Mobile Communication in Distributed Environment
Abstract
The thesis purpose was to develop a framework for coordination of devices over mobile environment. The framework is specially designed for mobile devices due to constraints in mobiles of less memory and resources, enables zero configuration for communication. The purpose of study was to investigate existing tools and technologies for mobile communication. For this purpose UPnP (Universal Plug and Play) architecture is investigated over Prism-MW (Programming in the small and many - middleware) for android platform. We have developed the control point devices communicate with server devices. And also we have developed demonstrator application to show how devices will work.
Acknowledgements
We are heartily thankful to our supervisor, Jesper Andersson, whose encouragement; guidance enabled us to develop an understanding of the subject.

We are also thankful to all of those who supported us during the completion of the thesis.
# Content

## 1 INTRODUCTION ................................................................. 1

1.1 Problem Definition .................................................................. 1
1.2 Research Question .................................................................. 1
1.3 Scope .................................................................................. 2
1.4 Goals .................................................................................. 2
1.5 Motivation ........................................................................... 2
1.6 Plan .................................................................................... 2
1.7 Report Structure ................................................................... 3

## 2 BACKGROUND ........................................................................ 4

2.1 Coordination Frameworks ........................................................ 4
   2.1.1 Universal Plug and Play (UPnP) ........................................ 4
2.2 Jini ..................................................................................... 5
2.3 Development Frameworks for Mobiles ...................................... 5
   2.3.1 Prism-MW .................................................................. 5
   2.3.2 OSGi ....................................................................... 7
2.4 Android .............................................................................. 9
2.5 Existing Systems/Implementations ........................................... 9

## 3 DESIGN AND IMPLEMENTATION ............................................ 10

3.1 Server Side .......................................................................... 10
   3.1.1 Server Side Architecture .............................................. 10
   3.1.2 Server Side Class Diagram .......................................... 11
3.2 Client Side .......................................................................... 13
   3.2.1 Client Side Architecture ............................................ 13
   3.2.2 Client Side Class Diagram .......................................... 13
3.3 Problems in Implementation .................................................. 16
3.4 Evaluation .......................................................................... 16
   3.4.1 Use Case 1: Send Notifications .................................... 17
   3.4.2 Use Case 2: Receive Notifications ............................... 17
   3.4.3 Use Case 3: Discover Device ...................................... 18
   3.4.4 Use Case 4: Describe Services ................................. 19
   3.4.5 Use Case 4: Call Services ......................................... 20

## 4 CONCLUSION & FUTURE WORK .......................................... 22

4.1 Conclusion .......................................................................... 22
4.2 Contributions ..................................................................... 22
4.3 Future Work ...................................................................... 22

## 5 REFERENCES ........................................................................ 24

APPENDIX A ........................................................................ 25

Device Description .................................................................. 25

APPENDIX B ........................................................................ 26

Service Description .................................................................. 26
List of figures
Figure 2-1 Prism-MW Class Diagram [8] .......................................................... 6
Figure 2-2 OSGi Layered Architecture ......................................................... 8
Figure 3-1 Server Side Architecture .............................................................. 10
Figure 3-2 Server side Class Diagram ........................................................... 12
Figure 3-3 IServiceNotifier ......................................................................... 13
Figure 3-4 Client side Architecture ............................................................. 14
Figure 3-5 Client side Class diagram ............................................................. 15
Figure 3-6 IDeviceNotifier and IServiceNotifier Class Diagram ................. 16
List of Abbreviations
1. UPnP: Universal Plug and Play
2. Prism-MW: Programming in the small and many – middleware
3. SSDP: Simple Service Discovery Protocol
4. HHTP: Hyper Text Transfer Protocol
5. SOAP: Simple Object Access Protocol
6. GENA: General Event Notification Architecture
7. URL: Uniform Resource Locator
8. IP: Internet Protocol
9. UDP: User Datagram Protocol
10. TCP: Transmission Control Protocol
11. PDA: Personal Digital Assistant
12. LAN: Local Area Network
13. WAN: Wide Area Network
1 Introduction

Technology advancement in handheld devices and wireless networking has led these little mobile computing devices into distributed system. [16]

“A distributed system is a heterogeneous system. Diverse system components are connected to each other via a common network”. [15]

These computing devices including PDAs and mobile phones have two basic advantages over the basic computers, they are portable so we can easily move them and also they are easily able to connect to a network in different places. We can easily move these devices into different places and can take advantages of local and remote services. This type of distribution which includes the mobile devices and other devices is called spontaneous networking. For example devices which are embedded in the appliances give services to the users as well services to the other devices. It will be very useful for a user to control washing machine from a mobile device in their home network and similarly washing machine can notify the user on his mobile device that washing is done. [16] This type of communication between mobile to mobile and mobile to other devices is our research area.

