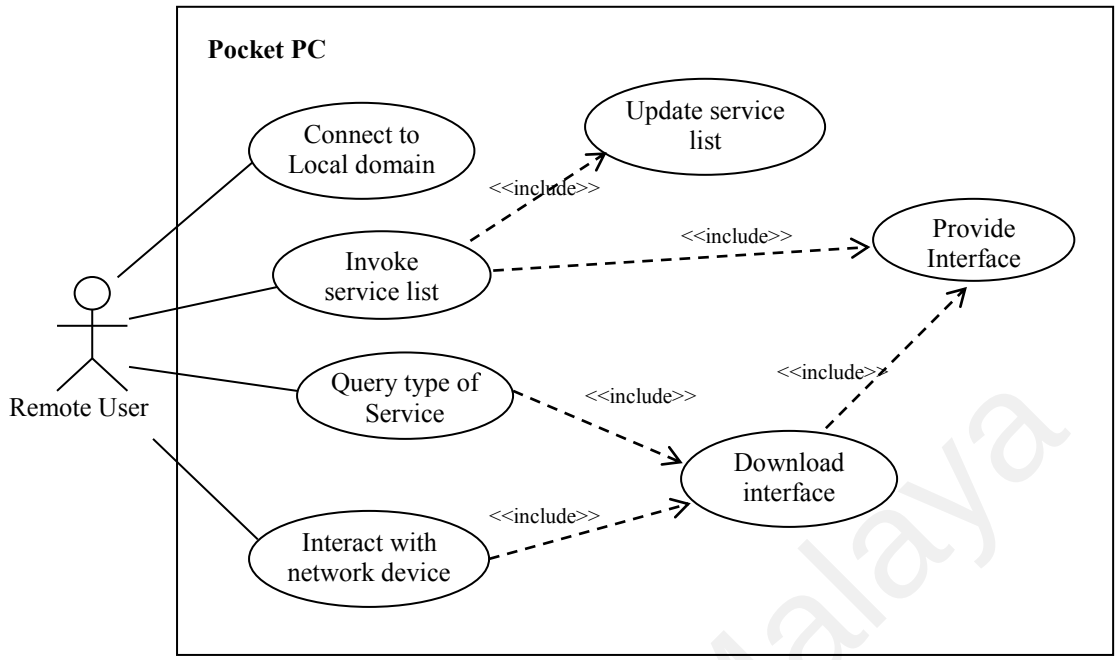

Figure 3.2 Use case diagram for remote user

In figure 3.2, the remote user starts the request for remote communication to the resident gateway. The resident gateway discovers the user device when the user device announces its presence. The resident gateway will request the user to provide password for authentication. The remote user submits the password. If the password is correct, the remote user able to join the domain and receive a list of available services.

Then the remote user sends a request of the selected service for user interface representation. The resident gateway return with the user interface description. The remote user device receives the user interface representation and invokes the selected service by sending a command to the resident gateway. The Resident gateway processes the command and sends it to the selected service. The service will respond to the user device and both interact.

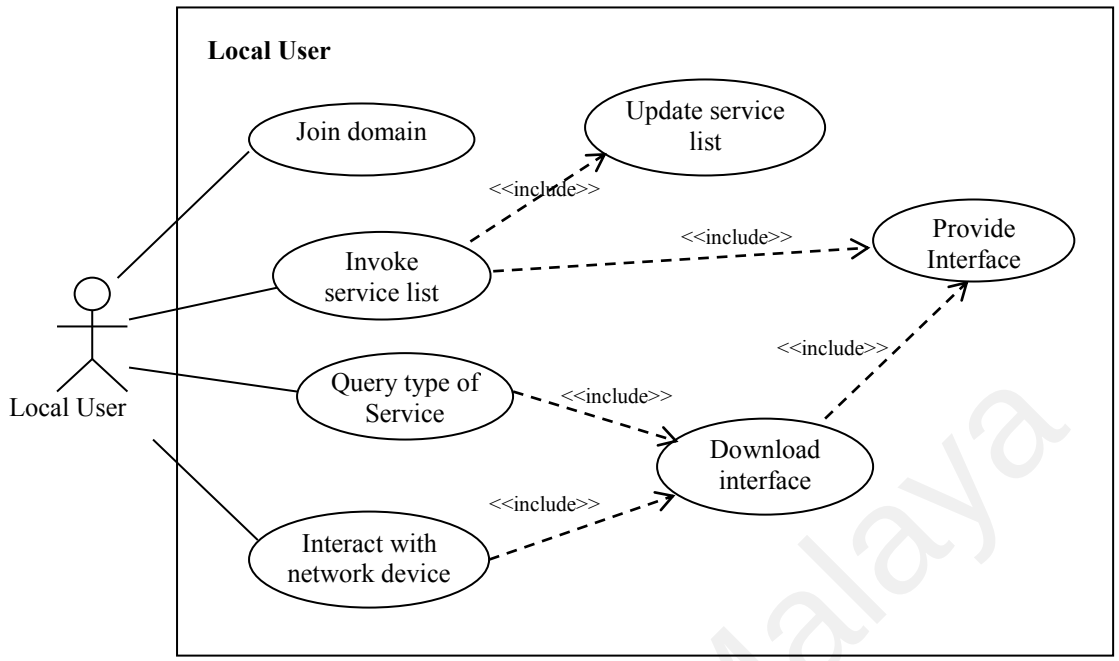

Figure 3.3 Use case diagram for local user

In figure 3.3, the local user starts by sending request to the resident gateway. The resident gateway discovers the user device when the user device announces its presence. Then, local user joins the domain and receives a list of available services.

Then the local user sends a request of the selected service for user interface representation. The resident gateway return with the user interface description. The local user device receives the user interface representation and invokes the selected service by sending a command to the resident gateway. The Resident gateway processes the command and sends it to the selected service. The service will respond to the user device and both interact.

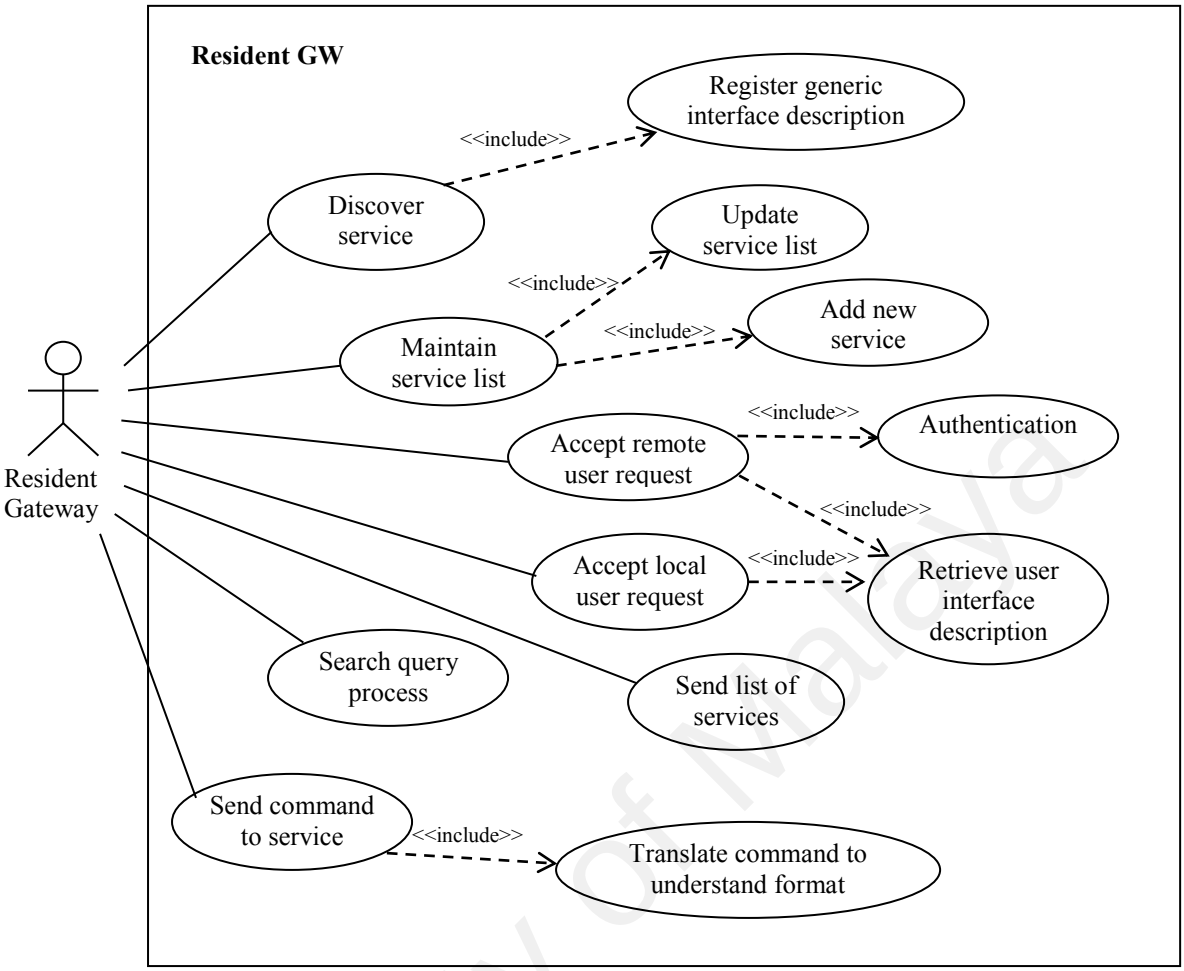

Figure 3.4 Use case diagram for Resident Gateway

In figure 3.4, the resident gateway discovers services and devices and registers them in the service registry. It also maintains the service list with the update statuses of the services. If there is new service, it will add them into the service registry. If the service unavailable or move out from the network, it will remove it from the service registry.

The resident gateway accepts request from remote user and local user. If accept request from remote user, it will send authentication process and the user interface description. When it accept request from local user, it will send the user interface description. Then it sends the list of services to the users.

When the users select the service, the resident gateway will send a command to the specific service. The service will respond and both the user and the service interact.

3.6 Summary

Resulting on the analysis done throughout this chapter, a scenario of the service discovery is defined and all the related functional and non-functional requirements are listed in details. The specifications of the system are defined with the flow of the data in different module.

