DLNA MEDIA SERVER FRAMEWORK FOR EMBEDDED SYSTEMS

_____

A Project

Presented

to the Faculty of

California State University, Chico

_____

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

in

Computer Science

_____

by

Ajitabh Prakash Saxena

Spring 2011

DLNA MEDIA SERVER FRAMEWORK FOR EMBEDDED SYSTEMS

A Project

by

Ajitabh Prakash Saxena

Spring 2011

APPROVED BY THE DEAN OF GRADUATE STUDIES
AND VICE PROVOST FOR RESEARCH:

_____
Katie Milo, Ed.D.

APPROVED BY THE GRADUATE ADVISORY COMMITTEE:

_____
Abdel-Moaty M. Fayek
Graduate Coordinator

_____
Anne Keuneke, Ph.D., Chair

_____
Seung-Bae Im, Ph.D.

ACKNOWLEDGMENTS

I want to thank Dr. Seung-Bae Im who, in spite of his busy schedule, accepted to be the member of the committee and for committing his time for the project. I want to thank Dr. Anne Keuneke, the Chair of the committee, who at multiple instances went out of her way to help me every time I requested. Without her, it would have taken me much longer to get to this point. Thank you both for your time and guidance.

I also wish to express my appreciation to my wife Renuka and my daughter Archana for all their support, time and encouragement throughout the course and during the project. I can never return them the time they sacrificed during the course.

TABLE OF CONTENTS

CHAPTER PAGE

LIST OF TABLES

LIST OF FIGURES

FIGURE PAGE

ABSTRACT

DLNA MEDIA SERVER FRAMEWORK FOR EMBEDDED SYSTEMS

by

Ajitabh Prakash Saxena

Master of Science in Computer Science

California State University, Chico

Spring 2011

Devices such as cell phones, set-top boxes, televisions, etc., are generating digital entertainment content at an alarming rate. The challenge is to access and share the content across these devices. Universal Plug and Play (UPnP) and Digital Living Network Alliance (DLNA) specifications enable seamless sharing of digital content across these devices. Many companies developed software development kits for rapid development of UPnP and DLNA compliant devices.

This project identifies some of the limitations of currently available UPnP software development kits with respect to their deployment on embedded systems. The project also addresses these limitations by implementing the UPnP specifications with major emphasis on its deployment in embedded devices. The project considers factors such as memory footprint of SDK, portability across multiple operating systems, abstraction of services, debug ability, development cost and performance. A simple DLNA media server was also developed using the SDK and some open source

components. This effort was undertaken to prove that the framework can be used to develop a DLNA compliant media server that can be effectively deployed on embedded devices.

To test the portability across multiple operating systems the SDK was developed using Microsoft Visual Studio 2008 environment along with Microsoft platform SDK. As each development milestone was accomplished basic testing was performed on Linux and Windows operating systems. After significant work was accomplished the SDK was also tested on a MIPS based embedded system. Behavioral differences on all the three platforms were noted and analyzed.

CHAPTER I

INTRODUCTION

Project Overview

In the last few years, many homes have been transformed into “digital homes.” Devices such as digital televisions, server class desktop computers, digital cameras and mobile phones with digital cameras have become commodities. Most of these devices today have either wired or wireless IP connectivity. As a result of this digital revolution, the rate at which digital entertainment content is produced and consumed has increased exponentially.

As more devices capable of generating and consuming digital content became available the need to share the digital content seamlessly between such devices became a priority. This led to the birth of the Digital Living Network Alliance (DLNA) [1] in 2003 which was responsible for developing protocols and standards to govern seamless sharing of data and allow interoperability between digital devices. The DLNA committee published its first set of interoperability guidelines [2] in June of 2004. DLNA capable devices soon started to appear.

DLNA protocols address the interoperability of devices both in terms of the type of content being shared as well as the capability of other devices to consume it. For example, a DLNA media server device might be exposing different types of media content but other devices on the UPnP network might only be interested in MPEG2 video

content. The DLNA interoperability guideline requires the Media Server to profile the digital content that is being exposed to other devices. A DLNA profile identifies certain parameters of the digital item that is being exposed by the media server. Other devices on the UPnP network will then be able to parse the profile associated with every digital item and consume the content more effectively.

The Universal Plug And Play (UPnP) protocols [3], [4], [5] deal with the networking aspects of devices and form the backbone of DLNA specifications. The DLNA specifications augment the UPnP specifications and provide enhancements related to exchange of data between the devices. The UPnP protocols were developed by the UPnP forum with the following major goals:

- Easy, seamless, and robust connectivity.
- Ad-hoc networking.
- Zero-configuration networking.
- Platform independence.
- Media and device independence.

Today it is almost a mandate that any device capable of generating or consuming the digital content must be DLNA compliant. Embedded devices such as mobile phones, digital televisions, set-top boxes, etc., have full featured DLNA media servers. This also implies that these devices implement UPnP services since they are the backbone of any DLNA implementation.

To accelerate the development of DLNA compliant devices many implementations of UPnP specifications are available in the form of UPnP Software Development Kits (UPnP SDK). Because of reasons noted in this introductory chapter

and in detail in Chapter III of this document, the majority of available UPnP frameworks are not fit to be deployed on embedded systems. This project develops an UPnP framework which addresses limitations of available UPnP SDKs and implements a refined version for use on embedded systems.

Problem Description

As noted, there are many implementations of UPnP standards that are available as UPnP software development kits. This project is an effort to develop an UPnP framework which can be deployed, with little or no effort, on low end embedded systems like mobile phones.

Embedded systems differ from personal computers in terms of available computing resources since they are specialized computing devices. They are designed for a specific application. Hence, the components of embedded systems are chosen depending on its application. Generally, the majority of embedded devices have the following limitations.

- Embedded devices have limited RAM restricting the size of the executable that can be loaded.
- They have limited processing power and may not be multi-threaded.
- Embedded systems usually have limited permanent storage.
- Embedded systems run unconventional operating systems like Windows CE, flavors of Linux and real-time operating systems. These operating systems may provide different sets of services or provide some service using a different set of APIs. This

includes the usage of threads, file IO operations, operating system specific socket operations etc.

Purpose of Project

Although discussed in detail in Chapter III, below are the highlights of major limitations of generic UPnP implementations.

- They have features that are not needed by a particular embedded device.
- They have cost associated with them in terms of royalties, license fees or product cost.
- The implementations do not facilitate the usage of hardware components if available on a particular embedded device. Proper abstractions are needed so that a software implementation can be over written to utilize a particular hardware service.
- The implementations do not cater to the need for deployment on different embedded platforms. This takes away the flexibility of using the same code on a friendly platform and on embedded devices, reducing debug ability.

The purpose of this project was to implement the basic UPnP standards specifically for embedded devices with limited storage and computing power. The implementation tackles the above stated limitations. Specifically, the following guidelines were adhered to:

1. The first guideline for the project was to keep the SDK as light weight as possible. This implied that the implementation could not include the full feature set for UPnP specifications. Only the mandatory features, essential to the functionality of media servers were required.

2. The second guideline was to have abstractions to enable the use of any hardware component if available on a particular embedded platform.

3. Third guideline was to have operating system independence and hence use of proper abstractions was necessary to achieve platform independence. This way the framework could be deployed on variety of environments. This also enables the user of the SDK to develop and debug UPnP media device on a friendly environment like the Visual Studio IDE and then deploy it on the embedded device. Only with a recompilation for the target platform the framework can be deployed on different environments. In order to achieve this goal it was also necessary to have the major components of the framework loosely coupled so that one part of the framework can be modified without major modifications to other parts.

4. Fourth guideline was to make the framework as efficient as possible by reducing CPU intensive tasks. For example, UPnP standards use SOAP for remote procedure calls. The parameters in SOAP are passed using comma separated string values. As string parameters travel from one function to another a copy of memory from a source location to a destination location is needed. This copy is CPU intensive and was a major area of optimization.

Problem Statement

There are many UPnP implementations available to cater to different needs. These implementations are generic in nature and as a result are feature rich. Many of these are not designed with consideration to embedded systems. At the same time, some of these are actually designed for embedded systems. The problems with these UPnP

implementations are that they are not cost effective, not flexible and their functionality cannot be over written without extensive work. If they are open source there are royalties associated with them. If they are not open source they are not cost effective and they cannot be modified which makes it difficult to use any custom hardware features which may be available on a particular embedded system. They often contain features which are not required in all embedded devices since embedded devices like cell phones do not have the computing power needed to support these features. The embedded devices need thin, efficient, modularized, operating system independent implementation that can be debugged effectively.

This project was an effort to create UPnP implementation that can be effectively used across a variety of embedded devices.

Definition of Terms/Acronyms

This section describes the terms that will be used throughout the document. Only the basic aspects of the terms are described. Details about these terms can be found in various references outlined in the reference section of this document. Chapter II also discusses these terms in detail.

CDS

Content Directory Service. This is a service implemented by Media server devices to expose the digital media content to UPnP network.

CMS

Connection Management Service. This is service implemented by Digital Media Servers to manage the connections.

Control Point

A Control Point is an UPnP device that is responsible for invoking actions exposed by different devices on UPnP network.

DLNA

The term DLNA stands for Digital Living Network Alliance. The DLNA alliance is a committee of participating companies and is responsible for development of standards governing the interoperability and compatibility of UPnP devices.

DLNA Media Server Device

DLNA Media Server is an UPnP device which exposes digital content of some form for other UPnP devices to consume. The DLNA Media server implements UPnP protocols so that it can talk to other UPnP devices. It also implements the DLNA protocol to expose the content in correct format so that other DLNA devices on the UPnP network can get the information about the digital content available on the UPnP network.

Media Rendering Device

A Media Rendering device is an UPnP device which can consume a digital content. The control point establishes the connection between the media rendering device and a media server device and starts a streaming session.

SDK

The term SDK stands for software development kit.

UPnP

The Universal Plug and Play (or UPnP) is a set of specifications dealing with different networking aspects. Devices implementing the UPnP specifications are called UPnP devices.

UPnP Media Server Device

UPnP Media Server is an UPnP device which implements only the UPnP protocols and does not implement the DLNA protocols. Such servers can expose the media content to other UPnP devices but the exposed content does not carry any information to indicate how the content can be consumed.

Outline of Document

The remainder of this document is outlined as follows:

Chapter II is the literature overview and provides the reader with sufficient background to understand the UPnP architecture and DLNA guidelines. The understanding of the concepts outlined in this chapter is fundamental for the understanding of the rest of the project.

Chapter III discusses the architecture and limitations of embedded systems and identifies the problems with available UPnP software development kits. The chapter also studies the Intel UPnP framework and outlines its limitations.

Chapter IV describes the project in details including the development phases, development and testing environment, the development methodology and the validation procedures utilized to validate the work.

Chapter V describes the design and data structures use in the project. Certain design choices are discussed and clarified. Important components of the UPnP framework and the Media server are also described in details.

Chapter VI evaluates the goals achieved through this project. The usability and performance analysis of the implementation is done in this chapter. The performance

of the product is measured in terms of CPU utilization and memory usage. The chapter also discusses future enhancements that may be of value to the product.

CHAPTER II

LITERATURE REVIEW

Universal Plug and Play Architecture

This section briefly describes the UPnP architecture [3], [4] and the DLNA interoperability guidelines [2]. Concepts outlined in this section are fundamental for understanding the project.

Figure 1 shows a typical home network with several devices sharing digital multimedia content across different platforms.

Typically, the Dynamic Host Configuration Protocol (DHCP) and Domain Name Server (DNS) servers are available on the router or on the network. The DHCP server manages the IP address assignment while the DNS server is responsible for name resolution. In order to setup the network outlined in Figure 1, typical steps needed are as follows:

- The user assigns names to each device so that the devices can be addressed using the name.
- Digital content will need to be made available on the media server. This will require the user to copy the content on the media server and then share it using some file sharing protocol.

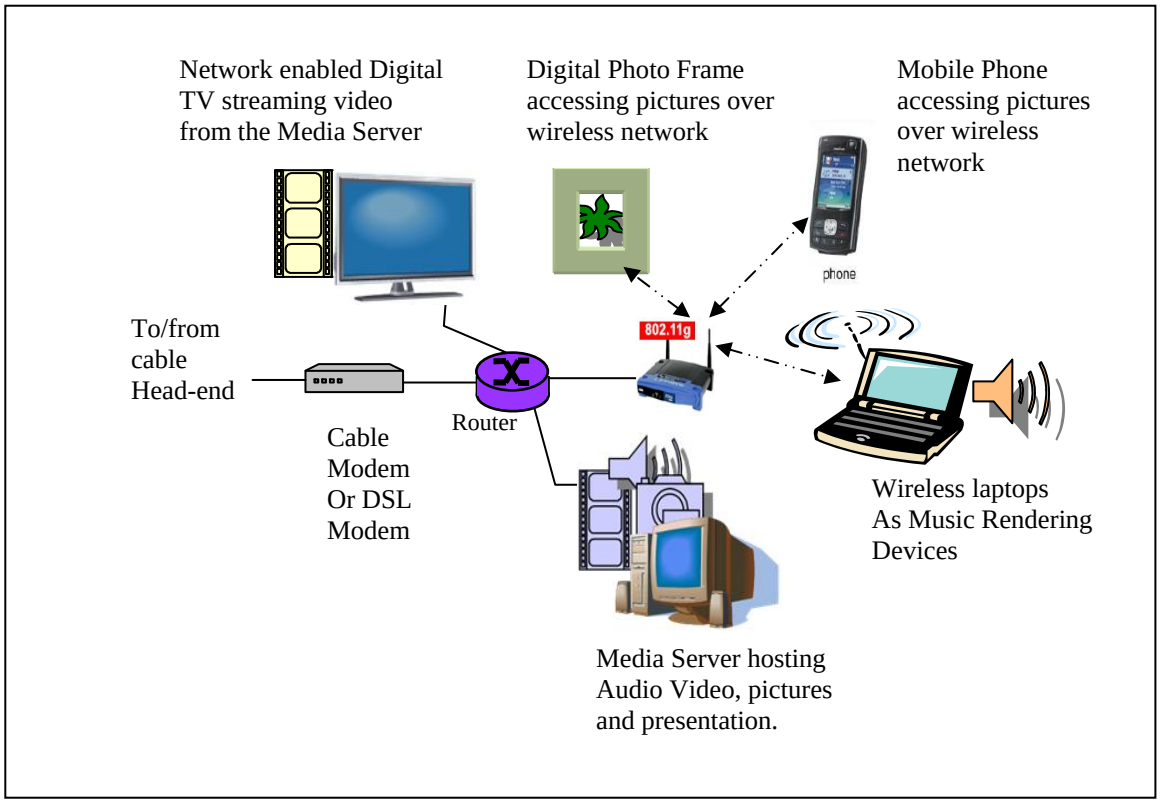


Fig. 1. A typical home entertainment network.

- When a new device is introduced into the network, the device will need to be configured to find the media server and then consume the available media content using a file transfer protocol.

There are some serious limitations with this approach. There is too much manual intervention needed to setup the network and then share the media content. Following are some of the problems with the above approach.

- There is a too much manual configuration required on the client and on the server side.
- Since the server has to be configured manually a new device joining the network cannot share its media content unless it is manually configured as a media

server. This is a considerable effort every time a device with digital content joins the network. Mobile devices like cell phones join and leave the network frequently and have the ability to generate the media content. The desired behavior is that the devices should configure themselves as media servers and share the contents without any manual configuration.

- Similarly, when a new device joins the network with capability to consume media content, it will need to be manually configured first to locate the server and then find the digital content. The desired behavior is that the device should automatically locate and consume the digital content available on the network.
- If there is no DHCP server available, there is no way of configuring the network without manual configuration of IP addresses. This includes assignment of static IP addresses which again is a manual process for each device that joins the network.

In order to address these issues, the UPnP architecture [5] was formulated to provide different plug and play services to devices on the network. Following are the features of UPnP architecture:

- **Device Connectivity** protocols define how the device joins UPnP network. These protocols define mechanisms for a device to advertise its services or invalidate advertisements. Other UPnP devices on the network understand these advertisements and configure themselves to utilize these services. It is done transparently without human intervention using the SSDP protocol [12].
- **Ad-Hoc Networking** refers to capability of UPnP device to operate in the absence of other infrastructure devices such as DHCP server for IP address management. The devices have built-in capability to select a network address from a pool and use it.

These devices are also responsible for verifying and resolving IP address conflicts as part of address selection process.

- **Zero-Configuration Networks** means that the user does not need to configure any device with any of the configuration parameters. As the device joins the UPnP network its configuration parameters are assigned to it by the network itself. The device gets an IP address automatically. It selects its role on the network without any manual intervention. It then advertises its services over the network. UPnP control point devices listen to the advertised services. The control point devices are then responsible for establishing communication between the data source and data sink devices on the network. Once the communication is established the data sink devices consume the data directly from the data source device without the intervention of the control point device that initiated the transfer.

- **Platform Independence** in UPnP terminology refers to the fact that there should be no dependency on any hardware components. Hence, devices can be developed on any hardware and software platform.

- **Media and Device Independence** refers to the fact that UPnP technology is physical layer independent. The only requirement is availability of an IP stack. The medium may include, phone lines, power lines, wired or wireless Ethernet, RF, and IEEE 1394.