1.1 Problem Definition

With the invention of PDAs and handheld devices, need of coordination between mobile devices is increased. We can describe coordination between mobile devices as [12]:

- Announce presence of a mobile device on the network
- Discovery of mobile devices
- Ability to describe the services
- Query/understand the services of the other devices

The coordination has become a serious research issue. The idea is to explore this huge field creating a fix point, which will be the basis for future projects. The approaches we will use are based on surveys of existing approaches and by implementing demonstrators, understand what is possible and what is not. This thesis should develop a framework(s) that are aligned and integrated with existing mobile device technologies. Demonstrator applications, built on top of the framework, will be used to evaluate the framework parts and their integration

1.2 Research Question

As our thesis will be on study of different protocols for communication, we will try to answer these questions:

- How a device can notify the other devices when going online/offline?
- How devices will know about each other’s services supported?
- How devices will send and receive the data with other devices?
- How devices notify the other devices when any event occurs?
1.3 Scope
This thesis is exploratory, which implies that it contains a survey of available techniques, tools, and approaches particularly in the mobile device domain. In addition the thesis should explore and identify the limits with focus on the Android platform.

- Explore the available tools and protocols for communication between devices.
- Implement the protocols if needed.

1.4 Goals
The main goal of the thesis is to design a framework for communication between mobile devices which will implement server and client devices. Server devices will have services which will send data to client devices when these services will be called.

The main goals will be:
1. Client devices should be able to detect server devices automatically over the network.
2. Client devices should be able to get server device’s services automatically.
3. Client devices should be able to get data from server device’s services.
4. When any service state changed on the server device it should send notifications to the client side devices.

1.5 Motivation
In this busy world of computing every person needs and finds ways for the computing to be done remotely. Mobile computing is a way to achieve an individual’s remote demand, for instance the person can access his/her home camera from mobile. In this regard the mobile computing has its own milestones to be considered which involve:

- Mobile devices have less memory to handle more communication
- Computation affects the performance of communication
- Large no of devices come on the network
- A device can online to and offline from the network dynamically.
- A mobile device can get a device’s data from other nearest mobile device
- A device is a general device.

The issues involved in mobile computing inspire us to develop such a framework which implements UPnP architecture.

The framework will then be used for the future projects applications which might react like devices or control points.

Developers will be able to configure devices on mobile devices easily by using the framework.

Android Systems are latest incarnation in the field of mobile computing which claim to be efficient and successful systems. The framework will use android platform because it gives the mobile environment to connect devices easily.

1.6 Plan
We plan to survey UPnP (Universal Plug and Play) device architecture. This survey will be the basis for deriving suitable descriptions and protocols for mobile devices. Then we plan to implement proof of concept demonstrators by extending Prism-MW with UPnP specific connectors on android platform.
"The UPnP architecture offers pervasive peer-to-peer network connectivity of PCs of all form factors, intelligent appliances, and wireless devices."[2]

"PrismMW is an extensible middleware platform that enables efficient implementation, deployment, and execution of distributed software systems in terms of their architectural elements: components, connectors, configurations, and events."[8]

"Android is a software stack for mobile devices that includes an operating system, middleware and key applications."[9]

Further, we plan to evaluate our approach by introducing our demonstrator in a number of simulated and real environments.

1.7 Report Structure

The core of the report covered in chapters 2-4.

Chapter 2 describes the background of the Platform (Android which we used for the development of our Mobile Communication Application), background of new suggested frameworks for Componentized Software namely PRISM & OSGI, and background of Device Architecture UPnP (which is standard for Device Communication).

Chapter 3 presents the Design & Implementation. In the section we show high level Architecture of the system we are developing, then Class Diagram, and the Use-Case scenarios.

Chapter 4 shows conclusions, contributions as well as suggestions for future work, for Mobile Communication between devices.
2 Background

In this chapter we are discussing existing coordination and development frameworks.