CHAPTER 4 PROPOSED ARCHITECTURE DESIGN

Chapter 4 illustrates the detailed explanation of the proposed architecture design of services discovery and devices interaction in a heterogeneous environment. This designed is based on analysis of requirements and available technologies. The proposed architecture will meet the objective of this thesis and it is part of the research contribution.

4.1 Physical Design

As discussed in Chapter 3, there are three physical entities involve. They are the remote user accessing the office network over the Internet, the resident gateway and services in the personal area network of an office.

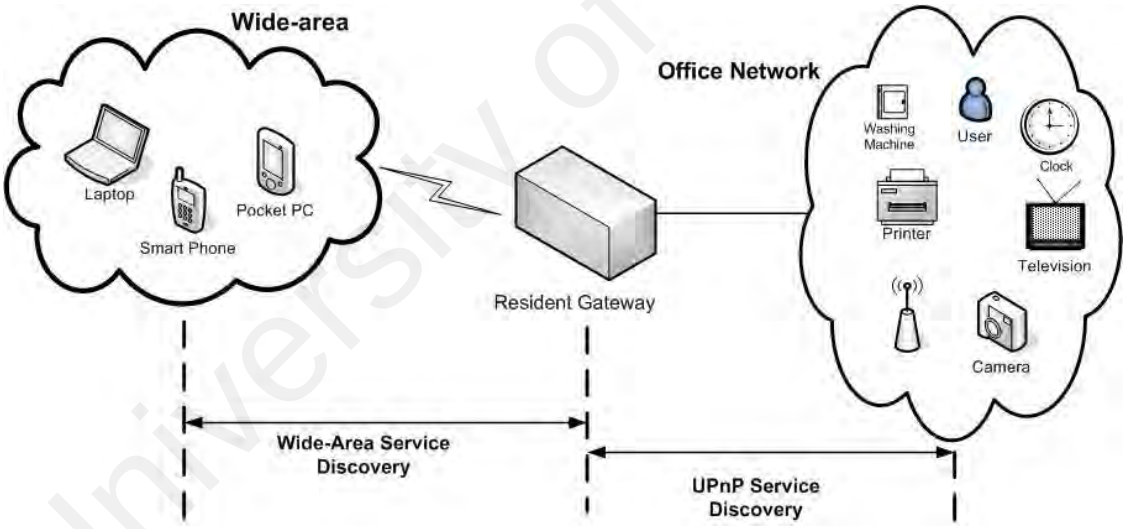

Figure 4.1 Overview of the Physical Design of Service Discovery

4.2 Technologies and Protocols

4.2.1 Protocol used for Wide-Area Communication

As discuss in the previous chapter, there are few requirements to be met for wire-area communication protocol. These requirements need to be addressed by the potential protocol solution. Below are some requirements to be addressed:

- Limited address capabilities
- Security that prevent unauthorized access to PAN devices
- Simple protocol which is suitable for small devices
- Resource limitation

The most suitable technology as part of the solution for wide-area communication is the SIP protocol. SIP technology was chosen based on the functionality provided compared with the suggested requirements. Below provide some rationale for this decision [43].

- Interoperability - Enables communication between devices in the PAN independent of the type of local device communication protocol being used.
- Security - Provide authorization, authentication and encryption when the requested function is executed. Beside that, it also provides encryption to the payload for end-to-end privacy.
- Scalability - SIP is a very scalable protocol. It works well in both Local Area Network and Wide Area Network environments. It has the characteristics that are independent of the transport protocol; it works with the Transport Control Protocol (TCP) as well as with the User Datagram Protocol (UDP).
- Mobility - SIP support mobility concept. A user agent can move from one environment to another and still be discovered. This is possible because of the REGISTER method defined in section 3.3 which informs a SIP server of the user agent new location.
- Extensibility - It allows new methods, new message types, new type or address forms and it supports the four types of communication modes (control, query, event notification, and multimedia session) identified for network devices.
- Service Convergence - It uses existing infrastructure which makes it easier to administer and maintain wide range of services.

4.2.2 Deployment of the Resident Gateway

The Resident Gateway plays an important role because it is the central control of communication of user device from Internet and the PAN networked devices. The resident gateway accepts wide-area requests from remote users and discovers and manages services in the PAN.

The OSGi platform is the better choice for a services gateway platform in the home or PAN network environment. OSGi is a Java-based framework for delivering services to residential user over network. Services can be configured dynamically by activation and de-activation service or application packages (bundles). OSGi service gateway is able to support services component deployment in a dynamic way. Services can be managed remotely allowing service providers to adapt their products to customer needs.

OSGi provides a very flexible environment for the device management mechanism where devices and services can be installed or uninstalled dynamically. Device services can represent different levels of abstraction. Example, a single device can be seen by various OSGi entities as a UPnP device, as a printer or as a USB device.

OSGi provides a middleware layer that can accommodate different network technologies and it is independent of them. For example, an OSGi platform can interconnect UPnP devices and SIP devices or Bluetooth devices and TCP/IP devices. Figure 4.2 show services install as bundle in the OSGi Gateway.

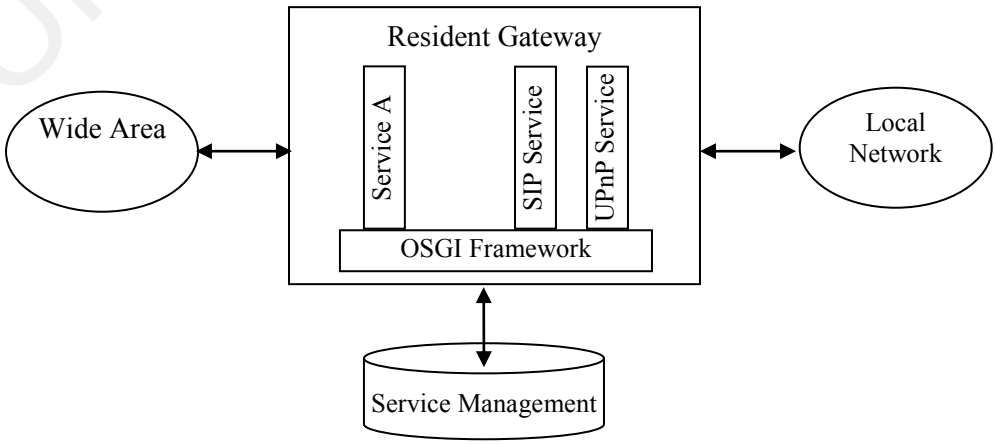

Figure 4.2 OSGi Gateway

In OSGi gateway, all devices and services categories are recorded in the OSGi device service registry. Device service is then responsible for device discovery and registration with the OSGi device registry. OSGi is able to perform inter-gateway bridging over different gateways. Devices registered with one OSGi service registry are exported by bridging and imported into another service registry of other gateway. With OSGi cross-framework device and service mobility is achieved.

OSGi was proposed as the technology to be implemented in the resident gateway has the following features:

- Connecting PAN networks to Internet with broadband access
- Providing routing and address translation
- Bridging various PAN such as home network.
- Enabling secured remote access and data exchange of home devices
- Remote service and device management
- Allowing technologies in WAN and PAN evolve independently

4.2.3 Protocol used for Personal Area Network

There are many service discovery technologies available nowadays for PAN such as Bluetooth Service Discovery, Jini, UPnP, Salutation, SLP, X10 and etc. UPnP was selected out of all these service discovery protocol to communicate devices in the proposed PAN network environment. UPnP is a standard with XML format description for its device. Therefore, it provides rich service description for the device. UPnP is supported on many operating systems and able to deploy application from multiple devices.

UPnP was selected to be the solution for PAN because it is independent of the physical communication medium such as Bluetooth USB and WLAN. Besides that, UPnP also realized the zero-configuration setup and interoperate over multi range of devices. When a new UPnP-enabled device is online, it can be noticed and controlled by a other UPnP device or UPnP Control Point without driver installation.

One of the important reasons, UPnP was chosen because UPnP able to be translated to OSGi method calls. An UPnP service is possible to be imported into OSGi framework and appear as a valid OSGi entity. The installed service is completely accessible by other OSGi entities.

UPnP provide pervasive peer-to-peer network connectivity. There are no registries in the UPnP peer-to-peer network. Users discover services through broadcast and multicast. In the peer-to-peer architecture, network traffic flow increases and single point of failure can be avoided.

4.3 Scenario Architecture Design

This section describes a propose solution for the scenario illustrated in section 3.2.

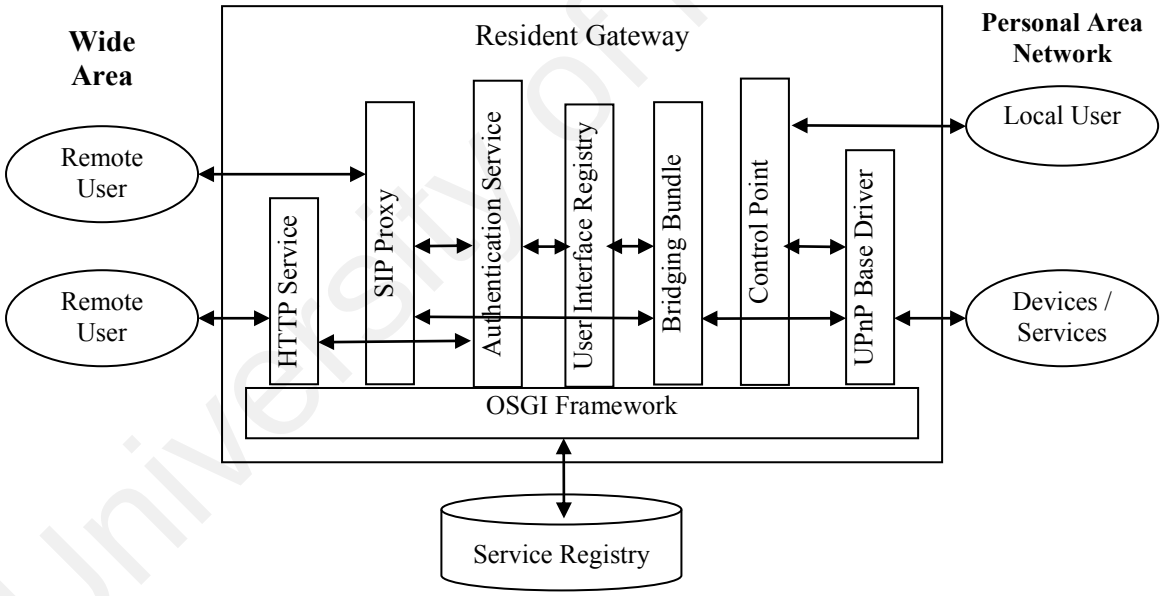

Figure 4.3 Proposed Architecture Design

4.3.1 HTTP Service

This bundle communicates *Remote User* with *Authentication Service* and interacts with *User Interface Registry* and *Service Registry* to retrieve the suitable interfaces and services. This module allows users to interact with the networked devices using the universal user

interface, web browser. Images, resources and other files can be made available through the HTTP service.