The phrase UPnP stands for “Universal Plug And Play.” The word “universal” means that the implementation is not restricted to any platform. The implementation can run as an application on a personal computer or it can run on embedded devices. The word “Plug And Play” in UPnP refers to features like Ad-Hoc networking, Zero

Configuration networking and Device connectivity which allow any device implementing UPnP protocols to behave as a “Plug And Play” device when it joins the network.

The UPnP standard [5] defines many types of devices. A Control Point [5] is an UPnP device that can invoke actions on other devices. UPnP media server [6] is the device that hosts the media content for the other UPnP devices on the network. Other UPnP devices include devices capable of rendering the media content. A control point is an UPnP device (usually built into an UPnP Media Rendering device) that uses the functionality provided by the other UPnP devices on the network. It is the only device which can invoke actions exposed by other UPnP devices. By invoking actions on UPnP devices the control points establishes communication between two devices. For example the control point acts as a client device for the UPnP media server and invokes actions provided by the UPnP Media server device. It also invokes actions on media rendering device. By invoking these actions on the two devices a control point establishes a streaming session between the media server device and the media rendering device. Once this streaming session is established the control point is not involved in the transfer. Figure 2 shows the relationship.

A physical device can implement UPnP specifications [5], [7] for more than one type of logical UPnP device. For example, a personal computer can implement the UPnP Media Server specifications [6] and UPnP Media Rendering Device specifications [5], [7]. With this, the device actually exposes more than one logical device. Each logical device implements services as outlined in UPnP specifications. Further, each service implements a set of actions which can be invoked by any control point on the UPnP network. Any entity on the UPnP network can assume a role of any UPnP logical device

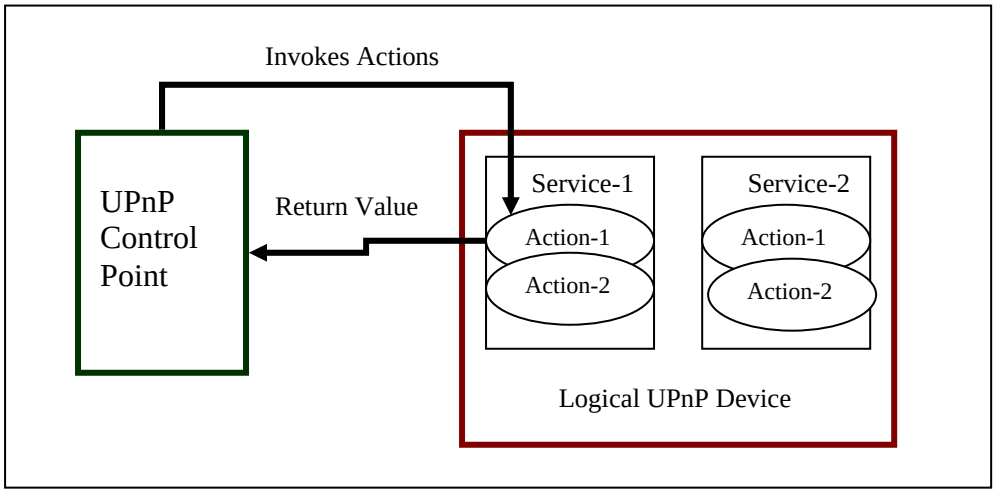


Fig. 2. Relationship between UPnP control point and logical devices.

provided it implements the required set of services and actions as defined in UPnP specifications [6], [8], [9].

UPnP uses two different protocol stacks as shown in Figure 3. In the control phase, the control point sends requests to the UPnP devices to invoke some action. This is achieved using SOAP (Simple Object Access Protocol) [10] message in the form of a HTTP request. For an event subscription, the control point uses GENA (General Event Notification Architecture) [11] over HTTP.

The Presentation block in Figure 3a refers to the presentation phase where the HTML document is exchanged between the control point and an UPnP device. An UPnP device provides access its control functions through a presentation page. The control point can then invoke these control functions as per the selections on the presentation page. The term “control function” refers to the device features that are fundamental for operation of the UPnP device. Examples of control functions of an UPnP device include volume control, power and sleep operations and firmware upgrades.

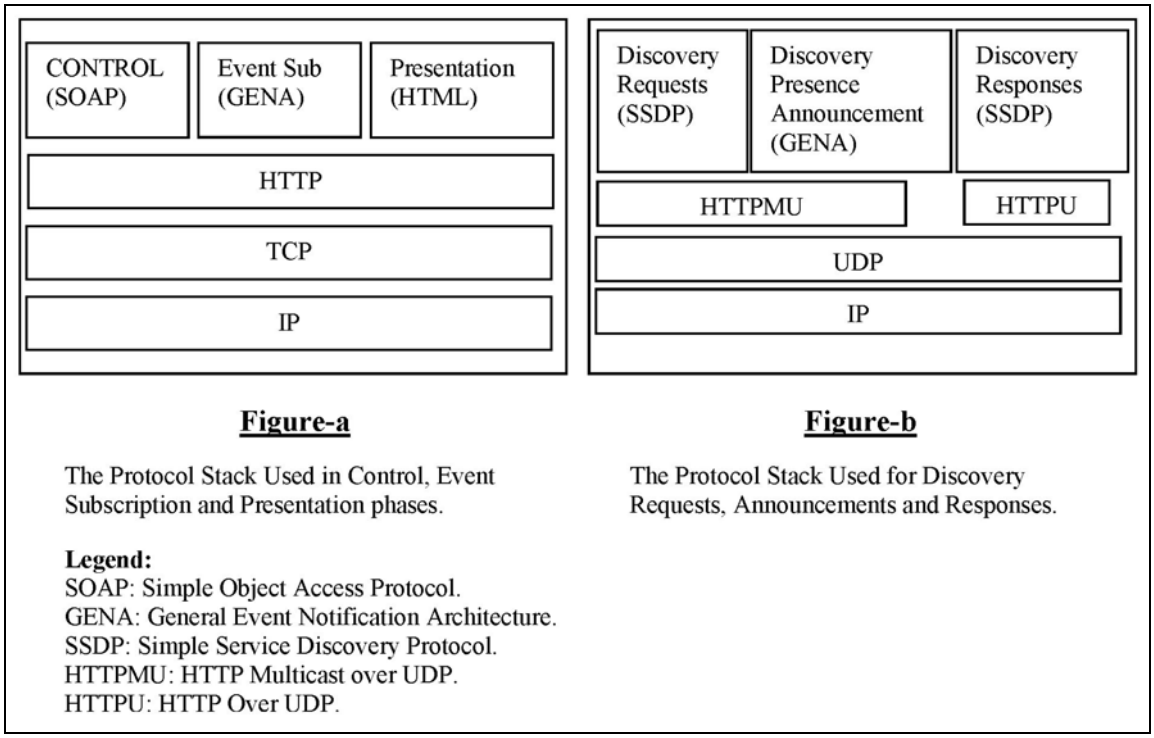


Fig. 3. UPnP protocol stacks.

For device discovery requests and responses, the UPnP uses HTTP via UDP. The HTTPMU is a simple HTTP multicast packet carried by UDP. The HTTPU refers to HTTP packet carried over UDP. The device discovery packets use SSDP (Simple Service Discovery Protocol) [12] packets over HTTPMU. GENA NOTIFY header is used for discovery presence announcement. The responses to the discovery requests are given using the SSDP [12] protocol. Figure 3 shows the protocol stacks as defined in UPnP specifications [5], [7].

The life cycle of any UPnP device starts from the time when it joins the UPnP network and goes through different phases of operation. The following section describes the phases involved in the operation of UPnP device.

Universal Plug and Play Operational Phases

The life cycle of any UPnP device can be described in five operational phases. This section describes these phases and the operation of UPnP device in each phase.

The Addressing Phase

In keeping with the zero configuration goals, it is mandatory that the device gets an IP address automatically without any manual configuration. There are two protocols available to address this requirement. These protocols are DHCP and Auto-IP.

The addressing phase can be summarized as following:

- A device tries to obtain an IP address via DHCP. If the device gets an IP address, it exits the addressing phase.
- If for some reason the DHCP was not able to assign the IP address or the DHCP server is not available, the device proceeds with Auto-IP.
- The Auto-IP protocol is designed to select and assign an IP address when it is not possible to get an IP address from the DHCP server. The protocol can be outlined as follows:
    - » Choose an arbitrary IP address temporarily.
    - » Test if the address is available.
    - » Periodically check for a DHCP server.
    - » Upon finding a DHCP server, switch to a DHCP-assigned address.

The DHCP protocol operates as follows:

- The client broadcasts a DHCP DISCOVER message on its local network segment. In the DHCPDISCOVER message, the client may optionally suggest the IP address it wants to use and the desired lease duration. This allows the client to request the

same IP address that it previously had and hence preserve the IP address across power cycles of the device.

- The server responds with a DHCPOFFER message that includes the IP address it is offering to the client along with other parameters such as the sub net mask and the default router's IP address.
- Once the client receives the DHCPOFFER message in response to its DHCPDISCOVER message, it proceeds to accept the address by sending a DHCPREQUEST message that includes a server identifier value.
- The server identifier value in DHCPREQUEST message states that the client has accepted the IP address from the indicated server. This also tells the other servers that their offers are declined.
- The selected server receives the DHCPREQUEST and acknowledges with a DHCPACK message containing the IP address for the client. If the server has already allocated the selected IP address, it sends a DHCPNAK instead, which causes the client to reinitiate the DHCP protocol.

In a case where the DHCP server is not available, the UPnP devices use another address allocation scheme known as Auto-IP. This alternative scheme is required so that the goal of zero configuration networking can be achieved. In Auto-IP, the devices choose random IP address in non route-able IP address range 169.254/16. Once the address is chosen ARP is used to verify that there is no IP address conflicts. Auto-IP does not replace DHCP but just augments it so that the UPnP devices can operate even when the IP address cannot be assigned by DHCP servers.

The Discovery Phase

All types of UPnP devices behave identical in the addressing phase. The UPnP control point devices, the UPnP media rendering devices and the UPnP media server devices go through addressing phase to acquire an IP address. After the addressing phase each UPnP device behaves in accordance to its role. The control point devices are responsible for invoking services and actions provided by other UPnP devices. The media server devices are responsible for exposing the media contents over the network. The media rendering device is responsible for rendering the contents. Since only the control point devices have the ability to invoke actions on other devices, it is the responsibility of the UPnP control point devices to establish a streaming session between a media server device and a media renderer device. Further, once the streaming session is initiated the control point device no longer participates in the actual data transfer.

In the discovery phase, the UPnP device advertises its presence and may search for other devices on the network using the Simple Service Discovery Protocol (SSDP) [12]. There are advertisements and search requests broadcasted on the UPnP network. Advertisements describe the type of services that the device provides and the search request has fields to describe the type of device being searched for.

For example, the control point devices may be interested in rendering devices to consume the digital content and media server devices to host the digital content over the UPnP network. When the control point is added to the network there are two events that occur. First, the control point sends the presence advertisements and then a search request with the type of device it is interested in finding. The UPnP device meeting the search criteria responds to the search request.

The presence announcement and the search requests are multi-cast announcements over UDP. The search response is sent only to the requester device. The advertisements and the search responses contain a URL to the device description document which describes the device in details. Once the URL to the device description document is known other devices on the UPnP network can get more details about the device using this URL. Please refer to Appendix-1 for detailed format of device description document.

Each announcement is timed. The advertisement and the search response are valid only for the specified amount of time. The device will need to renew the advertisement after the timer has expired or other devices on the network will assume that the device is no longer available. If the device leaves the network, it is required to broadcast SSDP [12] “bye-bye” message to let other devices know that its services will no longer be available.

An UPnP device will not get the old advertisements if it joins the UPnP network after the advertisements were broadcasted. To discover other devices over the network it sends out the search requests and listens for responses. From the response, it knows the URL to the device description document and will be able to enter the description phase. Figure 4 describes the advertisement and discovery mechanism.

Table 1 outlines different types of requests and responses used in the device discovery phase.

The Description Phase

UPnP devices enter this phase after the discovery phase. In this phase, the device tries to access the device description document using the URL obtained in the

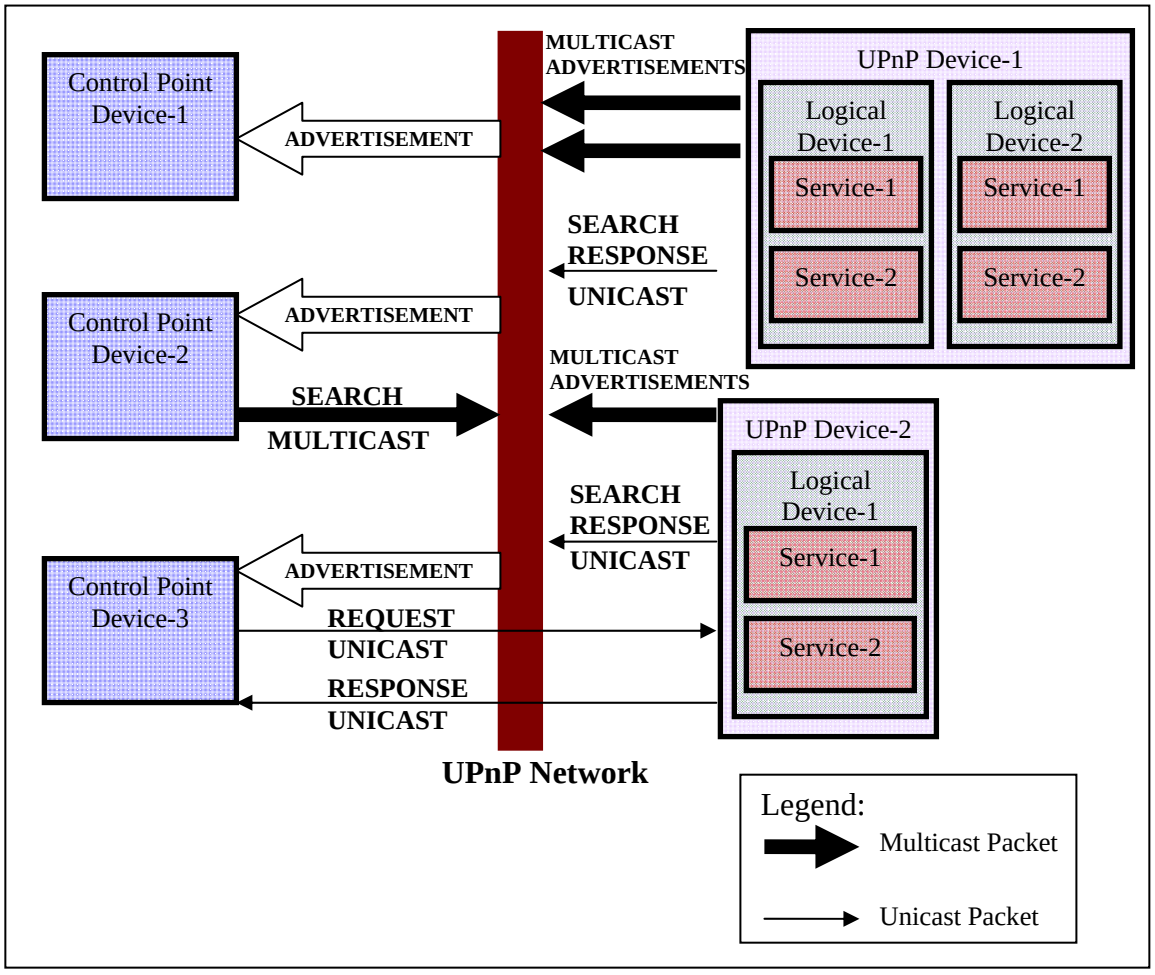


Fig. 4. UPnP discovery phase.

discovery phase. Every UPnP device has a device description document and a service description document. The formats of these documents are outlined in Appendix A and Appendix B.

The UPnP devices access the device description document using the HTTP GET method with a URL to the device description document. For example assume that a control point has received the URL “http://119.116.120.120:8081/dev/XYZ” in a discovery response. The following is the HTTP message issued to the device (with IP address 119.116.1.1 and port 8081) for retrieving the description document:

TABLE 1
MESSAGES DURING DISCOVERY PHASE

Message Type	Message Format	Comments
SSDP Discovery Request	M-SEARCH * HTTP/1.1 Host:IP:PORT Man: SSDP:DISCOVER Mx: 3 ST:urn:schemas-upnp.org:device:MediaServerDevice:1	The IP Address is the multi-cast IP address. MX header specifies the time in which the device should respond.
SSDP Discovery Response	HTIP /1.1 200 OK Location:http://192.168.0.1:4321/upnphost/udhisapi.dll?content=uuid:6859ddde-89cd-46df-bab8-1394523aec23 Ext: USN:uuid:6859ddde-89cd-46df-bab8-1394523aec23::urn:schemas-upnp-org:device:MediaServerDevice:l Server: Microsoft-Windows-NT/5.1 UPnP/l.0 UPnP-Device-Host/l.0 Cache-Control: max-age=1800 ST:urn:schemas-upnp-org:device:MediaServerDevice:l Content-Length:0	Only devices matching ST header responds. Responses should be sent only to the requester. Response includes Location field which is the URL of the device description document.
SSDP Presence Announcements	NOTIFY * HTTP/l.l Host: 239.255.255.250:1900 Cache-Control: max-age=1800 Location:http://192.168.0.1:2869/upnphost/udhisapi.dll?content=uuid:6859ddde-89cd-46df-bab8-394523aec23 Server: Microsoft-Windows-NT/5.1 UPnP/l.0 UPnP-Device- Host/l.0 NTS: ssdp:alive ST:urn:schemas-upnp-org:device:MediaServerDevice:l USN:uuid:6859ddde-89cd-46df-bab8-1394523aec23:: upnp:rootdevice	GENA NOTIFY method is used. ST is the “Search Target” and specifies how the control points can search for this device. Includes Location field which is the URL of the device description document. Cache-Control header specifies the validity time of the advertisement. As the duration of advertisements expire UPnP control points must assume that the device is no longer available on the network.
Device Unavailable Announcement	NOTIFY * HTTP/!.1 HOST: 239.255.255.250:1900 NTS: ssdp: bye-bye ST: urn: schemas-upnp-org:device:MediaServerDevice:l USN: uuid: 6859ddde-89cd-46df-bab8-1394523aec23:upnp: rootdevice	Notifies control points that device is going away. The SSDP bye-bye message should be sent corresponding to each advertisement the device broadcasted on the network. If the device experiences a fault and is not able to send out bye-bye messages, then the advertisements will time out after sometime and all other devices will assume that the device does not exist anymore.

```
GET dev/XYZ HTTP/1.1
Host: 119.116.1.1:8081
Accept-Language: ANY
(Blank line)
```

This HTTP request causes the device to return its device description in the following format:

```
HTIP/l.l 200 OK
Content-Language: <Language For Description>
Content-Length: <Length of body in bytes>
Content-Type: <text/xml>
Date: <when responded>

<?xml version=”1.0”?>
XML device description
```

The following are some attributes of a valid device or service description document [5].