2.1 Coordination Frameworks

Coordination Frameworks provide architectures to communicate devices over the network. Here we are discussing two major coordination frameworks; UPnP, and Jini.

2.1.1 Universal Plug and Play (UPnP)

UPnP devised by UPnP forum [1]. UPnP Forum started in October, 1999. The forum now includes more than 912 companies [2].

UPnP is a technology to provide a framework for connected stand-alone devices on networks, to communicate with each other in a simple and robust manner.

The UPnP term originated from PnP (Plug and Play). Plug and Play technology used for to connect devices to computers dynamically. Although UPnP is not similar with Plug and Play technology completely because PnP attaches devices to computers and UPnP attaches devices to networks [3].

- **Overview**
  UPnP supports zero configuration networking. Through UPnP a device can automatically be detected on a network, announce its services so other devices can know. [5]

- **UPnP Layers**
  There are 6 basic layers of the UPnP [6]:
  - **Addressing**
    In addressing a device get an IP address. IP address can be set manually or can be taken through DHCP (Dynamic Host Configuration Protocol) from a DHCP server. The main focus of this layer is to get an IP for the device.
  - **Discovery**
    In Discovery a device announce its presence on the network. SSDP (Simple Service Discovery Protocol) is used over UDP. When a device starts it sends SSDP Alive packets on an UPnP multicast address on the router. The devices which are interested to know about the other devices will be listening on the multicast port. Similarly when a device wants to go offline it will send the SSDP ByeBye packets on the multicast address.
  - **Description**
    After discovery the next step is description. In discovery SSDP Alive packets has a device description URL. A device description includes device information includes name, manufacturer details and general details about its services. It has a relative or base URL for the device service’s description also. Service description includes the device supported actions information like name and input/output arguments and status variables information. The interested devices get the device description URL from the discovery layer and use the Http to get the device and service descriptions. For sample xml file, refer to appendix A for Device Description and appendix B for Service Description.
  - **Control**
    After knowing the services information these services can be called using the SOAP (Simple Object Access Protocol). Device description includes the url of each service where these requests can be made. The interested devices used that url to call the action or query the status variables.
• **Eventing**
Through eventing a device can notify the other devices about its service status changed. For this purpose the interested devices subscribe itself for a particular service to the device. After subscription a device can send notification messages when that service status changed. For notification messages GENA (General Event Notification Architecture) protocol is used over Http. If a device wants to no longer receive these notification, that device can unsubscribe from these events through GENA.

• **Presentation**
A device can have a webpage, depending on the webpage capabilities a user can control the device or can know the status information. [6]

2.2 **Jini**
Jini from Sun Microsystems makes a network which allows plug and play devices to join or leave the network without the need of any configuration. [21]

The main component of a Jini system is a lookup service, which register devices and services. Jini uses the term federation for the coordination between devices. [20] Federation is set of lookup services which dynamically hold the information about registered devices.

• **Lookup Services**
Lookup services should be running all the time on a network. Using unicast or multicast devices can find these services and register their self. A device must register itself to a lookup service to enter into federation. Devices can register with a set of properties of name/value pair so other devices can find them easily.

• **Exposing Interfaces**
In registering a device can upload java code to the lookup service. This is called “Proxy” and that code can be used to call an interface on the device. So a querying device can automatically download that code and can call methods of registered device. This is done through Java RMI (Remote Method Invocation) which is used to call the remote methods.

• **Invoking Services**
Invoking services means calling the remote methods through RMI so there should be Java on both sides. These methods can be well defined for a standard device type or can be get from look up services by “Proxy”. [20]

2.3 **Development Frameworks for Mobiles**
Now we have to choose which framework or architecture for componentized software is best. There are many but we study Prism and OSGI.

2.3.1 **Prism-MW**
Prism-MW (Programming in Small and Many) is a style of software architecture, used to overcome the complexity of software in highly distributed, mobile, handheld, and heterogeneous environment [7]. It is a middleware for applications in the environment [4].

Development of large systems involves more complexity than development of small systems or programs. And the complexity even increases by decreasing the cost of and by putting more flexibility in hardware. The complexity also increased by the factor of distributed environment, mobility. This shows a new challenge to be faced by the computer scientist and software engineers. The technology Prism-MW is a solution to reduce the complexity inherent in large and distributed systems in an efficient manner [7].
Prism-MW is a lightweight architectural implementation for distributed software systems. It represents a framework containing extensible classes from which an application can develop its architecture easily as illustrated in Figure 2.1 [7].