4.3.2 SIP Proxy

This bundle similar to the HTTP service which communicate the remote user to local devices. The typical MIME payload type for networked devices is called Device Message Protocol (DMP). This DMP is in XML-based specification. *SIP Proxy* receives messages from remote user and translates the SIP message payload from Device Massage Protocol format to the OSGi method. It also translates the user interface format to the suitable format for the remote device. *SIP Proxy* accesses the *Authentication Service* for authentication, *User Interface Registry* for generic user interface and the *Service Registry* for available services.

4.3.3 Authentication Service

This bundle provides authentication and authorization to remote user before accessing to the PAN devices. Authorized remote user is only able to interact with the devices. After the user is authenticated, *Authentication Service* will allow *SIP* and *HTTP service* to setup session with *Service Registry*.

4.3.4 User Interface Registry

This bundle provides generic user interface descriptions for all the devices and services available in the PAN. The generic user interface description is translated by *HTTP Service* and the *SIP Proxy*. *User Interface Registry* also responsible to update the version of the user interface representation.

4.3.5 Bridging Bundle

Bridging bundle is introduced to perform application layer bridging to translate between diverse device frameworks. Bridging between UPnP and SIP frameworks will utilize both UPnP and SIP OSGi services. The bridging bundle can instantiate UPnP and SIP device objects and translate UPnP events into SIP based event notifications or vise-versa. It can encapsulate UPnP SOAP control message into SIP messages with appropriate message payload and header translations. [47]

4.3.6 Control Point

The Control Point is a bundle that provides the controller function. This function enable user device to interoperate with devices and services available. Control Point is capable of discovering and controlling other devices. Control Point retrieves the device and service description which are register in the service registry. It also can accepts action invocation request and invoke actions to control the service by sending SOAP message. After Control Point subscribe to the event, it will receive event message of any changes of the device state variable. More about Control Point were describe in Chapter 3.

4.3.7 UPnP Base Driver

The UPnP Base Driver bundles translate the messages from UPnP device to the OSGi method calls. It is an OSGi bundle that implements the UPnP protocols. UPnP Base Driver also manages the interaction of devices using UPnP protocol. The responsibilities of UPnP Base Driver are to import UPnP devices from network to the OSGi framework, register them to the Service Registry and export UPnP device from OSGi framework to the network.

4.3.8 Service Registry

Service registry allows bundles to cooperate by registering the service object and search for the requested matching objects. It also receives notification when services become registered

or unregistered. Network devices and services interact with *Service Registry* using *UPnP Driver*. Service Registry stores every registered devices and services available in the network. Services are always registered with an interface name and a set of properties. It has the property details such as names, addresses, user interfaces and etc.

Discovering services is done with notifications or by actively searching for services with specific properties. A simple and powerful filter language is used to select exactly the services that are needed. The OSGi framework supports LDAP query syntax, with operators such as and (&), or (|), not (!), less (<=), greater (>=), approximate (~=), equals (=), substring (*) and etc.

4.4 Architecture Model for UPnP Control Point and UPnP Device

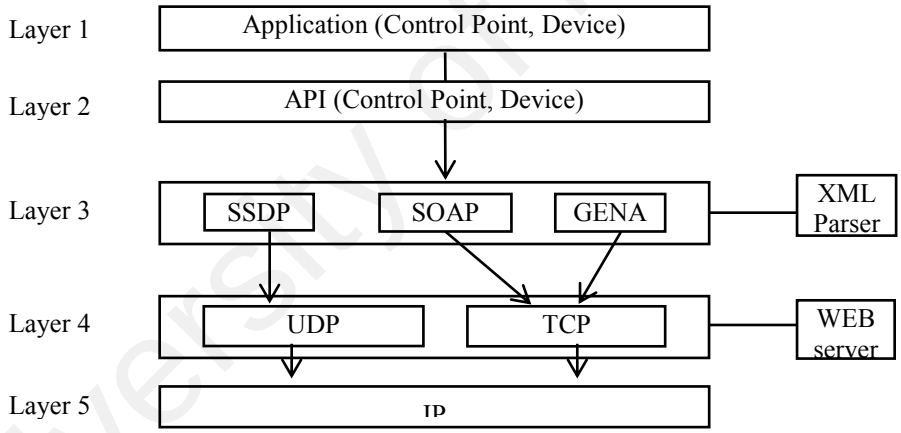

Figure 4.4 Architecture Model of UPnP control point and device

Figure 4.4 describes the architecture model of the design of the UPnP control point and the UPnP device. In this model, there are 5 layers. The top layer is the application which is built on top of the API layer. Layer 2, the API layer is the collection of several classes. The API for both control point and device are implemented in Java on top of the standard protocol. The API is target to embedded systems. Layer 3 is the layer with three components of protocols SSDP, SOAP and GENA. Component in layer 3 are started by the API. Layer 4 represents two protocols UDP and TCP. The SSDP component uses UDP protocol to

implement the discovery, and uses TCP protocol for the controlling (SOAP) and eventing (GENA) function. Layer 5 is the IP layer; therefore all the elements must be IP-enabled.

XML Parser and WEB server do not belong to any of the layer. XML Parser can be invoked by any component in layer 3 to parse XML content. The WEB server is the HTTP server running in the device and handles device and service descriptions.

4.5 General Design

In general, the proposed architecture design consists of three main entities of interaction which are the User (Pocket PC), Resident Gateway (OSGi Gateway) and Local Service (Printer). However, there are two possible scenarios; the user may access the resident gateway as a remote user using SIP or as a local user using UPnP service.

Scenario A: Remote user accessing the Print Service

In this scenario, the interaction between a remote user with SIP-enabled Pocket PC and the local Print Service is proposed with the following processes:

1. SIP enabled pocket pc sends a message to register with the local *SIP Proxy* (in the Resident gateway) to determine its location and availability.
2. *SIP Proxy* will resolve the address of the user device and will refer to *Authentication Service* to authenticate user.
3. After the user is authenticated, *SIP Proxy* will acknowledge the user.
4. Then the SIP enabled pocket pc will send request for user interface for the device.
5. *User Interface Registry* returns the generic user interface.
6. SIP enabled pocket pc request to register with the *Service Registry*.
7. SIP enabled pocket pc request for available services in the *Service Registry.*
8. *Service Registry* will discover local UPnP services using the *UPnP based driver*. *Service Registry* will return the list of available services.
9. The pocket pc selects the wanted service from the list and sends a request to the *SIP Proxy* for invoking the service.
10. *SIP Proxy* will send the invocation request to the *service registry*.

11. Then, the *Bridging Bundle* will perform the bridging between the pocket pc with the requested service which is the print service.

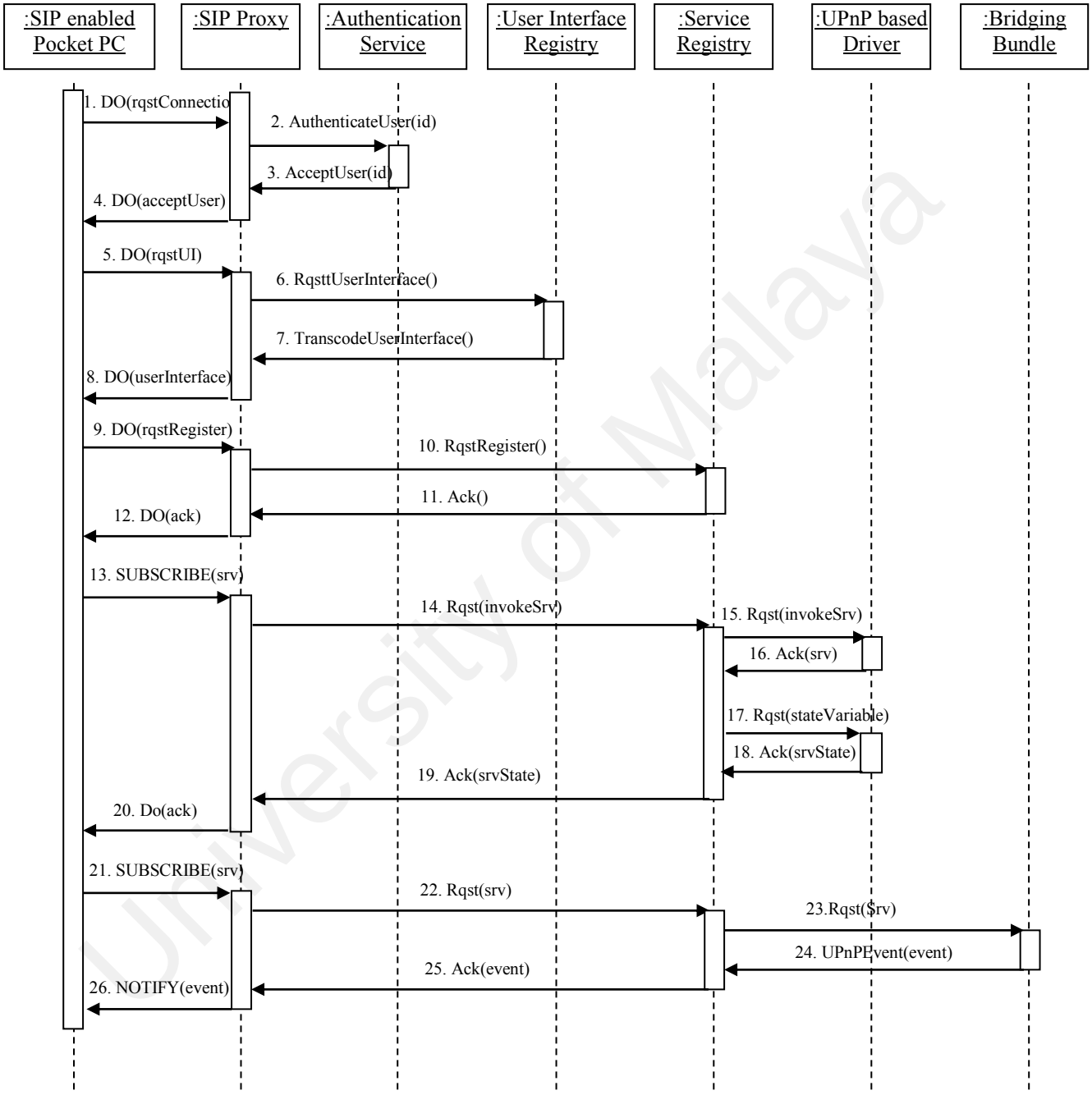

Figure 4.5 A Sequence diagram of a Remote User subscribing to an UPnP service.