- All elements and their attributes are case-sensitive.
- All values are case-insensitive except URI.
- The order of elements is not significant.
- Duplicates are not allowed for mandatory elements. They can only have exactly one instance.
- Recommended or optional elements may not occur at all or may occur only once. UPnP Devices (Control Points) must ignore any un-parse-able elements along with their sub elements and continue to function as if the element never existed.
- The device description document has an URL to service description documents so that information about the services can be retrieved. The process to retrieve the service description is identical to the process of retrieving the device description.

- The device and the service description documents are valid as long as the device remains on the network with a valid advertisement.
- If the advertisement expires, it should be renewed by issuing a new (duplicate or modified) advertisement.
- If the advertisement is changed (or renewed) the Control Points must re-read the description document as it may have changed.
- The control point cannot assume that two advertisements issued by the same device are identical.

The Control Phase

From the description phase, the Control Point Devices [5] know about other UPnP devices on the network, their services and actions exported by each service. In this phase, the control point devices use this information to invoke the exported actions with indicated parameters. Simple Object Access Protocol (SOAP) [10] is used to invoke the actions. SOAP [10] brings together XML and HTTP to provide a web based remote procedure call architecture. XML specifies the content of the message and the application layer protocol HTTP is used to exchange messages.

Each service element in device description document contains a URL where the control points can send SOAP [10] messages. The messages received by the UPnP device on this URL translates into a function call with parameters indicated in the message. The return results of the function call is then translated into another SOAP message and returned to the caller of the function. Figure 5 shows the action request SOAP messages taken from the specifications.

```
POST CONTROLURL HTTP/l.l
Host: CONTROLURL host: port
Content-Length: length of body in bytes
Content-Type: text/xml; charset=utf-8”
SOAPAction:“urn:schemas-upnp-org:service: serviceType:v#actionName”
<s:Envelope xmlns:s=’’http://schemas.xmlsoap.org/soap/ envelope/”
s:encodingStyle=’’http://schemas.xmlsoap.org/soap/ encoding/”>
<s:Body>
        <u:actionName xmlns:u=”urn:schemas-upnp-org:service: serviceType: v”>
        <argumentName>in argument value </argumentName>
        Other in arguments and their values go here, if any
        </u: actionName>
</s:Body>
</s:Envelope>
```

Fig. 5. SOAP header for invoking remote function.

The following are the points to consider from the message template in Figure 5:

- The “<actionName>” is the name of the action in the service which the caller wishes to invoke. This element must include the XML namespace of the service being called.
- If the action has arguments, each argument is provided with the name of the argument.
- The data types of the arguments are defined by the UPnP service description in an architecture independent format.

The output of the invoked function call is also returned using a SOAP message. Figure 6 shows the format of the returned SOAP message [10], [5].

In addition to function invocation, the UPnP specifications permit the control point to query the information about any state variable exposed by a particular service running on an UPnP device. The control point may directly query the value of a state variable using some predefined actions. It should also be mentioned that getting the value

```
HTIP /1.1 200 OK
Content-Length: length of body in bytes
Content-Type: text/xml; charset=”utf-8”
Date: when response was generated
Ext:
Server: OS/version UPnPl.0 product/version
<s:Envelope
Xmlns:s=‘’http://schemas.xm1soap.org/soap/envelope/”
s:encodingStyle=’’http://schemas.xmlsoap.org/soap/ encoding/”>
        <s:Body>
<u:actionNameResponse xmlns:u=”urn:schemas-upnp_org: service: serviceType: v”>
        <argumentName>output argument value</argumentName>
        Other output arguments and values, if any
</u:actionNameResponse>
</s:Body>
</s:Envelope>
```

Fig. 6. SOAP message format for function response.

of the state variable using this method is discouraged by UPnP specifications; instead event subscription mechanism is the recommended method. This method is discussed in the following section.

The Eventing Phase

Once the control point retrieves the device description document, all the necessary information about the device is retrieved. This includes the services, the actions, and the state variables that the device supports. The actions and the state variables are described in the service description document. The service description document also describes if the state variable is “evented” or not. If the variable is “evented” then the UPnP device will generate an event whenever the value of the variable changes. Figure 7 shows an event message sent by an UPnP device to subscriber at IP address 119.116.120.120 and port 8081.

```
NOTIFY http://119.116.120.120:8081/dev/events HTTP/l.l
Host: delivery host: 8081
Content-Type: text/xml
Content-Length: 80
NT: upnp:event
SID: uuid:subscription-UUID
SEQ: 10
<e:propertysetxmlns:e=”urn:schemas-upnp-org:event-1-0”>
        <e:property>
              <connectionCount>10</connectionCount>
         </e: property>
</e:propertyset>
```

Fig. 7. An example even message for evented variable “connectionCount”.

The control points can subscribe to these events using the General Event Notification Architecture (GENA) [11]. In the device description document, outlined in Appendix A, there is a URL specifically for managing the event subscription requests.

GENA [11] operates in a publisher-subscriber model. In this model, the entity that generates an event is called the publisher. Other entities on the network can subscribe to this event and these are called subscribers. When the evented variable changes its value, the publisher sends an event message to all the subscribers. This event message is of the format shown in Figure 7. The UPnP specifications [5] mandate that each event message must contain current values of all the evented variables hosted by the publisher publishing the event message. A subscriber may decide to unsubscribe at any given time. In this case it will send a “unsubscribe” request to the publisher. On receipt of this request, the publisher will remove the subscriber from its list of subscribers and stop sending the events to it.

GENA [11] introduces three HTTP headers that are used in the eventing phase. The following describes these headers in details:

- SUBSCRIBE header is used to subscribe to events.
- UNSUBSCRIBE header is used to unsubscribe to events.
- NOTIFY is used by the publisher of the event to send events to the subscribers.

When the subscriber sends a “subscribe” request to the publisher, a URL is sent in the message. This URL is called the “CALLBACK” URL and is used by the publishers to send the events to it. Service description template in Appendix B defines an attribute, “sendEvents” for every state variable. This attribute has value “yes” if the variable is evented or a “no” if the variable is not evented.

UPnP specifications define a mechanism for controlling the rate at which the events are generated. This mechanism is called event moderation. This is required because of the fact that an evented variable may change its value very frequently. If an event is generated on every value change then it would result in very high network traffic. To avoid this situation, event moderation mechanisms are provided in UPnP specifications [5].

Event moderation uses two optional integer attributes; Maximum-Rate and Minimum-Delta. The Maximum-Rate specifies the maximum reporting rate of an evented variable. The Minimum-Delta specifies by how much the state variable must change for it to be included in the event message.

The UPnP devices are physical layer independent and can operate over variety of medium including wireless mediums where the probability of packet loss is higher. Also consider that the multicast GENA event notifications uses HTTP multicast packets

over UDP thus the packet delivery is unreliable. Hence, to detect a dropped message, each event message is tagged. The tag key starts with zero and is incremented by one for each message. The subscriber keeps a running counter of messages. When a subscriber gets a message with a tag key not aligned with the running counter, the subscriber knows that some message is missing. The subscriber may decide to unsubscribe and then subscribe again. By doing this it will force the publisher to publish the values of all the evented variables. The publishers are required to send the current values of all the variables on a subscription request. Table 2 shows different messages that are exchanged in the eventing phase.

The Presentation Phase

The presentation phase is the simplest of all phases and is optional as per UPnP specifications [5], [7]. The presentation phase is required for the administrators to administrate the device and change some device settings on the fly. Devices supporting presentation phase exposes a presentation URL in device description document. The UPnP control points can use this URL to perform different administrative tasks on the device.

The URL for the presentation page is retrieved from the device description page. The <presentationURL> field in the device description is the location of this page. The presentation phase may provide a user interface to interact with the device in different ways. For example, network print servers have web pages for configuration and control allowing the administrators to set the operational parameters, print test pages, and watch the ink levels of the printer, etc.

TABLE 2
MESSAGES IN EVENTING PHASE

Message Type	Message Format	Comments
Event Subscription Message	SUBSCRIBE *publisher Path* HTTP/1.1 Host: *publisher Host:Port* Callback: *deliveryURL* NT: *upnp:event* Timeout: Seconds- subscription duration	The subscription request contains a URL where the publisher can post the events. The subscription is only valid for specified amount of time.
Event Subscription Acknowledge Message	HTIP /1.1 200 OK Date: when response was generated Server: OS/version UPnP l.0 SID: uuid:subscription-UUID Timeout: Second-actual subscription duration	The publisher uses this message to grant the subscription. There is a subscription identifier which is assigned by publisher to identify this subscription session. Once the subscription is accepted then a initial event message is sent to notify the stating values of evented state variables.
Subscription Renewal Message	SUBSCRIBE *publisher path* HTTP/l.l Host: *publisher host:publisher port* SID: *uuid:subscription UUID* Timeout:*Seconds-requested subscription duration* (blank line)	The subscription renewal message is used to renew a subscription after it has expired. The ID of the original subscription is included in the message to identify the subscription session to be renewed.
Subscription Cancellation Message	UNSUBSCRIBE *publisher path* HTTP/l.l Host: *pub1isher host:port* SID: *uuid:subscription_UUID* (blank line)	The Unsubscribe request for a particular subscription is used to stop receiving any more events from a particular device. In short the specified subscription is cancelled and the publisher does not post events to this client anymore.
Event Notification Message	NOTIFY <deliverypath> HTTP/l.l Host: delivery host:delivery port Content-Type: text/xml Content-Length: Length of body in bytes NT: upnp:event SID: uuid:subscription-UUID SEQ: event key <e:propertysetxmlns:e=”urn:schemas-upnp-org:event-1-0”> <e:property> <variableName>newValue</variableName> Other variable names and values (if any) go here … </e: property> </e:propertyset>	Subscriber should acknowledge the receipt of event messages. If the subscriber does not respond with in 30 seconds the publisher continues to send new subscription messages and does not repeat any message. All the evented variables are sent in one message.

The UPnP Audio Video (AV) Architecture

An UPnP media server device is capable of hosting entertainment content throughout the network. It can join the network, provide its services and then leave the network. Since this is a zero configuration device, all this is achieved without any manual configuration.

Figure 8 shows an UPnP network with three devices in it. The UPnP media source device is responsible for hosting the digital contents for the entire network. UPnP media server device is an example of a media source device. The media sink devices are the consumer of the content provided by the media source device. UPnP media renderer

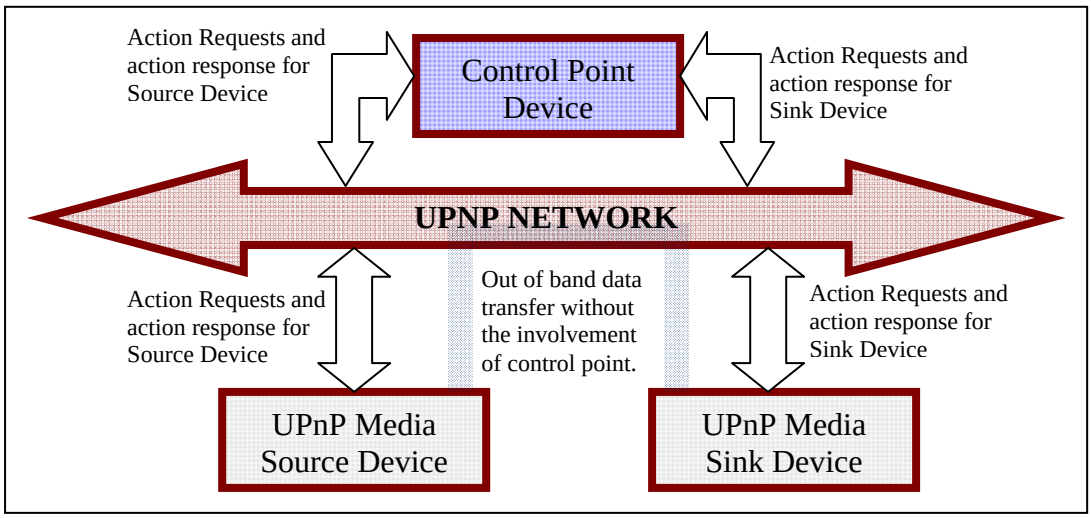


Fig. 8. Data transfer mechanism between UPnP devices.

device is an example of media sink device. The Control point devices acts as an interface between the media source and media sink device. The source and sink devices do not invoke actions on each other directly. Control Point devices are responsible for invoking

actions on each device and hence are responsible for establishing streaming sessions between a source and a sink device. Figure 8 shows the relationship.

From Figure 8, the following points can be outlined:

- Most AV scenarios involve the flow of (entertainment) content (i.e. a movie, song, picture, etc.) from one device to another.
- The control point configures both the source and the sink devices by invoking actions exported by each device.
- The devices do not interact with each other directly except during data transfer phase.
- The control point sets up the devices for data transfer and the actual transfer takes place out of band, without the involvement of control point. After the data transfer transaction is set up, the control point can be removed from the network without interrupting the transfer.
- Three entities are involved in the transfer, A UPnP Media Server (a source device), A Control Point, and a Media Renderer (a sink device).
- A single device can also act as UPnP Media Server, a Media Renderer and as a control point. For example a personal computer device can act as a Media Server and also as a Media Renderer.

The UPnP Media Server Device

UPnP media server device is a content provider device. It has access to a variety of digital content and has the ability to expose the content on the UPnP network and stream it to different clients on the network. The content can be Audio, Video, live

channels and/or still images. The UPnP media server devices [6] implement a set of UPnP services along with required methods and attributes as specified in the UPnP specifications [5], [6], [7]. Media server devices might include VCR, CD/DVD players, Jukeboxes, personal computers, camcorders, Mobile PDA, set-top box, satellite receiver, or audio tape player.

The media renderer device gets the content from the Media Server with the help of Control Points and then renders it onto a display. The examples of media renderer devices include MP3 players, electronic picture frames, network enabled music player, and digital televisions. The control point device coordinates and manages the operation of media server and media renderers. It is responsible for connecting two devices and starting the transfer. It may also provide optional user interface for functions like pause, fast forward, etc.

UPnP Media Servers implements the following services:

- Content Directory Service [CDS] [9].
- The Connection Manager Service [CMS] [8].
- The optional AV- Transport service [AVT].

Content directory service [9] is the heart of the media server device and allows the control points to enumerate the contents of the media servers. This service provides well defined actions and attributes to allow this behavior. Figure 9 lists the actions in this service.

A connection manager service is used to manage the connections. This service provides per connection operations so that the media server and the renderer have the

Name Of Action	Optional Status	Arguments		Description
		Input	Output	
GetSearchCapabilities	Required	None.	Comma separated values that can be used in search queries.	This action returns the searching capabilities that are supported by the device.
GetSortCapabilities	Required	None.	Comma separated values that can be used in search queries.	Returns the comma separated list of meta-data tags that can be used in to sort the output of the browse action.
GetSystemUpdateID	Required	None	Returns the System-Update-ID.	System-Update-ID is the required variable that changes when anything in the content directory changes. This variable is evented and is moderated at .5 Hz.
Browse	Required	Object-ID Flag Filter Starting Index Count Sort Criteria	Result Number Returned Total Matches. Update-ID.	Object-ID identifies each object in CDS to browse. Result is the DIDL document returned. Flags specify to browse the metadata or children. Index parameters specify an offset into an arbitrary list of objects. Count specifies the number of objects to be returned. Filter specifies to return only specific type of objects.
Search	Optional	Same As Browse.	Same As Browse.	
Other Optional Actions supported by CDS are				
CreateObject	Optional	Creates a new object in the specified container.		
DestroyObject	Optional	Deletes a specified object in a specified container.		
UpdateObject	Optional	Update the information about an object.		
ImportResource	Optional	The action gets a file from a remote source resource to a destination resource. The source and destination resources are specified using URI.		
ExportResource	Optional	This action transfers the file from a local source resource to a destination resource. The source and destination are specified using the URI.		
StopTransferResource	Optional	Stops the outstanding transfer.		
GetTransferProgress	Optional	Gets the progress of the transfer.		