Developers can easily reuse the classes to fulfill their specific needs for the application they are developing. The five classes are important for a developer to develop an architecture-based application: Architecture, Component, Connector, Port, and Event [7].

Common properties or features are held in the super class ‘Brick’ which is an abstract class of the above five important classes [4]. Components’, connectors’, and ports’ configurations are kept in Architecture. Architecture also facilitates components, connectors, and ports to be added, removed, reconnected, or reconfigured. Distributed Applications are implemented by connecting different architectures [4].

In architecture communication is done by setting Event. Event contains name and a set of parameters. Parameters contains the actual data information having parameter name, sender information can also be set in the parameters by developers. Events are of two type request events and reply events, so that a sender will set the type of event being sent to request if it wants some action or information to be get from the recipients of the request event. Same case with the reply event, in this case the sender will send event by setting its type to reply if it wants to response for some request [4].

Component is mainly computational part of the distributed applications. The whole logic of the applications is distributed over these components. The components save the states them-self within an architecture. Components can have multiple ports for other components to communicate with each other so that they can directly communicate. The components can also communicate with other components via connectors in the

![Figure 2-1 Prism-MW Class Diagram](image-url)
architecture. Like components, connectors can also have multiple ports to connect the components. A component can connect its one port to the port of connector so that it can communicate to other components through this connector. A component can dynamically change its implementation without replacing the whole component at system runtime [4].

2.3.2 OSGi
OSGi is a dynamic module system for Java [17]. OSGi divides the software into components/modules known as bundles. The bundles can communicate with each other through services using the framework provided by OSGi.

Bundles register services to export their functionalities and other bundles get those services to call the functionalities.

OSGi facilitates software developers to concentrate on essential parts of the software not the problem itself.

The complexity of the software is reduced by adopting OSGi.

Bundles hide their implementations from other bundles and provide interfaces through services to the bundles. The bundles which want to use the registered bundles on some services, listen to those services to communicate with each other.

The bundles which are components can be dynamically attached to the framework of OSGi, bundle framework environment.

OSGi is a layered architecture [18], illustrated in Figure 2-2.
“Bundles - Bundles are the OSGi components made by the developers.
Services - The services layer connects bundles in a dynamic way by offering a publish-find-bind model for plain old Java objects.
Life-Cycle - The API to install, start, stop, update, and uninstall bundles.
Modules - The layer that defines how a bundle can import and export code.
Security - The layer that handles the security aspects.
Execution Environment - Defines what methods and classes are available in a specific platform.” [18]

As in Figure 2-2, the OSGi framework can be defined as no of layers which are stacked on top of other layers and the lowest layer is Operating System. Execution Environment is the first layer that is on top of operating system & JVM. With this layer we can say that OSGi starts its visualization & work. The execution of all necessary things is handled here on this layer. Bundles are placed & executed on this layer. The loading of a bundle is defined on second layer Modules Layer. Modules Layer tells the execution environment which classes of the bundle to load and in what sequence and finally how should all bundles to be loaded before they can be operated by other bundles which already running. Module Layer makes the bundles into run state and Life cycle layer controls the way the bundles are running or operating. Services and also known as Service Registry and are objects used to link the communication between bundles and these can be real-world objects like devices for example a networked printer. Bundles Layer covers all four layers; Execution Environment, Modules, Life Cycle, and Services so that all these four layers can operate on bundles which are available to all of them [22].

Using the above framework developers can make software which may be decomposable into well defined bundles.

People also have adopted the OSGi framework for developing distributed applications [19].
2.4 Android

Android is an open source operating system. It was initially developed by Android, Inc which is purchased by Google but lately it is purchased by Open Handset Alliance [10]. Open Handset Alliance is a group of 71 companies who has made group for innovation in mobile technologies. Java programming language is used to create the applications in Android and it also has a customized virtual machine that is designed especially for mobile devices to optimize usage of memory and resources [11].

Android is a mixture of Operating system, middleware, and key applications. It is based on linux kernel which is actually hardware abstraction layer which directly interacts with the hardware, Android Core, and Application Framework [9].

Android application development is done by using the components like activities, services, broadcast-receivers, and content providers [9]. All components except broadcast receivers must be declared in AndroidManifest.xml file of an application. Broadcast receivers can optionally be declared in the same manifest file [9].