Scenario B: Local user accessing the Print Service

In this scenario, the interaction between the local user with UPnP-enabled Pocket PC and the local Print Service is proposed with the following processes:

1. When the UPnP-enabled printer is turn on, it will send a message to the Gateway and register under the uniform interface UPnPDevice in the OSGi Service Registry as standard OSGi UPnPDevice service.
2. The *Control Point* discovers the *UPnP printer service* in the *Service Registry*.
3. *UPnP base driver* export the service to the network.
4. When the UPnP Pocket PC is turn on, it is also registered in the *Service Registry*.
5. The Pocket PC discovers all the UPnP services available in the network including the *UPnP printer service*.
6. The Pocket PC send SOAP message to the printer service to invoke action.
7. The *UPnP base driver* will deliver the action instruction to the device.

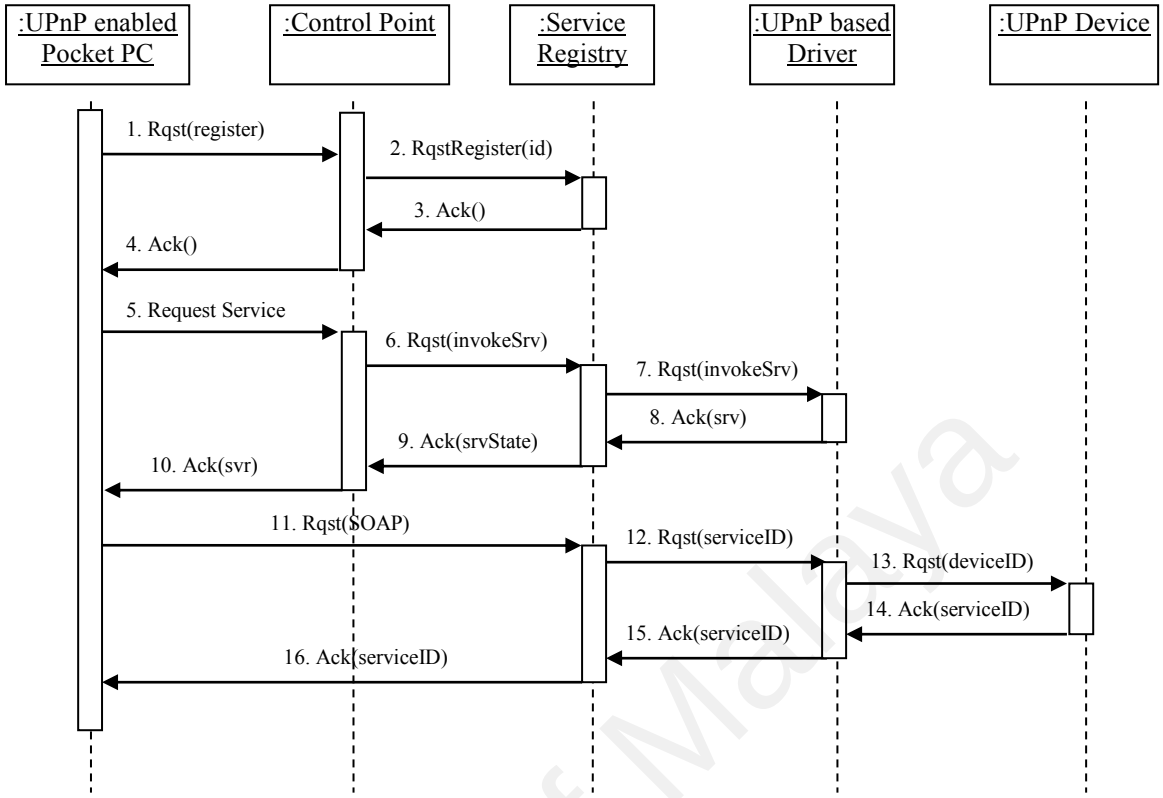

Figure 4.6 A Sequence diagram of a Local User accessing the UPnP service.

4.6 Summary

In this chapter described the physical design of the system together with the layout of the design. All the related technologies and protocols were described in details. This includes the protocol for wide area environment and personal area network. Technologies for the Resident Gateway are also elaborated in details. Besides that, all the modules for the architecture design are described. Finally, the flow of the communication methods was elaborated with sequence diagram.

CHAPTER 5 IMPLEMENTATION

Chapter 5 describes the implementation of service discovery concept to prove the functionality of the proposed proof-of-concept architecture designed in Chapter 4. This implementation was carried out with the technologies, standards and justification described in Chapter 3 and reusing many existing software components as possible.

5.1 Overview

This implementation requires proving the Scenario B explained in Chapter 4 where a user enters into the local personal area network accessing the print service. It is required to prove the device's (Pocket PC) action of subscribing and invoking a service (Print Service). It also proved that the resident gateway able to discover services and perform the UPnP service discovery mechanism.

To realize the proposed scenario, the diagram below outlines the highlighted component to be implemented.

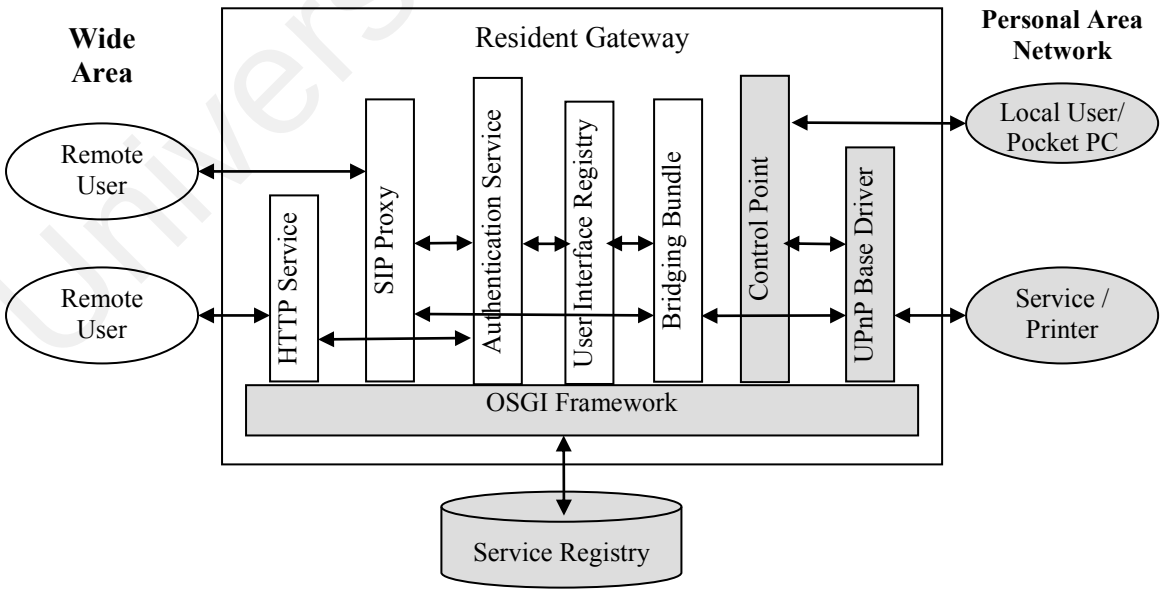

Figure 5.1 Implementation components of the architecture

As highlighted in figure 5.1, these components will be implemented to prove the proposed Scenario B. The resident gateway, a simulated pocket pc (device) and a simulated printer (UPnP service) will be implemented. For the function in resident gateway, they will be implemented as bundles on the OSGi Framework.

5.2 Environment

5.2.1 Resident Gateway Module

As justified in Chapter 3 and Chapter 4, OSGi based platform was chosen to implement the Resident gateway. There are few organizations provide OSGi framework in the market. However, there are two freely available OSGi implementations which are the Oscar OSGi framework and the Knopflerfish OSGi framework. Both of these frameworks are based on the OSGi standard, therefore it is not an issue to select which one. However, Knopflerfish OSGi was chosen as the framework because of its friendly user interface.

The service components are written in Java code. After the service component is written, it needs to be packaged and installed as bundle and run on the OSGi framework. All the resource and code needed to run the bundles are contained in the JAR files. The JAR file also contains a manifest and some combination of Java class files and native code. The bundles implement its own service and provide services to other bundles in the form of interface APIs.

A bundle can only be used after it is installed into the Knopflerfish framework. An installed bundle is uniquely identified by its location or its bundle identifier. The location string which is a unique URL is used to retrieve the bundle JAR. The framework provide bundle management mechanisms which allow installation, activation, deactivation, update and removal of bundles.

When a bundle is installed, it deploys a *BundleActivator* class and a bundle *Manifest* file. A *BundleActivator* class is a Java class that implements *org.osgi.framework.BundleActivator* and defines logic for the *start( )* and *stop( )* methods. The *BundleActivator* attach a set of attribute value pairs to the service when registering a service. A BundleActivator use the

fully qualified service name and an optional selection filter in LDAP query syntax over the service properties to look for a service.

A bundle *Manifest* is a descriptor, which are used to declare bundle native features. Examples of native features are like name, version, manufacture information and interaction relationship with other bundles. Both *BundleActivator* and *Manifest* receive the *BundleContext* and are called when the bundle is started and stopped.