Fig. 9. Content director service functions.

opportunity to do any house-keeping that they need when a new connection is created or an old connection is closed.

The AV- Transport service is used when the data transfer protocol needs to be negotiated. By default, the data is transferred using HTTP but if AV-Transport service is implemented, RTP (Real-time Transport Protocol) and RTSP (Real Time Streaming Protocol) protocols can also be used.

The Digital Living Network Alliance Standards

Digital Living Network Alliance (DLNA) [13], [14], [15] is a set of standards developed with the objective of making UPnP devices compatible with each other [2]. The DLNA standards are the guidelines to profile the digital content and categorize them based on parameters like encoding format, size, bit rate, and frame rate of streaming video. To get these parameters, the media server implementing the DLNA standards decode every digital item in its content directory service and then attaches a DLNA profile [14] to it. Along with the item, the profile is also exposed so that the consumer device knows these parameters for each item. The consumer device can then decide if it is interested in the content and can prepare itself by loading appropriate decoders and other software components to consume the content. Figure 10 shows the basic components of a DLNA digital media server.

Five components in Figure 8 constitute a DLNA compliant Digital Media Server. These components include:

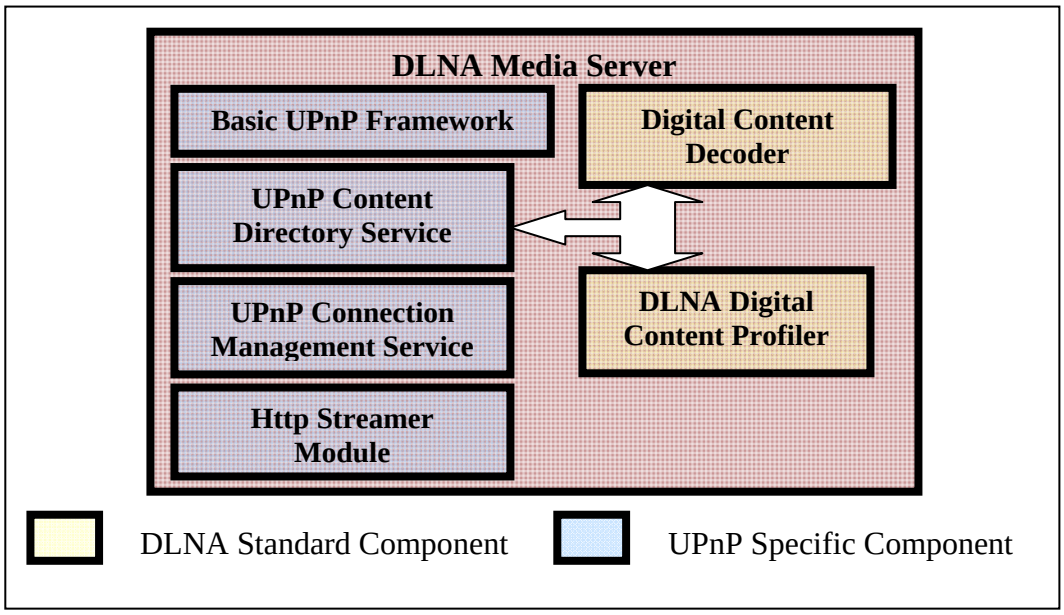


Fig. 10. Components of DLNA media server.

- The UPnP framework is the backbone of all the other components. This provides the UPnP specific functionality like the addressing, device discovery, content directory service, SOAP, eventing, etc.
- The HTTP streamer module is used for streaming of the digital content exposed by the content directory service. Once the contents are exposed using the content directory service the media server listens on a specific port for content requests. The content is then streamed to the requesting client.
- The “Digital Content Decoder” module is required because the content available through the media server’s content directory service [9] needs to be decoded so that a DLNA profile can be assigned to it.

CHAPTER III

PROBLEM BACKGROUND AND RELATED WORK

This section describes general limitations of embedded devices and the impact of these limitations on current UPnP implementations. To study these limitations and their causes, a case study of the most popular Intel® UPnP framework is also presented.

General Limitations of Embedded Devices

Embedded systems are specialized computing devices and hence differ from personal computers in terms of available computing resources. They are designed for some specific applications and they are usually less capable than general purpose computers such as PCs in terms of the computing power. The components of embedded systems are chosen depending on its application. Every embedded device provides different functionality and is usually different from other embedded devices in many aspects including processor architecture, processing power, bus architecture, and graphics architecture.

A majority of embedded devices have following limitations [16]:

1. Embedded systems run unconventional operating systems like Windows CE, flavors of Linux, and real-time operating systems. These operating systems may provide different sets of services; hence, a service available on one operating system may not be

available on other operating systems. This includes the usage of threads, file IO operations, operating system specific socket operations, timers, etc.

2. Embedded devices have limited RAM restricting the size of the executables that can be loaded.

3. They have limited processing power and may not be multi-threaded.

4. Embedded systems usually have limited permanent storage.

5. Embedded devices have limited debug capability. Majority of embedded devices have a serial port, which acts as a console input. The devices can output some debug messages on it. Some high-end embedded devices use processors that support JTAG ports. A JTAG is a port which can be used to establish communication between the processor and a debugger [16]. Using JTAG port, a debugger can perform operations like single stepping and setting breakpoints at different locations in the code. The problem is that JTAG is expensive and increases the cost of the embedded system and hence in most cases are excluded from production devices. JTAG ports also have certain security risks associated with them.

6. Embedded systems may not have all the hardware that is available in a personal computer. For example, some embedded systems do not have a real time clock and will have to depend on Network Time Protocol (NTP) to retrieve the time of the day.

When developing an UPnP framework for deployment on embedded devices, the above limitations should be considered in the basic design. These limitations dictate some design and implementation choices. For example operating system timer APIs should be used very carefully since the framework may be deployed on systems which do not have a real time clock.

Limitations of UPnP Software Development Kits for Embedded Devices

There are many UPnP Software development kits available today. Table 3 shows some of the leading UPnP software development kits.

TABLE 3
UPnP Software Development Kits

SDK Provider	Description
Allegro Software.	Allegro Software (http://www.allegrosoft.com) offers two different toolkits for developing UPnP devices. Both toolkits are ANSI-C based implementations and delivered as source code. The basic version of tool kit implements discovery and presentation. The advanced tool kit implements control and event mechanisms. It has a hardware abstraction layer that allows toolkits to run on a wide variety of embedded platforms.
Atinav Incorporated	Atinav's Ave-Link technology supports Java implementations of UPnP devices and control points. The Ave-Link SDK is divided into device and control point libraries for maximum flexibility, and supports version 1.0 of the UPnP Device Architecture Specification. For more information about the SDK please refer to UPnP Partner SDKs (http://upnp.org/sdcps-and-certification/resources/sdks/).
Lantronix	Lantronix's UPnP Early Adopters Kit (EAK) provides addressing and discovery capability in a small C-source-based library suitable for integrating into anyone of Lantronix's embedded platforms. Control point functionality is not supported.
Intel	The Intel® SDK for UPnP Devices for Linux is a C-based implementation that supports all phases of UPnP device development. The SDK has been released with full source code under the Berkeley Software Distribution (BSD) license. Intel® has made available a version of the Intel® SDK for Pocket-PC devices.
Others	There are other UPnP framework implementations available. Please refer to UPnP Partner SDKs (http://upnp.org/sdcps-and-certification/resources/sdks/) for their description.

Even though these SDKs are available, many companies developing DLNA compliant devices chose to create their own UPnP framework implementation. This is clearly evident from the fact that the majority of DLNA devices available today in the market do not use any of the standard UPnP software development kit. For example, companies like Qualcomm, Mediatek, Broadcom Corporation and many others use their own implementations of UPnP framework. This implies that none of these UPnP framework implementations are globally adapted.

In order to generally understand some of the limitations of any UPnP framework with respect to embedded devices, a case study for Intel® UPnP framework was conducted. The Intel® UPnP framework was selected for this study because it is the most popular framework and is also open source. It forms the basis of many other frameworks and many companies have modified this framework to create their own UPnP implementation suitable for their hardware.

The Intel UPnP Framework: A Case Study

Intel® is one of the earliest providers of an UPnP framework. The framework became the most popular choice because of its stability and lack of competition. This framework then became open source and the source code is being distributed under Berkeley Software Distribution license. This implies that it is royalty free, and the license does not place any restrictions on any derivative of this SDK.

Figure 11 shows the components of Intel UPnP software development kit. The information presented in Table 6 evaluates major parts of the SDK from the perspective of deployment on embedded systems. Information for this case study is taken from

references [18], [19]. Figure 12 summarizes the main aspects of the SDK when deployed on embedded systems.

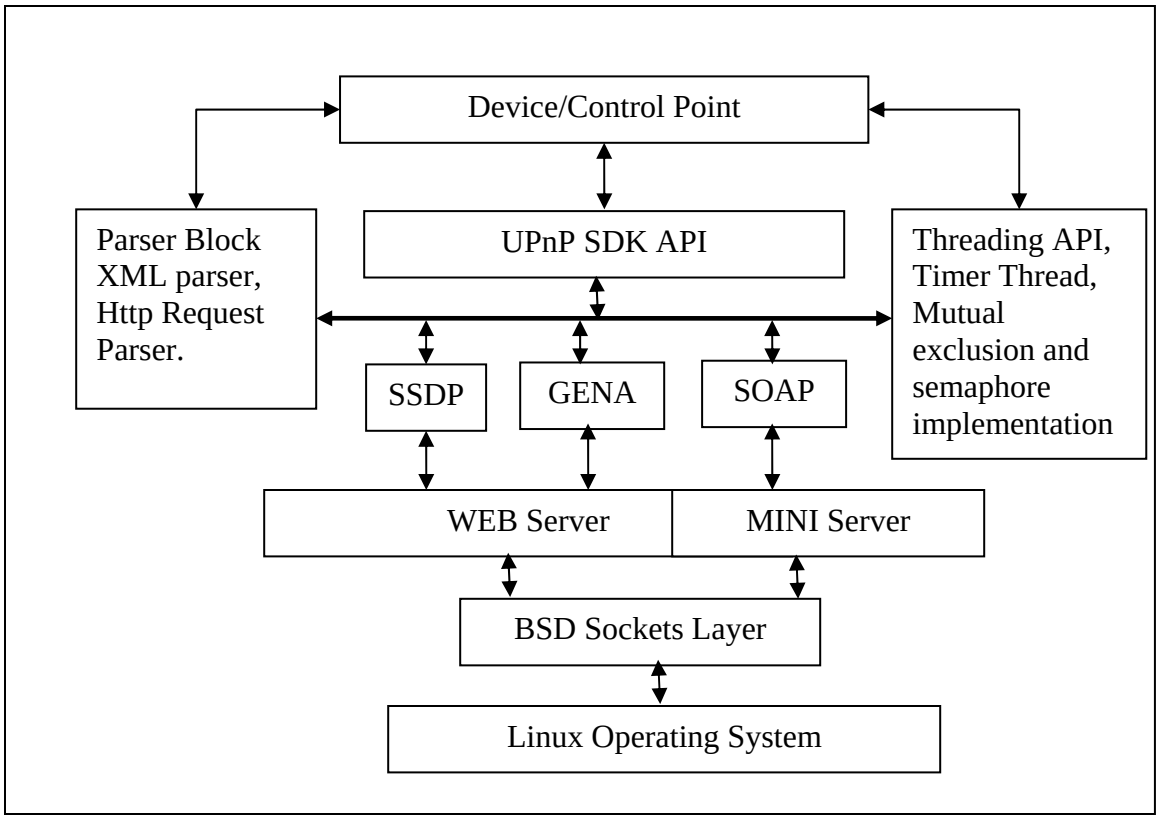


Fig. 11. Intel UPnP SDK architecture.

Summary of Limitations of UPnP Frameworks

From the above discussion, the following major limitations of generic UPnP implementations can be highlighted when considering them for embedded devices:

1. Majority of UPnP frameworks tend to be feature complete for marketability. They implement all of the features stated in the UPnP specifications [7], [5]. For embedded devices, many of the features are not needed. An example would be the AV transport service which is optional but most of the UPnP SDKs have support for it.

<table>
<tr><th>Intel® UPnP Framework Property</th><th>Comments from Embedded System Perspective</th></tr>
<tr><td>Language of Choice: C</td><td>C is a highly portable language supported on wide range of embedded systems. The choice of this language makes the Intel framework feasible for deployment on embedded devices.</td></tr>
<tr><td>Supported Operating System: Linux</td><td>Though there are many embedded systems running Linux, there are many embedded devices which do not run Linux. New class of mobile devices is currently flooding the market with Google’s Android operating systems. The Intel SDK cannot be used on any other platform except for Linux.</td></tr>
<tr><td colspan="2">Modules Analysis
<table>
<tr><th>Module</th><th>Description</th><th>Comments</th></tr>
<tr><td>XML Parser</td><td>This is a heavy duty XML parser module which also supports level DOM and C-DOM representation of XML.</td><td>Embedded systems do not need such elaborate XML parsing capabilities to develop DLNA media servers. Only specific types of documents need to be parsed and this functionality can be implemented with very few lines of code.</td></tr>
<tr><td>HTTP Parser</td><td>Parses HTTP request and generate appropriate responses.</td><td>This is a heavy duty HTTP parser which is usually more than needed for embedded devices. It implements many http request and response headers. A media server does not need all the headers hence this module can be optimized to a great extent.</td></tr>
<tr><td>BSD Socket Layer</td><td>This is the sockets layers used for underlying communication with other devices.</td><td>This module is highly dependent on operating systems providing BSD socket implementation. Many modern embedded systems provide fast and light weight replacements for sockets. Since this module is the core of the framework it is difficult to replace this implementation with a custom implementation.</td></tr>
<tr><td>SOAP, GENA and SSDP</td><td>These are standard implementations of the corresponding protocols.</td><td>This is very thin implementation and suitable for embedded devices.</td></tr>
<tr><td>Threading Library</td><td>Provide custom multi-threading capabilities to devices and implements scheduling of different request and responses.</td><td>Many embedded devices which may consume and host the digital entertainment content are inherently single threaded. They do not have enough processing power to provide multi threading capabilities. This implementation is inherently multi-threaded and hence may be unusable on some embedded devices without major rework.</td></tr>
</table></td></tr>
</table>

Fig. 12. Case study of Intel UPnP software development kit.

Another example is the media server available in Intel UPnP SDK [18] which is a high-end server that provides extensive functionality usually not required by embedded devices. Since they include some of the non-essential features, the size of the SDK itself is large which may be unacceptable in an embedded environment.

2. For embedded systems, debug ability becomes an issue for these SDKs since many embedded systems do not provide powerful debuggers, which are available in integrated development environments on a personal computer. There are hardware limitations for enabling debugging on embedded systems. For example, the embedded device may use processors which do not support JTAG and hence it will not be possible to perform *"live"* debugging by putting breakpoints and single stepping. As a result it is a challenge to debug software directly over embedded systems.

3. Generic UPnP frameworks are not optimized to use available hardware features of different embedded devices. For example, most modern set top boxes have one or more hardware decoders to decode the incoming video stream. DLNA media servers need video and image decoders to decode the media contents. This is necessary so that certain parameters can be obtained which are required to assign proper DLNA profile [14] to the media content. Standard frameworks do not expose interfaces to utilize the hardware video decoder. Another example would be the fast socket layer implementation to improve the network performance.

4. Some of the implementations use object-oriented languages like Java. These sometimes are not the language of choice for embedded systems. Java is an interpreted language and uses a Java Virtual Machine to run the java byte code. This introduces performance problems which may not be acceptable in embedded environments. Object

oriented languages like C++ are recently finding their way in to embedded systems. Traditional choice of language for embedded systems is the C language.

5. The UPnP implementations use operating system APIs and hence are tied to particular operating systems. This is a problem since different embedded devices may use different operating systems. The use of operating system APIs should be abstracted in such a way that only parts of the framework is actually dependent on the operating system. By rewriting the operating system specific files, the rest of the framework could be deployed on the targeted operating system. Also, consider that some operating system APIs might not function correctly on some embedded systems. For example, the timer APIs do not function correctly on embedded systems without the real time clock.

6. Some SDKs are not maintained by their provider and the support levels are very limited through the software development phases. This causes unnecessary delays in the software development life cycle.

7. The UPnP implementations which are candidates for embedded systems are not always free of costs. There are royalties associated with them which increases the total cost of the product in a highly competitive market.

CHAPTER IV

PROJECT DESCRIPTION AND DEVELOPMENT/TESTING METHODOLOGY

Problem Statement

There are many UPnP implementations available to cater to different requirements. These implementations are generic in nature and as a result they are feature rich. Many of these are not designed with embedded systems in mind. At the same time, some of these are actually designed for embedded systems. The problems with these UPnP implementations designed for embedded systems are that they are not cost effective, not flexible and their functionality cannot be over written without extensive work. If they are open source, there are royalties associated with them. If they are not open source they are not cost effective and they cannot be modified and the fact makes it difficult to use any custom hardware features which may be available on a particular embedded system. They often contain features which are not appropriate on some of the embedded devices since embedded devices like cell phones do not have the computing power needed to support these features. The embedded devices need thin, efficient, modularized, operating system independent implementation that can be debugged effectively.

This project was an effort to create UPnP implementation with consideration to above stated problems.

Project Outline

This project designs and implements the basic UPnP standards [5] specifically for embedded devices with limited RAM, storage and computing power. The implementation handles the problems stated in Chapter III (Summary). These problems are used to formulate guidelines and requirements for this project which are outlined as follows:

1. Since the embedded systems have limited RAM and processing power, it is required that the UPnP implementation should be lightweight. It should include only the features that are mandatory and are fundamental to the operation of media server devices.

2. Considering that some embedded devices have hardware components which can be used to enhance the performance of media servers, the SDK should provide mechanisms to overwrite default implementations easily so that the developer of media server device can use the hardware components easily. For example many set-top boxes today provide hardware decoders. In this case, the UPnP SDK should provide :

    a. A software decoder, which is the default decoder that the SDK will use.

    b. The SDK should expose simple APIs for using the software decoder.

    c. If a hardware decoder module is available then user of the SDK will need to expose the same APIs as provided by the software decoder. This way the decoder can be changed without impacting other parts of the implementation.

3. Proper abstractions should be provided to achieve platform independence so that the framework can be deployed on variety of operating systems.

4. Once the SDK works on a variety of operating systems, the developers using this SDK will be able to debug and develop UPnP devices on some familiar environment like Microsoft Visual Studio 2008. This greatly enhances the debugging ability of the system under development, a great benefit to the developers. The developed binaries then can be recompiled for embedded device and deployed on the target.

5. To enhance the usability of the framework, the interfaces provided by the framework should be simple and easy to use.

This project was an attempt to develop the SDK with guidelines stated above. The development of the SDK was done in six different phases. Each phase targeting a specific functionality. Testing was performed as a recurring activity during each development phase. The following section outlines the details of every development phase.

Project Development Phases

The project was divided in to six development phases with each phase targeting a specific functionality. Early phases focused on design and development of basic modules which would be required by the rest of the framework. Testing was performed in a recurring manner during every phase.

All the development was done on Microsoft Visual Studio 2008 development environment. At the completion of each phase, a porting effort was undertaken to validate

that the code can be compiled and executed in Linux environment. This was done so that the cross platform portability of the SDK can be verified.

This section describes each development phase in details and what was achieved in each phase.

Phase One: Basic Socket and Connection Management Phase

This phase targeted on developing basic socket interface. This socket interface is used for managing all the UPnP functions. The interface provides convenient APIs for the rest of UPnP framework to use. There are three types of sockets that are required.

- A TCP socket to receive the packets that are specifically addressed to this UPnP device. This socket is used during description, control and eventing phases. This socket is used throughout the life time of the media server device.
- A UDP socket is needed for handling the multicast packets. The packets received through this socket include SSDP [12] M-SEARCH request packets and the GENA [11] NOTIFY packets. The GENA [11] notify is a multicast packet because the notification is intended for all the subscribers.
- UPnP devices can use any port number that is available for it. Since the URL containing the port number for each device and service is communicated to other devices on UPnP network the scheme automatically becomes independent of port number.
- The third types of sockets in the framework are the sockets that are used only for sending, not for listening. An example is the socket used to transmit the multicast SSDP [12] advertisements.

As a part of this phase, simple APIs were created to create these types of sockets. This simplifies the socket implementation from rest of the framework and makes the SDK independent of the operating system socket APIs.

The listener sockets are abstracted in such a way that a callback function is associated with a socket. When there is data available on the socket, the callback function is called to process the data.

Phase Two: The Device Discovery Phase

This phase emphasizes on Simple Service Discovery Protocol (SSDP) [12] to handle device discovery. In device-discovery phase, XML documents are exchanged to parse the SSDP requests and advertise the services that a device exposes. A basic XML module was developed in order to parse and generate the XML documents.

In order to test and debug the implemented SSDP [12] protocol, a basic media server device was written which advertises its presence using the available functionality. This phase used different open source tools [20] for testing. Specifically UPnP AV Media Controller [20] tool was used to validate advertisements and search responses. The implementation was tested using two or more instances of UPnP AV Media Controller. This way the basic connection management and sockets were verified.

Once the basic implementation was in place, the Device Validation Tool [20] was used to fully validate the available functionality. The device validation tool can invoke different functions with different parameters. This was done to make sure that the Media Server can handle multiple requests simultaneously from different clients.

Phase Three: Control Phase

This was the longest development phase of all the phases and the focus was on the following:

- Basic infrastructure for simple object access protocol (SOAP) [10] was developed. The SOAP protocol deals with remote procedure calls and the parameters are passed as comma separated string values. The basic SOAP functionality was developed and tested outside the framework using stand alone application. Each SOAP API was invoked individually with different parameters.
- Content Directory Service (CDS) [9], which is the part of the Media Server, was implemented using the developed framework. The content directory service does the following:
  - » Expose the contents of a directory to other UPnP devices. This service is passed the path of the directory which has the contents to be exposed.
  - » A URI is assigned to digital content which is being exposed by the CDS.
  - » The media content will be categorized in to three categories namely video, image and the audio files.
  - » Later the CDS is enhanced to add the DLNA functionality so that the DLNA profiles [14] can be assigned to the contents.
- The content directory service is tested using open source tools for UPnP [20]. Among these tools the following were used:
  - » Device Spy Tool [20] for invoking remote procedures with different parameters.

» AV Media controller to browse and access the hosted contents through content directory service.

» Device Validation Tool [20] to validate the content directory service implementation.

- A basic Connection Management Service [8] was implemented since it is critical for the function of UPnP Media Server Device. Using the Device Spy Tool [20] operation of different functions exported by the Content Directory Service (CDS) and the connection management service was verified.

- The Content Directory Service contains three features that can provide enhancements to the basic service. These features are optional as per the specifications and hence are not implemented as part of core SDK. The following are the features that are omitted from the implementation.

» First is the ability to search the CDS for some media content. This is a resource intensive task since the search parameter should be parsed, the whole content directory may need to be scanned to create list of the items satisfying the search criteria and then the DIDL response is generated. This may also dictate use of data structures like trees, hash or lookup tables. Since search is optional feature and is resource intensive, it is not included in the core SDK.

» The second feature omitted from the implementation is the ability to sort the media content. The control point devices may request to sort the results of the "Browse" request and provide some criteria (such as artist name for mp3 files) for sorting. To implement sorting effectively there are two general approaches that can be adapted. In the first approach, the DIDL document being generated could be

adjusted every time a new item is added to the document. In this approach, major processing and memory to memory data transfers would be needed to generate a correct DIDL document. In another approach a two pass processing may be adapted where in the first pass a sorted list of media items is created and then the DIDL document is generated. The second approach is much more feasible but still requires two pass processing. Since “sorting” is optional, it is not implemented in the core SDK implementation.

» Third feature include the ability to dynamically add or remove the digital media items from the content directory. Once the CDS is changed the UPnP specification states that a CDS change notification should be sent to all the subscribers. On the receipt of this event a sequence of browse operations will occur to remunerate the content directory service. This is a optional feature and is not implemented as part of core SDK.

Phase Four: Subscription and Eventing Phase

UPnP service specifications [8], [9], [6] specify that some variables may be evented. The clients of the media server may subscribe to these events. It is the responsibility of the media server device to send the events to all the subscribers. They are notified every time the value of any of these evented variables change. The “Eventing and Subscription” use General Event Notification Architecture (GENA) [11]. This phase emphasizes on the development of GENA. The GENA implementation was tested using Open Source AV Media Controller and AV Media renderer tools [20]. These tools enable us to subscribe and unsubscribe to events and report event updates when they receive them.

Phase Five: Adding Data Streaming and DLNA Functionality

After the completion of previous phase, the framework had the capability to advertise itself, respond to search requests, and to expose the content on the UPnP network. The two important missing pieces included the ability of the media server to stream the contents and the DLNA functionality for profiling the contents being exposed on the network.

A HTTP streaming module was developed to stream the media contents. When the contents are exposed to the UPnP network, a URL is associated with each file being exposed. The HTTP streamer listens to the port specified in the URL and parses the request on that port. The requested file from the content directory is transferred in the response to HTTP GET method.

In order to prove that the framework can be used to develop a DLNA compliant media server, the DLNA functionality was added to the content directory service using open source components. Open source implementation of DLNA specifications [21] along with industry accepted open source FFMPEG decoders [22] were used for parsing the digital content and assigning a DLNA profiles to them. Only the content directory service was changed to add the DLNA compatibility.

It should be mentioned that there are many features listed in UPnP specifications [5] that are optional. In order to keep the size of the implementation suitable for deployment on embedded devices certain optional features are not implemented. Additionally, certain design choices are made to achieve operating system independence and efficiency. These design and implementation choices are the major

differentiators for this SDK implementation. For details, please refer to the “Competitive Analysis “section of Chapter VI.

Phase Six: The Final Testing Phase

The final testing included testing the DLNA media server on Microsoft Windows and Linux operating systems. This was a black box testing where all the functionality of the final DLNA media server was tested in an actual UPnP network environment. Microsoft® Windows Media Player® on Windows-7® has all the functionality needed to act as an UPnP DLNA client. This was used to test the implemented DLNA media server and the results are as follows:

- The DLNA media server was able to automatically advertise its presence to other devices on the UPnP network. Windows Media Player client was able to discover the implemented DLNA Media Server automatically.
- The DLNA media server was able to expose the digital contents and other devices on the UPnP network were able to browse the media exposed by the media server.
- The media server was able to handle the stream request from clients for digital video, digital audio and the image files.
- The Media server was tested on two different operating systems. First the server was tested on Windows Operating System on which it was developed and then on Linux operating system. This was done so that the cross platform compatibility can be verified. The behavior of the media server was identical on both the operating systems.
- Apart from the above testing the media server was also tested on an embedded system environment. A Broadcom Corporation’s set top box device was used for testing.

The Development and Testing Environments

Microsoft Visual Studio 2008 was the environment of choice for the development of SDK. This choice was made because the environment is very powerful with an integrated debugger and many other helpful tools. Using the Visual Studio, the code can be managed very nicely. With each development milestones achieved, the SDK was deployed on Linux to verify the cross platform compatibility. UBUNTU Linux 10.04 was used as an additional debug and development environment. The SDK was compiled and tested under this environment.

It should be mentioned that though the implementation is operating system independent, there were cases where a bug appeared in one operating system and did not appear on the other operating system. This was because the behavior of the operating system specific APIs in these two environments are different and so the assumptions carried over from one environment did not hold true under second environment.

After the majority of the SDK and the media server functionality were available, the implementation was tested on an actual embedded device. Broadcom Corporation's set top box device was used for this purpose. This hardware uses a MIPS based processor and is completely proprietary. This testing was required so that the portability of the implementation can be verified. It should be mentioned that the set-top box device used is readily available to developers and can be obtained on request from Broadcom Corporation.

CHAPTER V

DESIGN OVERVIEW

Previous sections described the goals, the development environment and the distribution of development effort across the different development phases. This section discusses the design of the framework along with some important data structures which are fundamental to the implementation. This section also discusses other components developed as part of the project to validate the framework.

DLNA Media Server Components

Figure 13 describes the components that are used in the project. It shows the components that constitute the UPnP framework. The implementation can be divided into three parts:

- The UPnP framework code that exposes a set of APIs. These APIs provide different UPnP devices with mandatory UPnP functionality.
- The Media server device was developed to test the UPnP framework. This is the device that uses the APIs exported by the framework. Two mandatory services are implemented. These are the Connection Management Service [8] and the Content Directory Service [9].
- Figure 11 also shows how the content directory service interfaces with open source DLNA implementation. The DLNA implementation is in the form of a library and

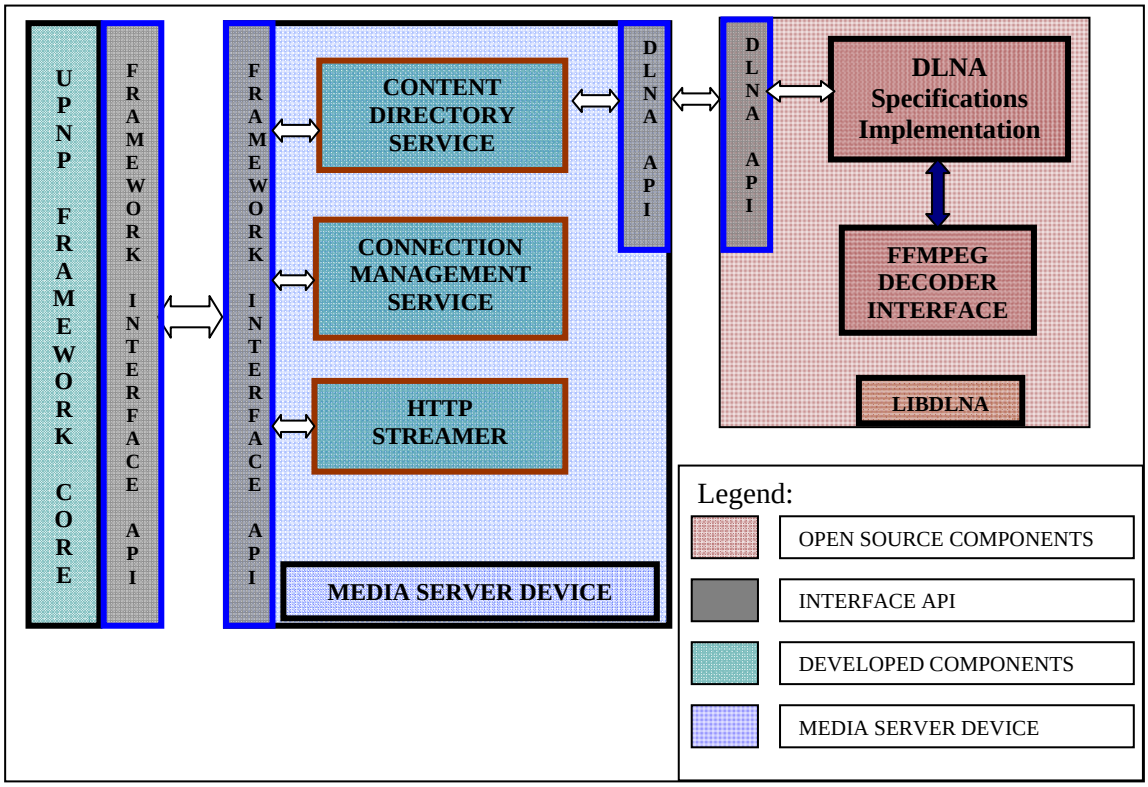


Fig. 13. Components used in developing UPnP framework.

is responsible for assigning DLNA profiles to contents that are being exposed by the content directory service. The DLNA library uses open source FFMPEG decoders to parse the contents. It should also be mentioned that none of the open source code has been modified, changed or developed in this block since the major focus of this block is to prove that the developed framework implementation can be used to develop a DLNA media server.

Though the focus of the project was to develop an UPnP framework it was imperative to develop a device using the framework so that the framework could be verified. Further open source DLNA components were used to make the media server a

DLNA compatible device. The following sections describe the design of all three components outlined in Figure 11.

Design of UPnP Framework Core

The framework provides mechanisms and data structures to register a device and its services with the framework. Once the service is registered the framework is responsible for the following operations.

- It handles the search and discovery of the device. The advertisements, device description and service description documents are generated automatically on behalf of the registered device.
- It handles the GENA subscriptions for the device. In case the evented variable changes its state the framework is notified using an API. After this the framework will send the GENA NOTIFY messages to all the subscribers of the event.
- The SOAP envelope parsing is implemented in the framework. On receipt of a SOAP envelope the framework parses it and invokes respective functions. The response is collected from the called function and then the SOAP response message is delivered to the caller.

Registering Device and Services with Framework

When the media server device is started, it registers itself and its services with the UPnP framework. The registration is required so that the framework can get the necessary information to generate the device description document and broadcast advertisements on the UPnP network. The registration process involves filling a data

structure and calling framework API. Figure 14 shows the data structure that a device needs to fill in order to register itself with the framework.

```
BUPnPDeviceInfo
char* friendlyName;
char* manufacturer;
char* manufacturerUrl;
char* modelDescription;
char* modelName;
char* modelNumber;
char* modelUrl;
char* serialNumber;
char* udn;
char* upc;
BUPnPIconInfo **iconList;
const char* presentationUrl;
```

Fig. 14. Data structure used to register UPnP device.

In the above data structure, a presentation URL is specified in case the device supports presentation phase. The presentation of the device is optional and hence this field is optional. Every UPnP device supports a set of services. In order to generate the device description document, the framework needs to know about the services, its methods, and state variables if any. Figure 15 shows three data structures. The "BUPnPServiceInfo" represents a service. Table 4 describes the field of "BUPnPServiceInfo" data structure.

The state variable exposed by the service is represented by "BUPnPStateVariableInfo" structure. In this structure the attributes field is the field which specifies different aspects of the state variable including eventing and moderation of the state variable.