A *BundleContext* object enables the bundle to access the OSGi framework functionality. The methods in the *BundleContext* object allow the bundle to install itself, register services, get information about other installed bundle, retrieve references to services, get and release service objects, get and release file object, and subscribe to events published by the framework.

Below are the steps that are needed to create a bundle [50]:

1. Step 1: Write a bundle Activator class
2. Step 2: Create a bundle Manifest file as a text file
3. Step 3: Build a JAR file that contains the compiled Activator class and the Manifest file
4. Step 4: Start the OSGi framework and install, run and stop the bundle

After a bundle is created, it will be installed into the OSGi framework. The framework creates a *BundleContext* object when it starts a bundle and passes the object to the *start()* method in *BundleActivator*. When a bundle is stop, *BundleContext* object is deleted and the *stop()* method in the *BundleActivator* is invoke. Therefore, the start and stop of the bundle depend on the method in the *BundleActivator*. Below are the mechanisms to start and stop a bundle.

Start bundle

- OSGi framework create the *BundleContext* object
- *Manifest* file refers *BundleContext* object to the method in *BundleActivator*
- *BundleActivator* invoke the *start()* method contained in the class and the bundle is started.

Stop bundle

- *BundleActivator* invoke the *stop()* method contained in the class
- The *BundleContext* is deleted and the bundle is stopped.

5.2.2 UPnP Pocket PC and Printer Module

In this simulation, a user interface running on a Pocket PC to control the printer using two services that are to turn on the printer power and to select the printer print quality. An UPnP system consists of a pocket PC and a printer is constructed. The UPnP Pocket PC and the UPnP printer are simulated in an UPnP network, while the resident gateway running on the OSGi platform as a control point. The control point will discover both the Pocket PC and the printer and the Pocket PC will discover the printer services.

These devices are developed using UPnP API in Java using Eclipse 3.1.1 SDK. This UPnP API which is based on the sample code from CyberLink [52] is able to simulate the UPnP device as describe in the scenario requirement in Chapter 3.

5.3 Implementation

5.3.1 Resident Gateway Module

To implement the resident gateway, firstly the Java2 SDK 1.4.2 was installed. Then the Knopflerfish OSGi 1.3.4 framework is installed on top of the Java SDK. Knopflerfish OSGi can be installed and started by double-clicking on the executable jar file *framework.jar* in knopflerfish.org/osgi directory. This will starts the Knopflerfish OSGi Desktop and a set of bundles as in Figure 5.2. This desktop displays the graphical view of the OSGi framework. The Knopflerfish OSGi desktop able to perform operation such as Install, Start, Stop, and Update on the bundles. Eclipse 3.1.1 with the Knopflerfish Eclipse Plug-in is used in the coding.

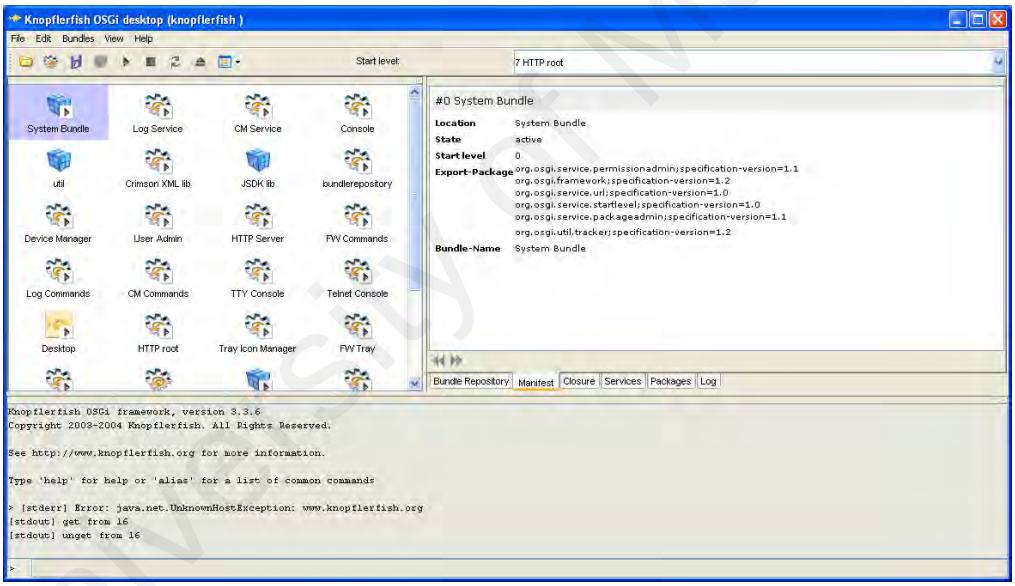

Figure 5.2 Knopflerfish OSGi Desktop

The section below discuss on the implementation of the bundles on the resident gateway OSGi framework. It will also explain the services offered and used by the bundle.

5.3.1.1 Control Point Bundle

An instance of ControlPoint class is created. *ControlPoint::start()* is used to activate the control point. When the control point is activated, it multicasts discovery message to the UPnP network automatically searching for all devices.

```
public static void main(String args[]){
	CtrlPoint ctrlPoint = new CtrlPoint();
	ctrlPoint.start();
}
```

When a UPnP device is added or removed from the network, the control point receives notify event from device. This notification alert is helpful to know the status of the device. Below is the sample code.

```
public class MyCtrlPoint extends ControlPoint implements NotifyListener {
	public MyCtrlPoint() {
		addNotifyListener(this);
		start();
	}
	public void deviceNotifyReceived(SSDPPacket ssdpPacket) {
		String uuid = ssdpPacket.getUSN();
		String target = ssdpPacket.getNT();
		String subType = ssdpPacket.getNTS();
		String location = ssdpPacket.getLocation();
	}
}
```

To get the discovered device list, *ControlPoint::getDeviceList()* is used. Below is the example of the code.

```
ControlPoint ctrlPoint = new ControlPoint();
...
ctrlPoint.start();
...
DeviceList rootDevList = ctrlPoint.getDeviceList();
int nRootDevs = rootDevList.size();

for (int n=0; n<nRootDevs; n++) {
	Device dev = rootDevList.getDevice(n);
	String devName = dev.getFriendlyName();
	System.out.println("[" + n + "] = " + devName);
}
```

Write a bundle Activator class for Control Point

The *BundleActivator* is the main class of the application. For a bundle to start, register or use other service, must have a *BundleActivator* implementation and reference to the *BundleActivator's* class name in its manifest file. The codes below are modified from the Domotics Software [51]. More complete code Control Point is shown in Appendix A.

```
public class Activator implements BundleActivator {
public static BundleContext context;
private ControlPoint cp;

public void start(BundleContext context) throws Exception {
              Activator.context = context;
                     cp = new ControlPoint();
       }

public void stop(BundleContext context) throws Exception {
              cp.close();
              Activator.context=null;
       }
}
```

Control Point bundle Manifest file

The *Manifest.mf* file was created in the folder named META-INF. Manifest.mf describes bundle metrics and its interfaces with other bundles.

```
Manifest-Version: 1.0
Bundle-Description: OSGi Generic Control Point to control UPnPDevice services
Bundle-Name: Generic Control Point 2.1.0
Bundle-Version: 2.1.0
Bundle-Activator: org.osgi.framework.controlpoint.Activator
Import-Package: org.osgi.framework, org.osgi.service.upnp,
org.osgi.upnp.extra.util, org.osgi.upnp.extra.controller
```

5.3.1.2 UPnP Base Driver Bundle

UPnP base driver bundle is first installed on the OSGi framework, then it is started to allow UPnP devices to interact with OSGi framework. UPnP base driver bundle is used to export the UPnP devices on OSGi framework to the other network and to import the UPnP services from other network to the OSGi framework. Any other bundles need the UPnP API specified by the OSGi Alliance to interact with UPnP base driver. UPnP base driver bundle was recently made available open source by Domotics Software [51]. The *BundleActivator* and the *Manifest* code for UPnP Base Driver is shown in Appendix A.

UPnP base driver consists of the device discovery function which contains the related main class that is *RootDeviceListener.* This class responsible to listen for any available device in the network. It is used to discover device. The complete code for this class is shown in Appendix A.

```
public class RootDeviceListener implements ServiceListener {
        private ArrayList devices;
        private DeviceNodeListener listener;
        public RootDeviceListener(){
                devices = new ArrayList();
        }
        public void setDeviceNodeListener(DeviceNodeListener listener){
                this.listener = listener;
        }
}
```

5.3.1.3 Generic User Interface

Generic User Interface or the device servlet has two main functions. These functions are to respond to service description request from device and respond to function invocation request. It is implemented using standard *javax.servlet* API. It uses the HTTPRequest and HTTP Response object to communicate between devices. Generic User Interface also interacts with the service registry to retrieve the proper service object and interact with them.

5.3.1.4 Service Registry

The service registry [32] contains and manages multiple service instances. Bundles that are installed on the OSGi framework will register in the service registry. These registered bundles are giving the Configuration Admin service the ability to create and configure instances of services that the bundle can provide. Each of the service instances is represented by a factory configuration object in the persistent storage of the Configuration Admin service. When the configuration is updated, the Configuration Admin service calls the *ManagedServiceFactory* method to update with the new properties [32]. A new factory instance is created by the managed service factory when *updated* is called.

5.3.2 UPnP Pocket PC and Printer Device

To create the device or service, there are two things that need to be done. First, the XML description file is written using the service description language. Second, the service is implemented. The service description includes name of the service, service type, a list of keywords, a list of functions available, and a list of properties. The description type are related and influenced by the service function offered. Therefore, the description type will be taken into consideration when create the service device.

As describe in the scenario, for the pocket pc to be able to discover the printer and connect to it, the discovery process has two steps; pocket pc announces it's present to be discovered by the control point in the resident gateway and the control point sends out a discovery message to discover the printer device.

5.3.2.1 Device Service Advertisement

The UPnP Pocket PC will announce its presence once it is switch on. When the UPnP Pocket pc is started it sends out *ssdp::alive* message automatically to a fixed multicast address. The control point sends a search request to the UPnP network, the active Pocket PC send the search response to the control point. Then, control point register this Pocket PC in the service registry with a uniform interface.

When Pocket PC is stopped by sending out *ssdp::byebye* message, control point delete it from service registry. Below show the Sequence diagram of the device service advertisement.