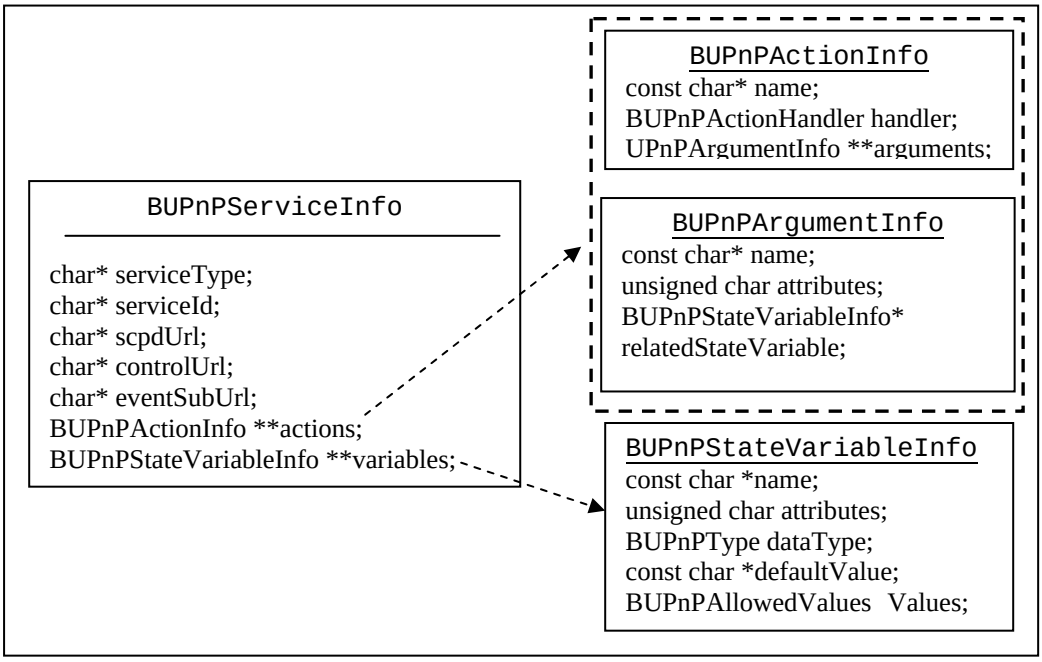


Fig. 15. Data structures used for registering services.

Once the above data structures are initialized with correct information certain framework APIs need to be invoked for completely registering the device. Figure 16 shows the APIs that are required to register a device and its services.

Both the connection management and the content directory services use the above data structures and APIs to register themselves with the UPnP framework. Once the device and services are registered, the framework has all the information that it needs about the device and its services and is ready to go in to the discovery phase by sending the advertisements over the network. The discovery phase is started when the API for starting the framework is invoked. Figure 17 shows the events that occur in the framework when all the devices and services are registered and the framework is started.

The framework is started using the “BUPnP_Start” API exported by the framework. When the framework is started a new thread is created. This is done so that

TABLE 4
DESCRIPTION OF BUPNPSERVICEINFO DATA STRUCTURE

Field Name	Data Type	Description
serviceType	Byte Pointer	This is a standard field and identifies the type of the service. The value of this field is described in UPnP Specifications [5].
ServiceID	Byte Pointer	This is identifier for the service and the value is defined in the UPnP specifications.
scpdUrl	Byte Pointer	Each service provides a service description document. This variable is a string and is appended to the URL of the service description document. The framework uses this string to identify the service from the URL. The framework automatically responds to HTTP GET requests on the SCPD URL and return the service description document as a part of the response.
controlUrl	Byte Pointer	Each service has a control URL on which the control points sends the soap envelope to invoke a function. The framework uses this string value and appends to a common URL string. This helps the framework to identify the service on which the function should be invoked.
eventSubUrl	Byte Pointer	Any service which supports evented variables provides an event-sub URL on which the UPnP devices can send GENA subscriptions. This string value is appended to a common event URL and helps the framework to identify the service for which the GENA request is targeted. The framework handles all the GENA subscriptions automatically.
actions	BUPnPActionInfo	This is a abstract data type for representing a action implemented in any service.
variables	BUPnPStateVariableInfo	This is a data structure to represent a state variable in any service.

the framework can wait and listen to socket for connections requests. In this thread there are two types of sockets created, one a TCP socket and another is a UDP socket. The

Framework API			Description
API Function	**Parameters**		
BUPnP_Initialize	**IN**	BUPnPOpenSettings	This does the basic initialization of the framework like allocating the memory for context information. The input to the function is a BUPnPOpenSettings structure which describes various parameters like port numbers, IP address etc.
	OUT	BUPnPError	
BUPnP_RegisterDevice	**IN**	BUPnPDeviceInfo	This function registers the device with the framework. A device is represented by a data structure which has all the device specific parameters.
	OUT	BUPnPDeviceHandle	
BUPnPDevice_SetDeviceType	**IN**	BUPnPDeviceHandle	Sets the device type of the device. The device type is defined by UPnP specifications [4], [7], [5] and is a fixed string. This is needed so that the framework can respond to search requests.
		const char*	
	OUT	None.	
BUPnPDevice_AddService	**IN**	BUPnPDeviceHandle	Each device provides one or more services. For example the Media server device provides two mandatory services to expose the media server contents and a connection management service to manage multiple clients. This API registers the service of a device to the framework. The service is registered with the framework and the description document is updated by adding these service parameters to it.
		BUPnPServiceInfo	
	OUT	None.	
BUPnP_Start	**IN**	None.	This function starts the framework after all the devices and their services are registered with the framework. After call to this function the SSDP advertisements are sent out and the device is ready to respond to search queries. The control points on the UPnP network can invoke functions on specific device using SOAP.
	OUT	BUPnPError	

Fig. 16. Functions for registering media server device.

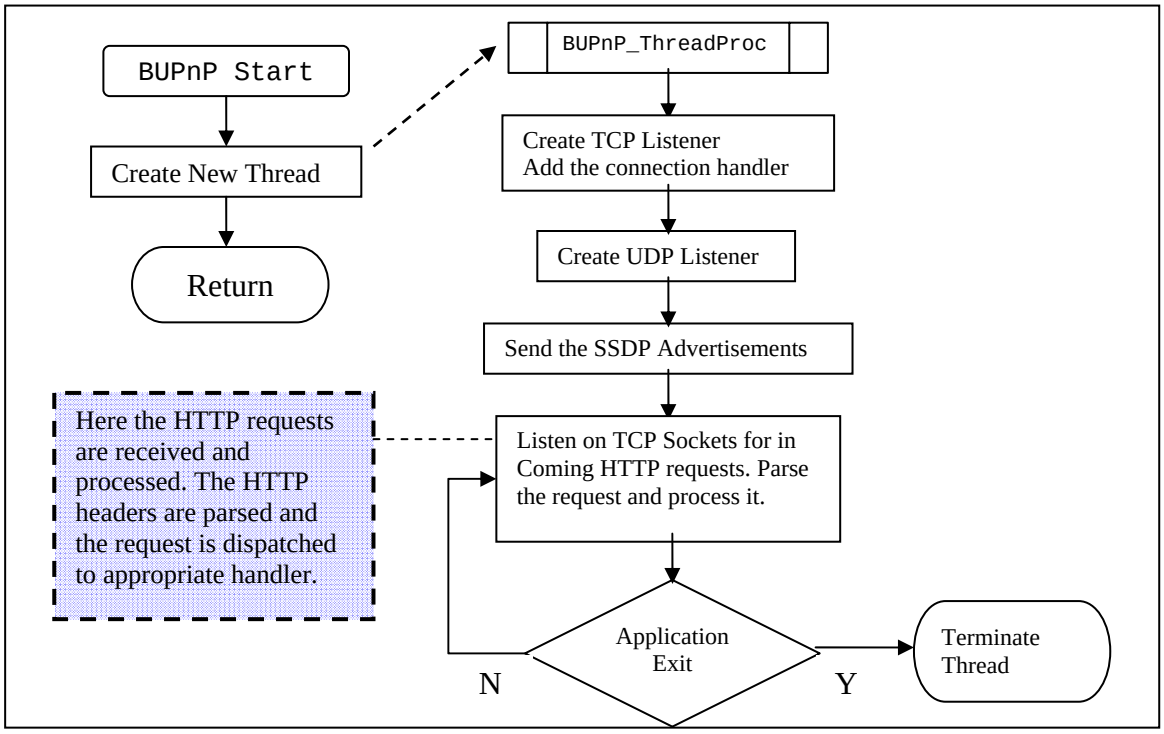


Fig. 17. Framework initialization.

TCP listener socket is created to handle the packets that are specifically addressed for this device. For example, this socket is used during the Description, Control and Eventing phases.

There are five different types of HTTP packets handled by the TCP socket. On the receipt of any header the packet is forwarded to its appropriate handler. Table 5 illustrates the HTTP methods and their handlers in the framework.

The UDP socket is created to receive multicast packets and to process them. These packets include the HTTP packets with M-SEARCH header for which the device needs to respond with device description document.

TABLE 5
FRAMEWORK MESSAGE HANDLERS

HTTP Method	UPnP Packet Handler	Comments
GET	Http_ProcessGet(…)	This is the request from a control points to get the device description document.
POST	Soap_ProcessPost(…)	This is the request to invoke a function call.
SUBSCRIBE	Gena_ProcessSubscribe(…)	This is a request to subscribe to evented variables.
UNSUBSCRIBE	Gena_ProcessUnSubscribe(…)	This is a request to unsubscribe the previous subscription.
NOTIFY	Gena_ProcessNotify(…)	This message notifies a change in an evented variable.

The SSDP [12] advertisements are sent separately and a new socket is created for sending the advertisements since the life of the socket is very short. The socket is destroyed after the advertisements are broadcasted.

Socket and Connection Abstraction

The sockets are abstracted in such a way that every socket can have a callback associated with it. The framework listens to the socket and on receipt of data the callback function is called. The callback and its context are abstracted in a data structure as described in Figure 18.

If the purpose of the socket is to listen for data then the callback function should be associated with the socket. When there is data available on the socket the call back function will be called to process the data. A callback function can be associated

```
                BUPnPConnection

SOCKET                          socket;
uint32_t                        timeout;
time_t                          expireTime;
BUPnPReceiveCallback            callback;
HttpContext                     context;
void*                           args;
BLST_ENTRY(BUPnPConnection)  link;
```

Fig. 18. Data structure for abstracting the socket handler.

with the socket using the “BUPnP_AddConnection” API exposed by the framework. To manage the connections a linked list of connections is used. When the data is available on a socket the linked list is walked through and the appropriate call back is invoked to process the data.

Timers and Callback Routines

The timers are used in the framework to renew the advertisements by sending the SSDP [12] “alive” messages. The timer is created with a callback handler to renew the advertisements. The timer is a free running timer thread and every time the timer expires the call back is called. Conventional timer APIs are not used instead operating system events are used to implement the timers. This is a design choice because uniform wait schemes can be implemented using events rather than timers. Also the timers are dependent on real time clocks which may not be available on some embedded devices. The wait period for events is calculated using more accurate schemes like microprocessor’s performance counters which use the microprocessor’s clock instead of real time clock. The following data structure (Figure 19) represents a timer instance in the implementation.

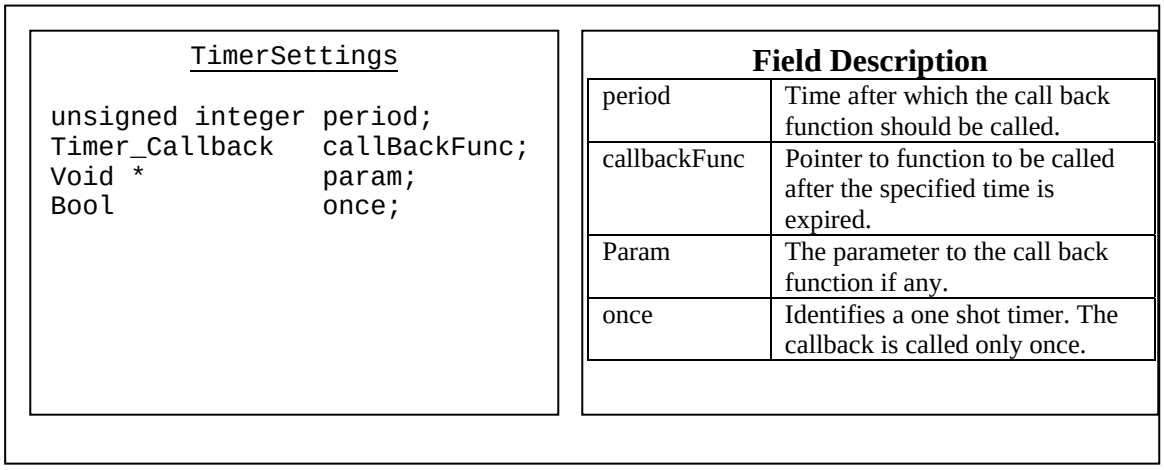


Fig. 19. Representation of timers in framework.

One shot timers are timers which are scheduled to run only once. The callback function is called only once and after that the timer stops running and hence the callback function is never called again. After this, the callback is removed from the list of call backs and hence is never called again.

As the timers are implemented using the operating system specific wait event schemes, the timers are independent of the speed of device. Since the timers use operating system specific functionality they are implemented in operating system specific part of the SDK.

Design of Content Directory Service

Though the Content Directory Service [9] is not a part of the framework but it is a core service of media server device and is responsible for exposing the digital media contents on to the UPnP framework. The content directory service is responsible for the following operations:

- During initialization, the content directory service is passed the absolute path of the file system directory which has the media contents that needs to be exposed.
- This requires the content directory service to recursively scan all the files and directories under the indicated path.
- Each media file is represented using an abstract data type. A database of media files is created. In this implementation the database created is in the form of simple linked lists but since the database is abstracted it is very easy to over write the database implementation and replace this with better implementations.
- The content directory service exposes actions which are used by the control points to browse the contents of the content directory.
- The content directory uses Digital Item Description Language (DIDL) syntax and semantics to expose the media contents. For example, when the “browse” action is invoked then the response to the browse is a string. This string is actually a DIDL document which contains the information about the media items that the content directory service is exposing to the UPnP network.
- The details about the action of the content directory service are described earlier. Refer to section UPnP Media Server Device [6].

During the scan of the file system, the content directory service creates a database of all the media items. The database consists of containers and actual media items. The database is created as follows:

- Since the media items are categorized as audio, video and photo items before the scan, three *containers* are added to the database, one for each type of the media content. The container is the object that holds the media items.

- The scan is initiated. The scan is initiated before the SSDP advertisements are broadcasted. This is because all the services exposed by the media server device should be initialized and are in a state where the actions can be invoked on them.
- During the scan, each media item encountered is decoded to determine certain parameters of the digital media item. These parameters include the type of the Media file, the bit rate of the video file, the compression used etc. These parameters are used to determine the DLNA media profile [14]. It should be mentioned that two open source components are used. First the FFMPEG decoder is used to decode the media content. Second, open source DLNA implementation is used to assign DLNA profiles.
- The response of the browse command is a DIDL document which contains a URL to retrieve each media item along with some information about the item. The browse action is the heart of the content directory service and is described in Figure 20.

Note that the data type of the each parameter is defined in the UPnP specifications [5]. When the SOAP request is received by the framework all the parameters are received as string and the framework is responsible for converting the SOAP envelope strings in to actual parameter types and then calling the function.

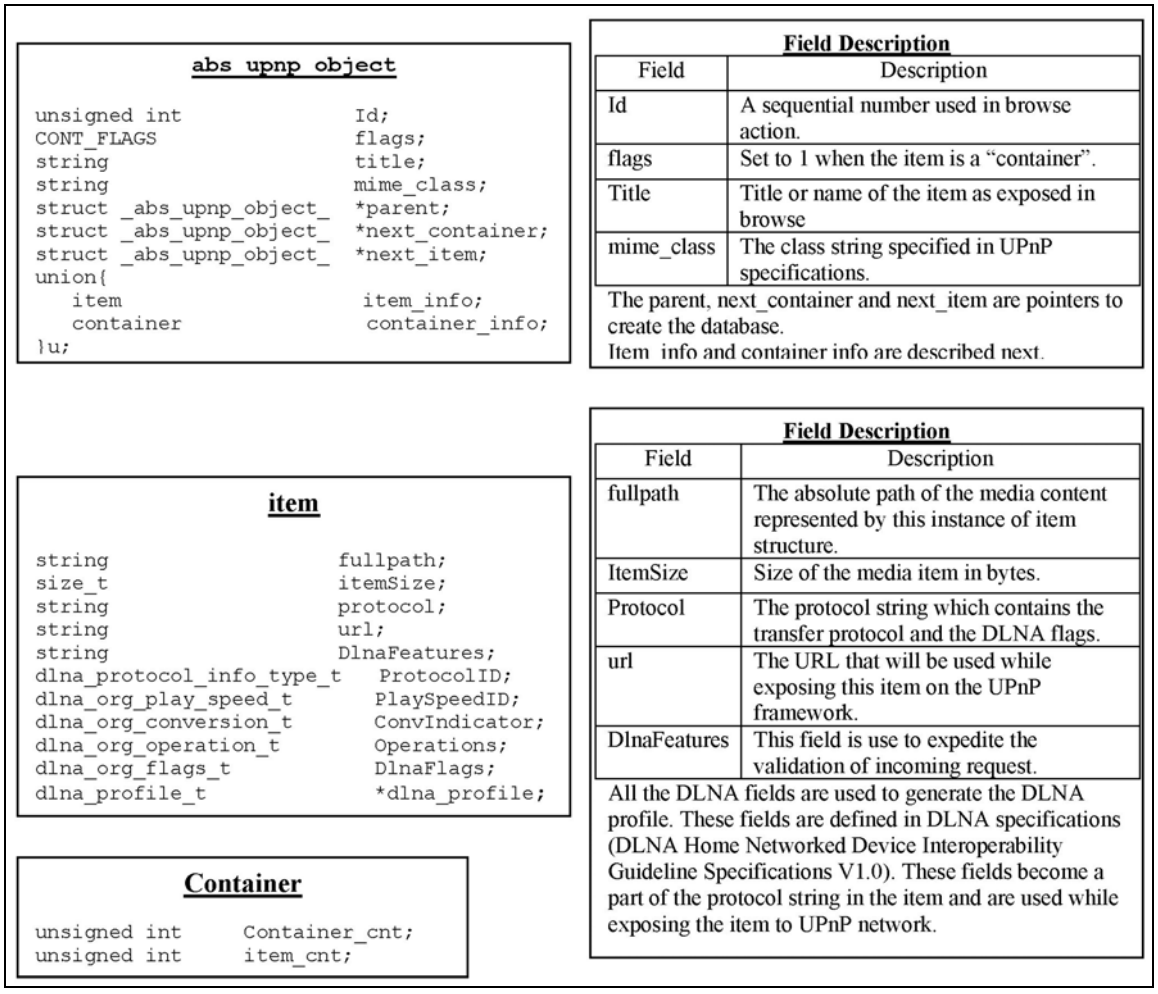
There are two types of items in the content directory service. The container items are the items that are analogous to file system directories. These are responsible to hold other items in them. Other items are the actual digital media items. Both these items are represented using the following data structure (Figure 21).