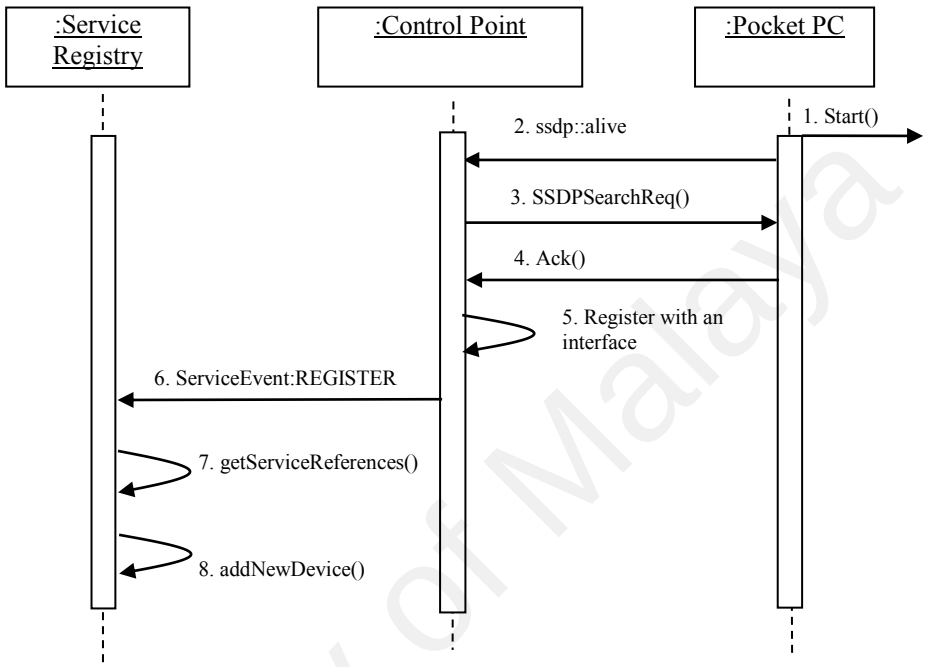

Figure 5.3 Sequence Diagram of Device Service Advertisement

5.3.2.2 Device Discovery

First, the Control Point is plugged into the network and started. Then, the SSDP sends out multicasts search request to search for any UPnP services that are connected to the network. If there is any UPnP device available on the network, it will respond to the request. Then the newly discovered Printer service will be registered by the Control Point in the OSGi framework *org.osgi.service.upnp.upnpDevice* as a uniform interface. The Service Registry bundle sends a service listener *addServiceListener()* object to listen to any service update. This service will be added and registered in the service registry. When there is any change of service state variable, the service references, *getServicereferences()* object is invoke and update the service registry. Below show the Sequence diagram of the discovery request.

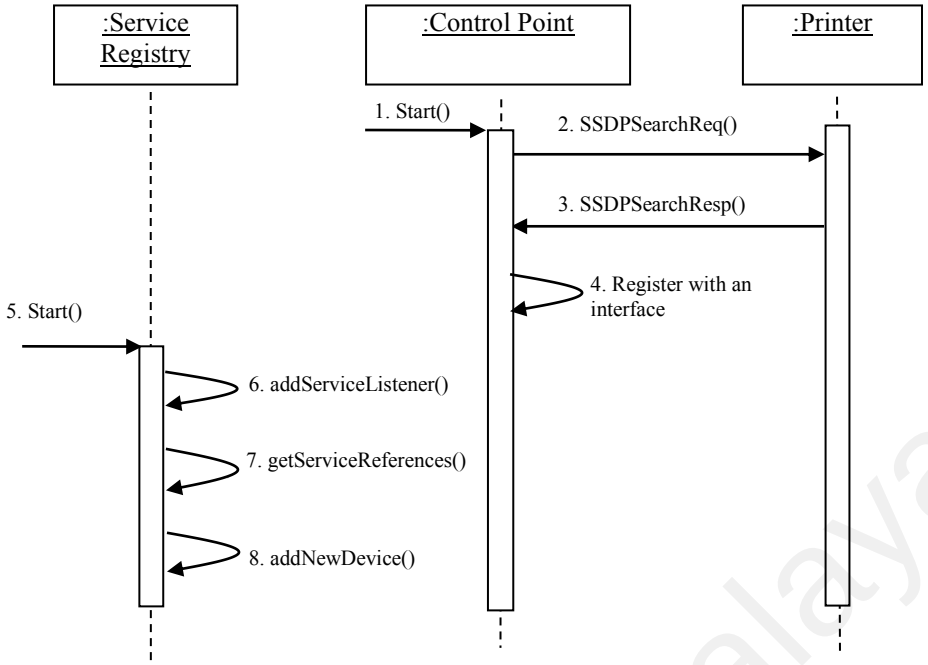

Figure 5.4 Sequence Diagram of Device Discovery

5.4 Testing

This section describes the testing that was performed during the implementation. These tests were done according to the defined requirements as in Section 3.3 and 3.4. The results are obtained, evaluated and presented in detail.

Scenario A: Pocket PC user on UPnP platform discover printer service on UPnP platform

In this scenario, both the pocket pc and the printer service were simulated on UPnP platform. Firstly, the remote control for printer and pocket pc was simulated using Eclipse on UPnP platform as in Figure 5.5 and Figure 5.6.

Right click on the PrinterRmtCtrlFrame.java file and select Run As Java Application. Then the remote control of the pocket pc and the printer will be displayed as in Figure 5.6.

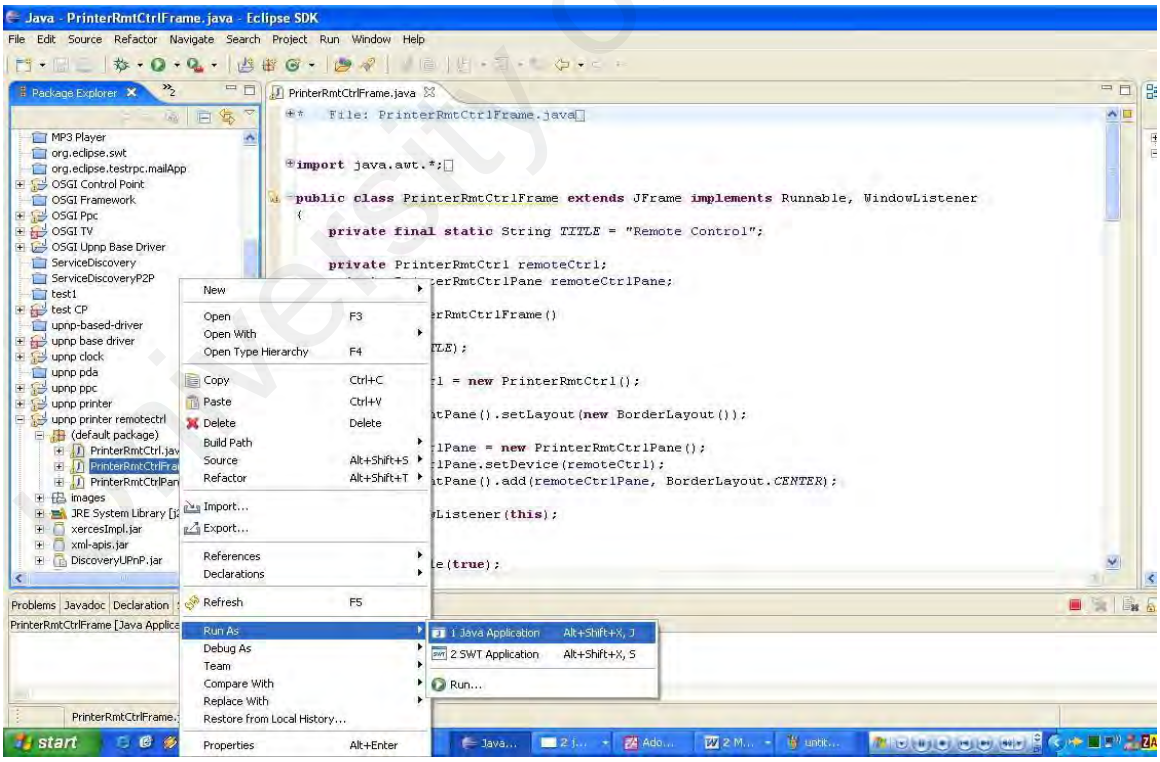

Figure 5.5 To run the simulated Remote Control

Figure 5.6 Remote Control for Pocket PC and printer

Then the printer was simulated on UPnP platform as in Figure 5.7 (a). When the power on the remote control is click, the printer is turn on as in Figure 5.7 (b).

Figure 5.7 Simulated Printer on UPnP Platform (a) Off (b) On

The pocket PC was simulated on UPnP platform as in Figure 5.8 and Figure 5.9. Right click on the PpcFrame.java file and select Run As Java Application. Then the pocket pc will be displayed as in Figure 5.9 (b).

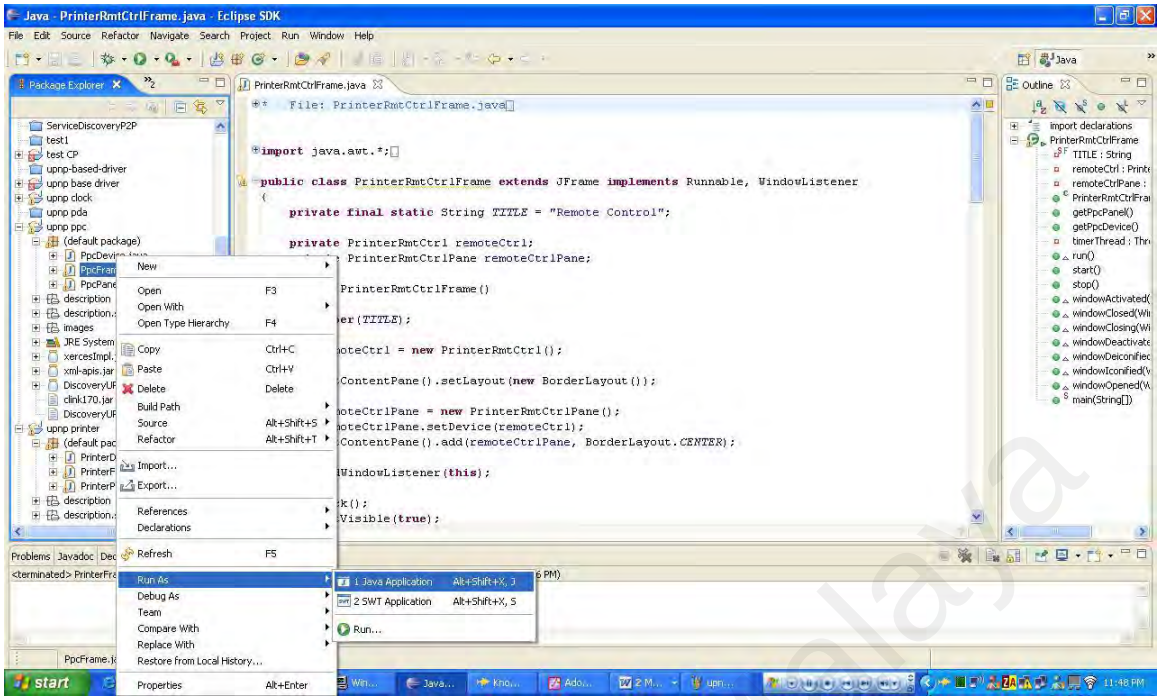

Figure 5.8 To run simulated UPnP Pocket PC

Figure 5.9 Simulated Pocket PC on UPnP Platform (a) Off (b) On

Figure 5.10 shows the three simulated UPnP components.

Figure 5.10 Simulated Printer and Pocket PC before Printer ON

Figure 5.11 shows that the printer was switch on by clicking on the printer power on the remote control. The pocket pc discovered the printer service and the text “UPnP Printer: on” displayed on the pocket pc.

Figure 5.11 Simulated Printer and Pocket PC after Printer ON

The printer was off by clicking on the remote control. Then the text on the pocket pc changed to “UPnP Printer: off”.

Figure 5.12 Simulated Printer and Pocket PC after Printer OFF

Scenario B: Pocket PC on OSGi platform discover clock service on UPnP platform

In this scenario, the pocket pc service was simulated on OSGi platform and the clock service was simulated on UPnP platform. The pocket pc discovered the clock device and displayed the current time on the screen.

Figure 5.13 displayed the steps to run the simulated Pocket PC using Knopflerfish. At the Knopflerfish desktop, click on the Simulated OSGi Pocket PC and run. When the Pocket PC was simulated, the Pocket PC will display as in figure 5.14 (a). Then the UPnP clock was simulated and displayed as in figure 5.15. The OSGi Pocket PC was able to discover the service from clock and displayed the current time on the Pocket PC screen in figure 5.14 (b).

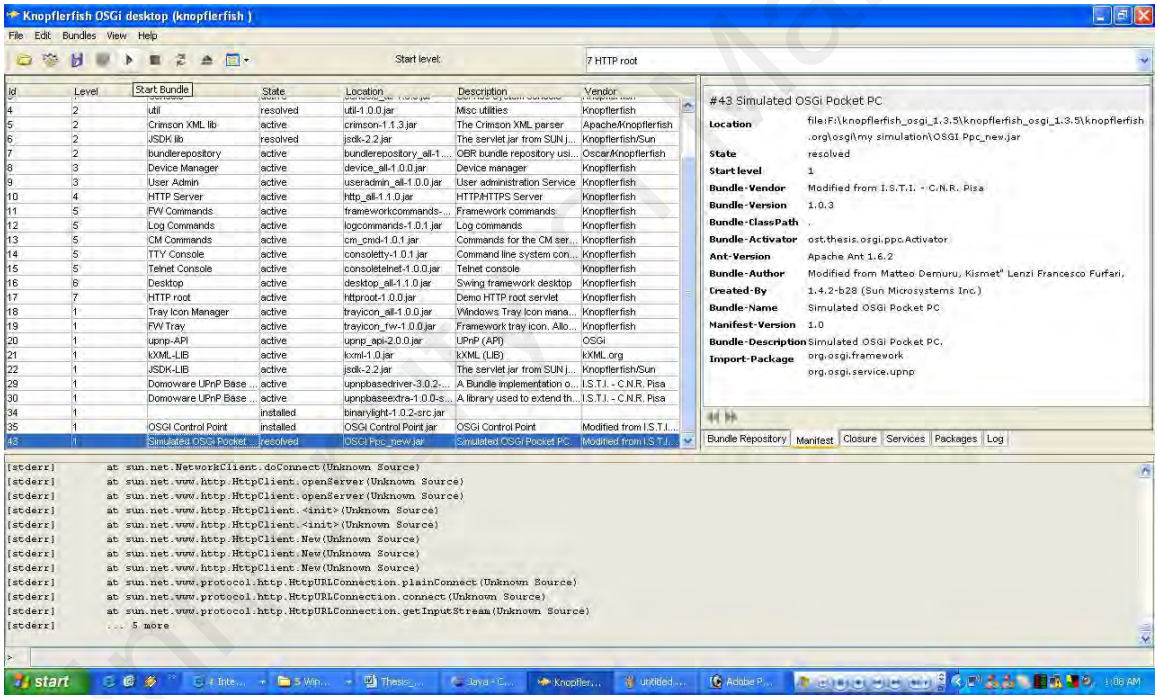

Figure 5.13 To run OSGi Pocket PC using Knopflerfish

Figure 5.14 Simulated OSGi Pocket PC

(a) UPnP clock OFF (b) UPnP clock ON

Figure 5.15 Simulated UPnP clock

Figure 5.16 displayed the simulated OSGi Pocket PC and UPnP clock after UPnP clock is switched on.

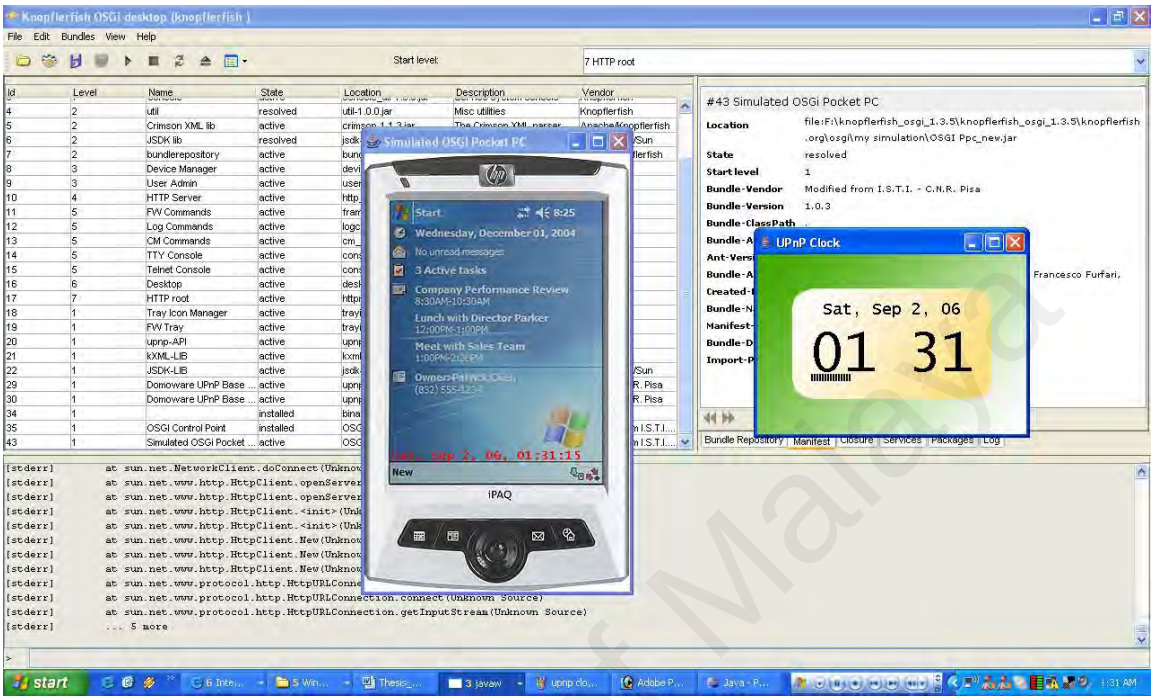

Figure 5.16 Simulated OSGi Pocket PC after UPnP clock ON

5.5 Summary

From the implementation above, the proposed component are implemented and tested. The result shows that the chosen technologies and tools are competent to implement the scenario. It proved the architecture reasonably well and the concept of UPnP service discovery in office personal area network. The service discovery is able to provide service availability and service information to the requesting device. The whole process of the discovery was unknown to the user. The services were displayed on the screen of the user's device.

After the simulation was tested, the performance of the implemented architecture was analyzed. The latency of the commands and requests sent between the devices was almost instantaneous in almost all the situations. The messages were also passed through the framework almost instantaneous. During the implementation, a big number of bundles were installed and started on the OSGi framework to test the performance. This may proved that

in such a scenario, it can still cope with large number of user devices and services. There was no degradation in performance with the load.

The implementation and the testing were tested with some java clients and they were connected to the gateway server simultaneously. There was no significant difference in performance with the implementation. All the Java clients were subscribed to receive UPnP notifications.

However, the service discovery solution for the external wide area communication with SIP technology was not implemented because it is beyond the scope of this thesis. This development with the expected tools which is SIP can be carried out as part of the future work.

There are no major issues that arise from the implementation. Lately, there are efforts being carried out to develop driver bundles in the OSGi platform with other protocols such as HAVi, Bluetooth, Jini and other communication protocols. This will surely solve many connectivity issues with other various scenario implementations.

CHAPTER 6 CONCLUSIONS AND FUTURE WORK

6.1 Conclusion

This thesis describes the proposed architecture design of service discovery in heterogeneous environment and implementation of service discovery using UPnP over OSGi platform in an office personal area network. The technology envisioned for this thesis has turned out to be feasible, flexible and future-proof. It achieves this flexibility by being open rather than proprietary standards. It is absolutely not because that the chosen technologies will remain the main solution in the future, but it is likely they will be strong candidates to provide personal area network service discovery solution.