The Browse Action	
Function Prototype	Browse(ObjectID, Flag, Filter, StartIndex, Count, SortCriteria)
Input Parameters	**Description**
Object-ID Type: unsigned integer	Every container or the media item in the database is assigned an identifier. This parameter to the browse action identifies the container that is required to be browsed.
Flag Type: string	The browse action can be invoked in two modes. In “metadata” mode the browse returns the metadata of the container. The meta data includes information like number of children, identifier of the parent etc. In “browse-children” mode the browse returns the children of the requested container.
Filter Type: string	The browse may be requested to filter-in only the items which are requested in the filter.
StartingIndex Type: unsigned integer	In order to control the size of the response the number of items returned in the DIDL response can be controlled using these two parameters. The StartingIndex specifies the start item which should be included in the response and the count specifies how many items from the starting index should be included.
Count Type: unsigned integer	
SortCriteria Type: string	The browse may be requested to sort the items based on this parameter.
Output Result	
The output of the browse action is a DIDL document. The content directory service implemented in this project returns a result string. This string is the DIDL description of all the items being returned in the browse.	

Fig. 20. The browse action details.

Design of Hyper Text Transfer Protocol (HTTP) Streamer

The content directory service described in previous sections expose the contents. For every media item there is an entry created in the returned DIDL document. This entry also has an URL associated with it. The implementation describes this as “content transfer URL” since this is the URL which will be used to transfer the actual



abs upnp object

```
unsigned int              Id;
CONT_FLAGS                flags;
string                    title;
string                    mime_class;
struct _abs_upnp_object_  *parent;
struct _abs_upnp_object_  *next_container;
struct _abs_upnp_object_  *next_item;
union{
   item                    item_info;
   container               container_info;
}u;
```

Field Description

Field	Description
Id	A sequential number used in browse action.
flags	Set to 1 when the item is a "container".
Title	Title or name of the item as exposed in browse
mime_class	The class string specified in UPnP specifications.

The parent, next_container and next_item are pointers to create the database.
Item_info and container info are described next.

item

```
string                    fullpath;
size_t                    itemSize;
string                    protocol;
string                    url;
string                    DlnaFeatures;
dlna_protocol_info_type_t   ProtocolID;
dlna_org_play_speed_t       PlaySpeedID;
dlna_org_conversion_t       ConvIndicator;
dlna_org_operation_t        Operations;
dlna_org_flags_t            DlnaFlags;
dlna_profile_t              *dlna_profile;
```

Field Description

Field	Description
fullpath	The absolute path of the media content represented by this instance of item structure.
ItemSize	Size of the media item in bytes.
Protocol	The protocol string which contains the transfer protocol and the DLNA flags.
url	The URL that will be used while exposing this item on the UPnP framework.
DlnaFeatures	This field is use to expedite the validation of incoming request.

All the DLNA fields are used to generate the DLNA profile. These fields are defined in DLNA specifications (DLNA Home Networked Device Interoperability Guideline Specifications V1.0). These fields become a part of the protocol string in the item and are used while exposing the item to UPnP network.

Container

```
unsigned int     Container_cnt;
unsigned int     item_cnt;
```

Fig. 21. Data structures representing containers and media content.

media item from the server to the client. The control point uses this URL to transfer the media item.

Here are the steps involved in transfer of a media item from the media server to other devices on the UPnP network.

- The Content Directory Service exposes the media contents along with a URL for every media item. This URL is used for accessing the media item on the server.
- The URL has the following format as shown in Figure 22.

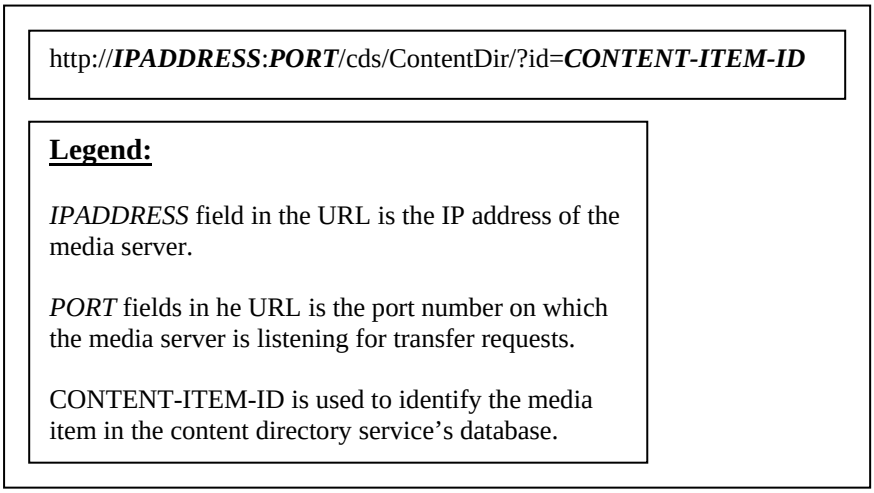


Fig. 22. URL exposed for every media item.

- During initialization, the HTTP-streamer component of SDK creates a socket to listen to the PORT exposed in the URL.

- On receipt of a new connection, the HTTP streamer creates a new thread and listens to the socket to get the HTTP request. Multiple threads are used in the project because the targeted operating systems inherently provide multithreading capabilities independent of the underlying processors. There can be cases where multithreading is not available in which case all the sockets would be monitored in the main thread of execution as we are doing for SSDP, SOAP and GENA implementations.

- There are two HTTP requests handled by the streamer module. The HTTP HEAD request generates a response with only the response headers. The media content data is not a part of the response. The HTTP GET request returns the header and the data in its response.

- Depending on the request type the response is generated and sent to the client.

It should be noted that the HTTP streamer module is not a part of the framework. Many UPnP frameworks have built in HTTP streamer which may also be a

valid design choice since this module is fundamental to the operation of media server devices. Since the requirement was to keep the core framework as small as possible the HTTP streamer module was intentionally kept outside of the core framework. Apart from size another benefit of this design choice was that the HTTP streamer module can be easily replaced with more sophisticated HTTP streaming implementations.

CHAPTER VI

EVALUATION AND CONCLUSION

The project was started with the goal of developing an UPnP framework for embedded devices. Some limitations of other frameworks were studied and certain requirements were outlined as detailed in previous sections. This section evaluates the work and discusses both the achievements as well as some limitations of the framework that need further work.

Goals Achieved

One of the goals of the SDK was that it should be a lightweight implementation. The size of the binary executable object file is less than one mega bytes. This implies that all the required UPnP functionality is available to any embedded device with less than one mega byte of RAM.

A very important requirement was that the SDK should be modularized so that parts of the SDK can be over written if needed for more efficient implementations. The SDK is developed such that each source file has a corresponding header file. The header file acts as an “interface” to the source file. It contains API declarations and data structure definitions required to use the functionality provided by the source file. For example the Simple Service Discovery Protocol is implemented in file named “SSDP.C” and a header file named “SSDP.h” is used to expose the APIs from the source file. This is a basic

design technique that is followed throughout the implementation to organize the code in modular fashion. Any implementation can be changed by changing the corresponding C file and exposing the APIs for the rest of the framework.

Since the operating system specific calls are abstracted the frame work is inherently operating system independent and was deployed on Linux and Microsoft® Windows operating systems. The SDK was also deployed on Broadcom Corporation's MIPS based embedded device. The results obtained in all the three environments were identical and the SDK framework behavior was as expected. The biggest advantage of this level of abstraction is that the SDK itself can be debugged on any powerful IDE environment and then deployed on the embedded device, greatly enhancing the productivity while the UPnP devices are being developed.

In summary the major goals achieved are:

- Operating system independence.
- Modularity.
- Less resource intensive implementation.
- The framework can be deployed on a Linux or a windows PC and also on embedded devices. This greatly increases the debug ability since the identified problems can be reproduced and debugged on a friendly environment, a solution can be developed and then deployed on the embedded device.
- The framework has the ability to overwrite certain default implementations if necessary. The framework is developed in such a way that every source code file has an interface header file associated with it. This means that the source code file can be rewritten without any changes to the framework provided that the interface to the

framework remains the same. The interface file provides the API to rest of the framework.

- As described in the design description the framework is very easy to use. The next section describes this in more details.

Usability Evaluation

Any software development kit with complex interfaces cannot be very successful because of the complexities involved in using it. This project paid special attention to make the interfaces simple and easy to understand and use.

As stated earlier in the design section (Chapter V), in order to use the functionality provided by the framework, the UPnP media server should be registered with it. It was also illustrated that the registration process involves filling up the informational data structures and calling appropriate APIs. There are only five APIs that were needed to register the device with the framework. This makes the framework very easy to use.

Once the device is registered, the framework has all the information needed to provide the mandatory UPnP functionality to the device. This includes the functionality related to addressing, discovery, control, and eventing phases of the framework.

In summary, the majority of the functionality can be obtained using this framework with very simple interfaces exposed by the framework. After the device is registered there is no interaction of the device with the framework during any phase for basic operation. In control phase the actions exposed by different services are called and

hence the developer using the framework does not need to parse the SOAP envelopes since it is the responsibility of the framework.

Performance Evaluation

The performance of the framework was measured in terms of CPU utilization and memory usage. The performance numbers differ in different phases of operation of the media server. The numbers also depend on the number of clients connected and the amount of data being transferred. The performance numbers presented here were taken on a personal computer with a configuration described in Table 6.

TABLE 6
CONFIGURATION USED FOR PERFORMANCE EVALUATION

Configuration Parameter	Parameter Value
Machine Type	Personal Computer. (Netbook PC)
Processor	Intel Atom Single Core.
Processor Clock Speed	1.1 GHz.
Physical Memory	1 Giga bytes.
Front Side Bus	400 MHz
Operating System	Windows XP Professional Service Pack-3.

In addition to the above configuration the test setup included the following:

- The media server device running on a PC with above configuration.
- Windows Media Player® on Microsoft Windows-7® has all the capabilities to act as an UPnP client. It has inbuilt control point and an UPnP media renderer device. This was used as a client device for the test.

To measure the performance of the framework while it was sending advertisements, the media server device was started. At this time only the media server

device existed on the network. Once the test result was obtained the Windows Media player was added to the network. This resulted in device discovery initiation. The peak CPU and memory utilization were noted between the time intervals of introduction of the Windows Media Player device to the UPnP network and the UPnP Media server was recognized in Windows Media Player.

Once the media server was recognized by the Windows Media Player the contents exposed by it were browsed using the content directory service exposed by the UPnP media server.

Final performance analysis included measurements during actual streaming sessions. First, the performance numbers were recorded with single streaming session between the Windows Media Player and UPnP Media Server Device. During the streaming session, the Intel AV Media renderer device was introduced on the network and using the Intel AV UPnP Control Point a new streaming session was initiated and performance data was recorded.

It should also be mentioned that the system running the Media Server device had an idle time CPU utilization of zero percent. Table 7 gives the recorded performance data.

These numbers may vary if a different video clip is chosen for testing. The reason is that the data written on the socket needs to be decoded on the client side. If the video clip is high bit rate clip with better compression then the client may not be able to consume the clip at the same rate at which the server is writing. This will result in socket busy conditions and the media server will be throttled.

TABLE 7
PERFORMANCE TESTING RESULTS

Performance Test Description	Recorded Data		Comments
	CPU Utilization (Peak)	Process Peak Memory Usage (MB)	
Media Server Sending Advertisements	~ 0 % (Same as idle)	2 MB	Sending advertisements is a light weight task and is not CPU intensive. Hence not much memory is utilized.
Discovery Phase On Introduction of Windows media Player Device.	~ 0 % (Same as idle)	2 MB	The discovery phase involves same type of processing as needed while sending the advertisements. Hence the same results as above.
Browsing contents using the content directory service.	~2 %	2.2 MB	There is a slight increase in the CPU utilization. When the browse command is received by the framework it calls the browse action in content directory service. A DIDL document is created. This document contains information about the contents that is exposed by Media Server. The size of this document is proportional to the number of digital items being exposed. During this test about 200 photos, 60 video clips and 30 audio files were being exposed.
Single streaming session between the Windows Media Player and Media Server.	~4%	2.8 MB	Video Streaming is a resource intensive task and hence there is an increase in CPU utilization. The data was read from the file in 64 KB blocks and written to the socket. NOTE: During development it was observed that best performance was achieved when the data was transferred in 64 K chunks and hence the reason for the choice of size.
Two streaming sessions. One using Windows Media Player and other using the Intel AV renderer.	~6%	3.3 MB	As expected there is an increase in the CPU utilization of the media server when streaming two video clips to two different connections.

The above test was then extended to more streaming sessions. Sony PlayStation-III has a built in control point and a media rendering device so it can behave as a fully functional UPnP Client. The advantage of using Sony Play Station-III device is that it can open multiple socket connections to transfer a single file. During the development it was observed that there were about twelve simultaneous connections opened by PlayStation to transfer a 700 megabyte file and media server was able to handle the requests with a CPU utilization of 65 %.

The numbers above were recorded on a relatively high-end machine as compared to embedded systems but these actually give an idea of how the framework will perform on an embedded device. In order to get some idea about the performance of the framework similar testing was performed on an actual embedded system. The device used for testing was Broadcom Corporation's Set top box device with the configuration shown in Table 8.

TABLE 8
CONFIGURATION OF BROADCOM SET TOP BOX DEVICE

Configuration Parameter	Parameter Value
Machine Type	A set top box embedded device.
Processor	MIPS based single core called "Zephyer" Processor.
Processor Clock Speed	850 MHz.
Physical Memory	512 MB.
Memory Architecture	Memory is partitioned in to two banks. The memory bus architecture is such that more bandwidth is given to the graphics and video acceleration core rather than CPU.
Operating System	Linux Kernel 2.6.31.

The results on the set top box were almost identical up to four streaming sessions with little but acceptable variations because of platform differences. As was the case with previous test the CPU utilization was ~6% for two streaming sessions and about ~11% for four streaming sessions. There was a drastic increase in CPU utilization and memory usage when the streaming sessions are increased beyond four. It should also be mentioned that these are acceptable performance levels for embedded devices. For example if the assumption is that the target embedded device runs at half the CPU clock frequency as compared to the above device, a fair assumption can be made that the CPU utilization will be double the measured numbers above. This assumption is valid only if other CPU parameters like instruction and data caches, number of instructions issued and memory bandwidth are assumed to be identical. This is a fair assumption because the Broadcom set top box device is a single core, single issue and the Linux kernel used for testing has the instruction and data cache disabled to mimic the actual embedded device.

It should be mentioned that the performance measurements done here are just to get an idea about the performance behavior of the implemented UPnP SDK. It should not be assumed that these numbers will be identical on other embedded systems. Since every embedded device has different processing resources these numbers will widely differ from one embedded device to another.

Competitive Analysis

In this section we analyze the implementation and compare it to generic UPnP SDK implementations currently available in the market. The analysis also brings out the key differentiators and gives a comprehensive view of the achievements of the project.

The analysis is done considering that the SDK would be deployed on embedded devices since this is the major goal of the project (Table 9).

TABLE 9
COMPETITIVE ANALYSIS

SDK / Module Feature	Competitive Analysis	
	Generic UPnP SDK Implementations	This Implementation
Operating System Independence	Not OS independent. Mostly targeted for a single operating system. For example the Intel's SDK is targeted for Linux. Since the SDK is not OS independent porting the SDK to other operating system involves modifying core SDK modules and is not a trivial task. Almost every part of the SDK needs modification to port it to another operating system. This is a considerable effort.	The SDK is inherently designed to be operating system independent. It has already been verified on Windows and Linux operating systems. The OS independence is achieved by placing the OS specific functionality in separate set of files. In order to port this SDK to another OS, only the operating system specific files will need modifications or enhancements. This makes portability easier.
Enhanced Debugging Capability	The debugging capabilities are dependent on the embedded device on which SDK is being deployed. If the hardware does not have debugging capabilities (like UARTS or JTAGS) it is very hard to debug the SDK. The debugging capabilities are also dependent on the OS for which the SDK is deployed.	Since the SDK is inherently independent of the operating system it can execute on an embedded device as well as on any personal computer. Hence it can be debugged under powerful debugging environments like Visual Studio. This makes the debugging independent of the embedded hardware and the operating system on which the SDK is being targeted. Reported problems can be reproduced and then debugged under any powerful debugging environment. This is the most powerful feature of this implementation and renders great benefits to the embedded software developers.
Expandability	Most of the available SDKs are delivered as binaries and hence cannot be expanded. In case of open source SDKs, like the Intel's, the modules are dependent on each other and it is very difficult to modify one module without impacting others. Considerable amount of work is needed to modify existing module.	The SDKs implementation is modularized. Each code file has an interface header file which exposes the functionality provided by the source file. It is trivial to expand or modify any module by adding new features to the source file. Corresponding interfaces can then be added to the header files so that other parts of the SDK can use them.