Resulting from the implementation and testing done, this thesis has fulfilled the objectives and goal of the research which is to illustrate an architecture design of interaction of devices and services in a heterogeneous environment. A detail studied had been done on all the related technologies required. A scenario was implemented in a simulation way to demonstrate the service discovery concept for an office environment. This service discovery is able to provide service availability and service information to the requesting device. All the processes of discovery are completely unknown to the user and it's carry out behind the communication. The contribution of this thesis is the architecture design for the service discovery over different environment.

The technologies used in this prototype have worked well together. The Java tool was a good solution to implement the functions and work well on UPnP and OSGi framework. An environment with the functional components has successfully implemented. A scenario was presented where the printer service can be discovered via resident gateway by a pocket pc when it is turn on. Any changes to the status of the printer service were updated on the pocket pc.

Many existing service discovery protocols have been surveyed. In this thesis, UPnP and SIP has been studied and justified as the PAN and WAN communication technology respectively. Although there is no exact description on how to do translation between the UPnP and SIP protocols, some studies and explanations has been done on both protocols.

OSGi standard was selected technology to deploy the resident gateway. Therefore, some research was carried out on it. The standardized OSGi gateway specification is supported by many famous companies and developers so that it can become the standard for service gateways in the near future.

Much effort was spent on the implementation of the OSGi framework resident gateway, the UPnP base driver, control point and UPnP pocket pc and printer. Based on this implementation, it is suitable for any possible extension by adding OSGi bundles for other functions in the future.

6.2 Future Enhancement

This thesis describes the service discovery in heterogeneous environment and demonstrated the UPnP technology to discover services. Future work that could be enhanced on the architecture would be the SIP service. SIP service enables device and service mobility. The SIP proxy is possible to be implemented by installing on the proposed architecture. The implementation of the SIP Service for the wide area communication will improve the functionality of the remote user. Currently, there are groups of expert carrying out studies on the details of the SIP service API.

Another option of enhancement is on the security aspect of the whole architecture. Authentication and authorization can be added when the user device trying to access to the personal area network services. This is to prevent unauthorized access to the services and to protect the confidentiality of the services.

REFERENCES

[1] MarkWeiser. (1991). The Computer for the 21st Century, Scientific American.

[2] Christian Bettstetter. (September 2000). Toward Internet-based Car Communications: On Some System Architecture and Protocol Aspects. In *Proceedings EUNICE 2000, Sixth EUNICE Open European Summer School*, Twente, Netherlands.

[3] M. Satyanarayanan. (August 2001). Pervasive Computing: Vision and Challenges. *IEEE Personal Communications*.

[4] Tom Broens. (July 2004). Context-aware, Ontology based, Semantic Service Discovery. Master's thesis, University of Twente, The Netherlands.

[5] Bluetooth Special Interest Group (SIG). "Bluetooth Protocol Architecture version 1.0", White Paper.

[6] Bluetooth Special Interest Group (SIG). "Service Discovery Protocol". SIG Specification version 1.0, Volume 1.

[7] Jini™ Architecture Specification, Sun Microsystems, Version 1.2, (December 2001). Retrieved 20 August 2006, from http://wwws.sun.com/software/jini/specs/.

[8] Salutation Architecture Specification, Salutation Consortium, Version 2.0c, (June 1 1999). Retrieved 15 June 2006, from ftp://ftp.salutation.org/salute/sa20e1a21.ps.

[9] S. Czerwinski, B. Y. Zhao, T. Hodes, A. Joseph, and R. Katz. (1999). An Architecture for a Secure Service Discovery Service, Fifth Annual International Conference on Mobile Computing and Networks (MobiCom '99), Seattle, WA.

[10] E. Guttman, C. Perkins, J. Veizades, and M. Day. (June 1999). Service Location Protocol, Version 2, IETF, RFC2608. Retrieved 16 June 2006, from http://www.ietf.org/rfc/rfc2608.txt.

[11] Universal Plug and Play Device Architecture. Microsoft Corporation, Version 1.0, (08 June 2000). Retrieved 30 July 2006, from http://www.upnp.org/download/UPnPDA10_20000613.htm.

[12] J. O'Sullivan et al. (April 2002). Service Description: A survey of the general nature of services.

[13] D. Chakraborty et al. (December 2000). "Service discovery in the Future Mobile Commerce", ACM crossroads volume 7 Issue 2.

[14] R. Hull et al. (June 2003). "E-services: A Look Behind the Curtain".

[15] E. Guttman, C. Perkins, J. Veizades, M. Day. "Service Location Protocol, Version 2". Retrieved 20 July 2006, from http://ietf.org/rfc/rfc2608.txt.

[16] Retrieved 22 July 2006, from Sun's Jini Homepage: http://www.sun.com/software/jini

[17] Cristoph Renner. (April 2000). Diplomarbeit: Service Discovery and Profile Mobility in a Car Environment. Master's thesis, Institute of Communication Networks, Munich University of Technology.

[18] Salutation Consortium. Salutation Architecture Specification 2.0c. Retrieved 22 July 2006, from http://www.salutation.org.

[19] P. St. Pierre and T. Mori. Salutation and SLP. Retrieved 22 July 2006, from http://www.salutation.org/techtalk/slp.htm.

[20] Universal Plug And Play (UPnP) Forum, 2001. Retrieved 22 July 2006, from http://www.upnp.org.

[21] Universal Plug and Play Device Architecture, Version 1.0 1999-2000 Microsoft Corporation. Retrieved 23 July 2006, from http://www.upnp.org/download/UPnPDA10_20000613.htm

[22] C. Bettstetter and C. Renner. (2000). A Comparison of Service Discovery Protocols and Implementation of the Service Location Protocol. Institute of Communication Networks, Munich University of Technology.

[23] Open Services Gateway Initiative, (August 2005). "OSGi Service Platform Core Specification, Release 4".

[24] D. Marples, P. Kriens. (December 2001). "The Open Services Gateway Initiative: An Introductory Overview", IEEE Communications Magazine.

[25] D. Valtchev, I. Frankov. (April 2002). "Service Gateway Architecture for a Smart Home", IEEE Communications Magazine.

[26] P. Dobrov, D. Famolari, C. Kurzke, B. Miller. (August 2002). "Device and Service Discovery in Home Networks with OSGi" , IEEE Communications Magazine.

[27] S. Chemishkian. (January 2002). "Building Smart Services for Smart Home", Workshop Proceedings of 4th IEEE International Workshop on Network Appliances.

[28] R. Anderson. (2001). "Security Engineering: A Guide to Building Dependable Distributed Systems", John Wiley & Sons.

[29] Mikko Ahola. (6th May 2003). "Machine-to-Machine Communications for Enhancing E-Commerce Logistics", Master's thesis.

[30] OSGi alliance homepage. Retrieved 2 August 2006, from http://www.osgi.org

[31] Oscar- OSGi framework implementation. Retrieved 2 August 2006, from http://oscar.objectWeb.org

[32] Knoplerfish project. Retrieved 2 August 2006, from http://www.knopflerfish.org

[33] Physalis project homepage. Retrieved 3 August 2006, from https://developer.berlios.de/projects/physalis

[34] Jeffree homepage. Retrieved 3 August 2006, from http://jeffree.objectWeb.org

[35] Stan Moyer, Simon Tsang et al. (September 2000). "Requirements for Networked Appliances: Wide-Area Access, Control, and Interworking", Internet Draft draft-tsang-appliancesreqs-01.txt.

[36] J Plomp. (January 2001). UIML in future home environments. Submitted to the first UIML conference held in Paris.

[37] SH Maes et al. (2000). Multi-modal interaction in the age of information appliances. IEEE International Conference on Multimedia and Expo.

[38] Stajano F. et al. (2002). Security issues for internet appliances. IEEE Symposium on Applications and the Internet (SAINT) Workshops.

[39]. Moyer, S. and Marples, D. (2000). The Internet Alarm Clock – A Networked Appliance Case Study. Telcordia Technologies.

[40]. Rosenberg, J, et. al. (2002). SIP: Session Initiation Protocol. IETF RFC 3261.

[41] Moyer S. et al. (October 2001). A protocol for wide-area secure networked appliance communication. IEEE Communications Magazine, 39:52-59.

[42] Moyer S. et al. (2002). SIP Extensions for Communicating with Networked Appliances. IETF Draft.

[43] A Roychowdhury et al. (2001). Instant Messaging and Presence for SIP Enabled Networked Appliances.

[44] Marples D. et al. (2002). Feature interactions in services for internet personal appliances. IEEE International Conference on Communications.

[45] Tsang S. et al. (2001). Accessing networked appliances using the session initiation protocol. IEEE Conference on Communication, pages 1280-1285.

[46] Network Working Group. Hypertext Transfer Protocol (HTTP 1.1). Retrieved 5 August 2006, from ftp://ftp.rfc-editor.org/in-notes/rfc2068.txt

[47] Bushmitch D. (2004). A sip-based device communication service for osgi framework. IEEE.

[48] UPnP Service Template Version 1.01. Retrieved 7 August 2006, from http://www.upnp.org/resources/documents.asp

[49] UPnP Device Template Version 1.01 Retrieved 7 August 2006, from http://www.upnp.org/resources/documents.asp

[50] Sun Microsystems. (October 2000). How to Write Your First JES Service version 2.

[51] Domoware software. Retrieved 28 July 2006, from http://domoware.isti.cnr.it/.

[52] UPnP development package provider - CyberLink, Satoshi konno. Retrieved 29 July 2006, from http://www.cybergarage.org/net/upnp/java/index.html

[53] Wu Lan et al. (December 2004). Service discovery for personal networks. Master's thesis, University of Stuttgart.

[54] Darragh O' Sullivan. (September 2005). An Advanced Appliance Interaction Architecture. Master's thesis, University of Dublin, Trinity College.

[55] Microsoft Technet website. Retrieved 9 August 2006, from http://www.microsoft.com/technet/prodtechnol/winxppro/evaluate/upnpxp.mspx

[56] Simon Tsang, Stan Moyer, Dave Marples. (November 2000). SIP Extensions for Communicating with Networked Appliances. Telcordia Technologies.

[57] Sumit Khurana, Ashutosh Dutta, Provin Gurung. XML based Wide Area Communication with Network Appliances. Telcordia Technologies, Columbia University.

[58] Jean-Marc Seigneur. (2001). House-Keeper a vendor-independent architecture for easy management of smart homes. University of Dublin.