Table 9 (Continued)

SDK / Module Feature	Competitive Analysis	
	Generic UPnP SDK Implementations	This Implementation
Suitability for Embedded Devices	The SDK available in market today are feature rich for marketability and hence the size of the binary increases. Embedded devices do not need all these features and only the basic feature set is required.	Only the features fundamental to the operation of the SDK are implemented. This reduces the size of the binary so that it can easily be deployed on low end embedded devices. The size of the compiled SDK library is less than one mega byte.
Simple Object Access Protocol Module	The SOAP module is usually large with very sophisticated parsing capabilities. It has capabilities generate different type of error responses along with different error strings. Since the target for generic implementation is marketability this module also has features that may not be required. For example the Intel SDK's SOAP module has control point functionality built in which implies that it can invoke functions on other UPnP devices.	The SOAP module is very light weight module with only the required parsing capabilities. The size of compiled SOAP object file is about 47 kilobytes. The module only responds with either an error or a success result.
General Event Notification Architecture Module	The GENA module is usually light weight module as the functionality is very limited in scope.	This is a light weight implementation comparable to any other available implementation.
Http Streamer Module	Generic implementations have a feature rich HTTP streamer module which is part of the core SDK. They provide features like chunked encoding. The HTTP streamer also includes a feature rich HTTP parser which has support for most of the HTTP headers. Intel's UPnP SDK includes HTTP parser has support for chunked encoding, elaborate error handling and a very resource intensive parsing engine.	This is very light weight HTTP streamer implementation which is used only to stream the media contents. Only the headers mentioned in the UPnP specifications are implemented and parsed. Chunked encoding is not required because the client requesting the data already knows the size of the content being streamed. This information is passed to the client during the enumeration of the media contents.
Content Directory Service (CDS) Module	The content directory service implementations use some sort of data base for managing the contents. The overhead of the database itself is large and hence the implementation becomes unacceptable for embedded devices. Some implementations provide extensive features like search and sorting which are CPU intensive operations. These operations are optional as per the UPnP specifications.	This implementation of CDS only incorporates the features which are fundamental to the operation of the device. The database is not used and the information about the media contents is maintained in form of linked lists. Optional features like search and sort are not implemented. These are CPU intensive tasks which impact the operation of the media server device.

Table 9 (Continued)

SDK / Module Feature	Competitive Analysis	
	Generic UPnP SDK Implementations	This Implementation
Simple Service Discovery Protocol Module	Generic implementations include control point functionality in their SSDP modules. The control point functionality includes the ability to listen to advertisements and parse them. This way control point devices implemented using the SDK can determine the type of the devices available over the network. Control points needs to know the devices on the network since they are responsible for invoking the services and actions provided by each device.	This implementation only has the SSDP features that are needed by the media server device. This includes the features to send advertisements, renew advertisements, sending response to M-SEARCH requests, sending SSDP-BYEBYE messages. This is the only functionality required by a media server device. Media server devices do not need the functionality of control point devices.

Conclusions

The goal of this project was to implement UPnP specifications for embedded devices. From the reasons outlined in previous chapters it is implied that the existing implementations do not fully cater to embedded devices and hence are not adapted widely. The developed framework has following advantages for embedded devices:

- Thin and lightweight implementation. Only required features are implemented as part of core SDK which makes it less resource intensive.
- The implementation is operating system agnostic and hence can be deployed on many operating systems. The framework was tested on Windows and Linux operating systems. This level of abstraction gives us many benefits including:
    - » The SDK can run on different operating system environments like Windows, Linux etc. This enhances the debug ability. Since embedded devices have limited debug ability this brings great benefits to the developers as the bugs

can be reproduced and fixed in a friendly development environment and then the fix can be deployed on the target embedded system.

» The SDK already has support for Windows and Linux environments. Hence, it is ready for deployment on variety of Linux based embedded devices.

- The SDK is very easy to use. The user of the SDK just needs to register the devices and its services with the SDK. This involves calling SDK APIs. After this is done there is no interaction needed with the SDK except in the control phase where the contents need to be streamed.
- The performance of the SDK has been analyzed on a personal computer and also on MIPS based set top box device from Broadcom Corporation. The performance numbers were identical up to four streaming sessions. Once the number of streaming sessions was increased beyond four the SDK performed better on a personal computer rather than embedded system. This is partially because of two reasons:

» The Broadcom set top box is designed such that more memory bandwidth is given to graphics operations rather than CPU intensive tasks.

» The personal computer device on which the performance tests were performed was inherently faster and had more memory. The level 1 and level two caches also gives it huge performance advantage over the set top box.

- The implementation is in C language, which is widely acceptable language for embedded systems and is highly portable across operating systems.
- The code provides abstractions in such a way that any part of the SDK can be modified without changing other parts of the framework. Every C file implements a functionality which is exposed by its interface header files. This implies that the

implementation can be modified without much effort if the interface to the rest of the framework is maintained.

- There are no restrictions or licensing necessary to use the code. The code will be available free for distribution. This enables the creation of intellectual property assets by different companies and individuals.

Future Enhancements and Recommendations

This section discusses future enhancements that can be of value to this project. The enhancements include the following:

- The UPnP specifications stated that in case the DHCP server is not available the UPnP device should be able to fall back to AUTO-IP in order to obtain the IP address. The SDK assumes that the DHCP server will always be available and does not implement AUTO-IP functionality.

- The parsing of SOAP envelope is effective but there are places where there are some extra memory copies for parameter passing. This can be avoided.

- The HTTP streamer was not developed as a part of the SDK. It is a part of media server device. This means that the SDK does not have support for streaming the exposed media contents. This design choice was considered because it was necessary to keep the size of implementation as small as possible. Another reason is that the streamer component is not a part of UPnP specifications and hence it is kept outside of the core UPnP implementation. But one can argue that an UPnP framework with out support for streaming content may be incomplete.

- The content directory service implemented as a part of media server uses a set of linked lists to maintain information about the digital content that is exposed to the UPnP network. It is recommended that a database be used instead.
- The Eventing architecture implemented in the framework was tested only with one evented state variable. This should be enhanced and tested with multiple evented variables.
- Support for Audio Video transport service is not implemented. This means that the HTTP will be used as a default data transfer protocol. The AV transport service will enable the use of protocols like RTP (Real-time Transport Protocol) and RTSP (Real-time streaming protocol). These protocols have built-in security features and will ensure that the data transferred is not compromised.

REFERENCES

REFERENCES

[1] Digital Living Network Alliance, “About Digital Living Network Alliance,” 2011. Retrieved November 21, 2010 from World Wide Web: http://www.dlna.org/about_us/about/.

[2] Digital Living Network Alliance, “DLNA Home Networked Device Interoperability Guidelines Version: 1.0,” 2004. Retrieved November 21, 2010 from World Wide Web: http://www.homemediaserver.ru/files/Doc/DLNA_Interoperability_Guidelines_v1%255B1%255D.0.pdf

[3] B. A. Miller, T. Nixon, Ch. Tai, and M. Wood, “Home Networking with Universal Plug And Play,” *IEEE Communications Magazine*, pp. 104-109, Dec. 2001.

[4] “Understanding Universal Plug and Play,” white paper, Microsoft Windows ME, 2000.

[5] Contributing Members of the UPnP Forum, “UPnP™ Device Architecture 1.1,” October 15, 2008. Retrieved November 21, 2010 from World Wide Web: http://www.upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf.

[6] Contributing Members of the UPnP Forum, “MediaServer:2 Device Template Version 1.01,” May 31, 2006. Retrieved November 21, 2010 from World Wide Web: http://www.upnp.org/specs/av/UPnP-av-MediaServer-v2-Device.pdf.

[7] UPnP Forum, “International Standards,” *UPnP Specifications*, 2010. Retrieved November 21, 2010 from World Wide Web: http://upnp.org/sdcps-and-certification/standards/.

[8] Contributing Members of the UPnP Forum, “ConnectionManagement:1 Service Template Version 1.01,” June 25, 2002. Retrieved November 21, 2010 from World Wide Web: http://www.upnp.org/specs/av/UPnP-av-ConnectionManager-v1-Service.pdf

[9] Contributing Members of the UPnP Forum, “ContentDirectory:3 Service Template Version 1.0, September 30, 2008. Retrieved November 21, 2010 from World Wide Web: http://www.upnp.org/specs/av/UPnP-av-ContentDirectory-v3-Service.pdf

[10] E. O’Tuathail and M. Rose, “Using the Simple Object Access Protocol (SOAP) in Blocks Extensible Exchange Protocol (BEEP) (RFC:4227),” *IETF Tools*, 2006. Retrieved November 21, 2010 from World Wide Web: http://tools.ietf.org/html/rfc4227.

[11] J. Cohen and S. Aggarwal, “Internet Draft: General Event Notification Architecture Base,” *IETF Tools*, 1998. Retrieved November 21, 2010 from World Wide Web: http://tools.ietf.org/html/draft-cohen-gena-p-base-01.

[12] T. Cai, P. Leach, Y. Gu, Y. Y. Goland, “Internet Draft: Simple Service Discovery Protocol/1.0,” *IETF Tools*, 1999. Retrieved November 21, 2010 from World Wide Web: http://tools.ietf.org/html/draft-cai-ssdp-v1-00.

[13] Digital Living Network Alliance, *Home Networked Device Interoperability Guidelines: Volume 1: Architecture and Protocols*. Lake Oswego, OR: DLNA, October 2006.

[14] Digital Living Network Alliance, *Home Networked Device Interoperability Guidelines: Volume 2: Architecture and Protocols*. Lake Oswego, OR: DLNA, October 2006.

[15] Allegro Software Development Corporation, “Networked Digital Media Standards: A UPnP/DLNA Overview,” 2006. Retrieved November 21, 2010 from World Wide Web: http://www.allegrosoft.com/UPnP_DLNA_White_Paper.pdf

[16] J. Catsoulis, *Designing Embedded Hardware*. Sebastopol, CA: O’Reilly Media, 2005.

[17] UPnP Forum, “Software Development Kits (SDKs),” *Standardized DCPs & Certification*, 2010. Retrieved November 21, 2010 from World Wide Web: http://upnp.org/sdcps-and-certification/resources/sdks/

[18] SourceForge, “Intel® UPnP SDK for Linux,” 2011. Retrieved November 22, 2010 from World Wide Web: http://sourceforge.net/projects/upnp/

[19] M. Jeronimo and J. Weast, *UPnP Design by Example: A Software Developer’s Guide to Universal Plug and Play*. Santa Clara, CA: Intel Press.

[20] Open Source Projects, “Projects,” n.d. Retrieved November 26, 2010 at World Wide Web: http://opentools.homeip.net/

[21] B. Zores, “Reference DLNA Open-source Implementation for Linux,” n.d. Retrieved November 24, 2010 from World Wide Web: http://libdlna.geexbox.org/

[22] FFMPEG, “FFmpeg Digital Media Decoder,” n.d. Retrieved November 24, 2010 from World Wide Web: http://www.ffmpeg.org/

APPENDIX A

THE DEVICE DESCRIPTION DOCUMENT

The template for the device description document [1] as described by the UPnP working committee is shown below.

```
<?xml version="1.0"?>
<root xmlns="urn:schemas-upnp-org:device-1-0"
configId="configuration number">
<specVersion>
        <major>1</major>
        <minor>1</minor>
</specVersion>
<device>
        <deviceType>urn:schemas-upnp-org:device:deviceType:v</deviceType>
        <friendlyName>short user-friendly title</friendlyName>
        <manufacturer>manufacturer name</manufacturer>
        <manufacturerURL>URL to manufacturer site</manufacturerURL>
        <modelDescription>long user-friendly title</modelDescription>
        <modelName>model name</modelName>
        <modelNumber>model number</modelNumber>
        <modelURL>URL to model site</modelURL>
        <serialNumber>manufacturer's serial number</serialNumber>
        <UDN>uuid:UUID</UDN>
        <UPC>Universal Product Code</UPC>
        <iconList>
                <icon>
                        <mimetype>image/format</mimetype>
                        <width>horizontal pixels</width>
                        <height>vertical pixels</height>
                        <depth>color depth</depth>
                        <url>URL to icon</url>
                </icon>
                <!-- XML to declare other icons, if any, go here -->
        </iconList>
        <serviceList>
                <service>
                        <serviceType>urn:schemas-upnp-org:service:serviceType:v</serviceType>
                        <serviceId>urn:upnp-org:serviceId:serviceID</serviceId>
                        <SCPDURL>URL to service description</SCPDURL>
                        <controlURL>URL for control</controlURL>
                        <eventSubURL>URL for eventing</eventSubURL>
                </service>
                <!-- other services defined by a UPnP Forum working committee(if any) go here >
                <!-- Declarations for other services added by UPnP vendor (if any) go here -->
```

```
        </serviceList>
        <deviceList>
                <!-- Description of embedded devices defined by UPnP (if any) go here -->
<!-- Description of embedded devices added by UPnP vendor (if any) go here --></deviceList>
        <presentationURL>URL for presentation</presentationURL>
</device>
</root>
```

REFERENCES

[1] UPnP Forum, “UPnP™ Device Architecture 1.1,” October 15, 2008. Retrieved November 21, 2010 from World Wide Web: http://www.upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf

APPENDIX B

SERVICE DESCRIPTION DOCUMENT

The UPnP service description template document [1] is shown below.

```
<scpd
xmlns="urn:schemas-upnp-org:service-1-0"
xmlns:dt1="urn:domain-name:more-datatypes"
<!-- Declarations for other namespaces by UPnP Forum working committee(if any) gohere -->
<!-- The value of the attribute must remain as defined by the UPnP Forum working committee-->
xmlns:dt2="urn:domain-name:vendor-datatypes"
<!-- Declarations for other namespaces added by UPnP vendor (if any) go here -->
<!-- Vendors must change the URN's domain-name to a Vendor Domain Name -->
<!-- Vendors must change vendor-datatypes to reference a vendor-defined namespace -->
configId="configuration number">
        <specVersion>
                <major>1</major>
                <minor>1</minor>
        </specVersion>
        <actionList>
                <action>
                        <name>actionName</name>
                        <argumentList>
                        <argument>
                                <name>argumentNameIn1</name>
                                <direction>in</direction>
                                <relatedStateVariable>stateVariableName</relatedStateVariable>
                        </argument>
<!-- Declarations for other IN arguments defined by UPnP Forum working Committee (if any) go here -->
                        <argument>
                                <name>argumentNameOut1</name>
                                <direction>out</direction>
                                <retval/>
                                <relatedStateVariable>stateVariableName</relatedStateVariable>
                        </argument>
                        <argument>
                                <name>argumentNameOut2</name>
                                <direction>out</direction>
                                <relatedStateVariable>stateVariableName</relatedStateVariable>
                        </argument>
                        <!-- Declarations for other OUT arguments defined by UPnP Forum working(if
                        any) -->
                        </argumentList>
                </action>
<!-- Declarations for other actions defined by UPnP Forum working committee(if any)go here -->
<!-- Declarations for other actions added by UPnP vendor (if any) go here -->
```

```
        </actionList>
        <serviceStateTable>
                <stateVariable sendEvents="yes"|"no" multicast="yes"|"no">
                        <name>variableName</name>
                        <dataType>basic data type</dataType>
                        <defaultValue>default value</defaultValue>
                        <allowedValueRange>
                                <minimum>minimum value</minimum>
                                <maximum>maximum value</maximum>
                                <step>increment value</step>
                        </allowedValueRange>
                </stateVariable>
                <stateVariable sendEvents="yes"|"no" multicast="yes"|"no">
                        <name>variableName</name>
                        <dataType type="dt1:variable data type">string</dataType>
                        <defaultValue>default value</defaultValue>
                        <allowedValueList>
                                <allowedValue>enumerated value</allowedValue>
                                <!-- Other allowed values defined by UPnP Forum working committee
                                (if any) go here -->
                                <!-- Other allowed values defined by vendor (if any) go here -->
                        </allowedValueList>
                </stateVariable>
                <stateVariable sendEvents="yes"|"no" multicast="yes"|"no">
                        <name>variableName</name>
                        <dataType type="dt2:vendor data type">string</dataType>
                        <defaultValue>default value</defaultValue>
                </stateVariable>
<!-- Declarations for other state variables defined by UPnP Forum working committee(if any) go here -->
<!-- Declarations for other state variables added by UPnP vendor (if any) go here -->
        </serviceStateTable>
</scpd>
```

REFERENCES

[1] UPnP Forum, “UPnP™ Device Architecture 1.1,” October 15, 2008. Retrieved November 21, 2010 from World Wide Web: http://www.upnp.org/specs/arch/UPnP-arch-DeviceArchitecture-v1.1.pdf