2.5 Existing Systems/Implementations

CyberLink for Java is an open source development tool for UPnP Developers. It supports server device creation as well as control points. It is also runnable on android platform. [24] CyberLink uses the Xerers package as the default parser for xml and SOAP requests. [25]
3 Design and Implementation

As we have to design and implement UPnP for both client and server sides, we have divided work in two parts:

- Server Side
- Client Side

3.1 Server Side

It consists of UPnP Device and its services. UPnP device is a server device which has data and that data will be available to clients by calling services.

3.1.1 Server Side Architecture

Server side architecture using Prism-MW is shown in the figure 3.1.

![Server Side Architecture Diagram](image-url)
A UPnP device has services and can also have devices. Outer device is called root device
and inner devices are called embedded devices which can also have services and embedded
devices.

A discovery handler component with SSDP connector will be attached to a root device. Discovery component will be used to send/receive multicast messages on the router. All devices whether embedded or root will be connected to device description handler which is connected to a http server connector. When a request of device description will be made on the http server, it will notify the device description handler, which in turn will get the device description from device and will send the response to the http server, and then http server will send the response.

Similarly all services will connect to Service description handler which is same as device description handler.

Services will also connect to Control handler and Event Handler. Control handler will be used to response the service actions and query about the service status variables on behalf of the Service to which it is connected. Control handler will implement the SOAP. As SOAP implementation is based on the HTTP, SOAP requests will be handled on the HTTP Server. HTTP Server will then send the requests to the Control handler. Control handler will parse the information like action name, input arguments, or status variable name and if request is valid then send the precise information to the service. Service will then send response of the action or query of status variable which will be again transmitted to SOAP response by the control handler and will send to http server.

Event Handler has both http client connector and http server connector. Subscription request will be made on http server which in turn passes these requests to the Event Handler. Event Handler will implement the GENA protocol. It will parse the requests and if the requests are valid it will send parsed information to the service. If server wants to notify the subscribers it will send status variables to the Event handler with list of subscriber URLs. Event handler then will make notification messages and will send these messages to Http client connector.

### 3.1.2 Server Side Class Diagram

The class diagram of a server side is shown in figure 3.2. As we are implementing on top of Prism-MW, our classes will be inherited from Prism-MW Component and Prism-MW Connector classes. There exist also configuration classes which hold the settings for these Component and Connector classes. Device, Service and handlers classes are inherited component classes. Connector class names end with Connector and similarly configuration class names end with Configuration.

IServiceNotifier is the interface which will be used by Service class to notify the action and status variable requests. Its diagram is shown in figure 3.3. It can be used to send live responses to the clients. For example a class has implemented this interface and registers itself to the service. When a request is made of an action the service will send event notification to that class with the action object. So then registered class by using that action object can verify input arguments and also can set output arguments to that action. After calling that function service will send that action to the ‘ControlHandler’ for sending the appropriate response to the client.
Figure 3-2 Server side Class Diagram
On the Client side, main component is Control Point which will detect the UPnP devices on the network. Control Point will maintain a list of available devices and will automatically add or remove devices when devices change their status to online/offline.

### 3.2.1 Client Side Architecture

Client side architecture is illustrated in figure 3.4. Like server side, here also we have Device and Service components but will be used for storing information of the Server-side devices and services. Discovery handler will be used by the control point for listening notification messages which are sent by server side devices and for sending search messages for devices and services. Device and service description handlers will be used to get the description and then parse that description. These handlers will be used only one time for a device or service and after getting the descriptions these handlers will be removed to save the memory.

Service will connect to control handler and event handler like in server-side. But the difference is control & event handler will use http client connector.

Control Handler will be used by the service to control server-side using action calls and status queries. When a request is made of ‘calling an event’ or ‘querying any service status variable’, Service will send that action or status variable to control handler then control handler will send request to http client connector and will get the response of the request from http client connector. Control Handler will then parse the response and will send the results to Service.

Event handler will be used by the service to send subscription or un-subscription requests to the server side service through the http client connector. Http client connector will get the event notifications and accordingly will reply to Service.

### 3.2.2 Client Side Class Diagram

Same as server side client side also consists of components, connectors and configuration classes. Here ControlPoint is a Prism-MW component. And same as server side Device, service and the handler classes are component classes. Connector class’s names are ends with Connector and similarly configuration class’s names are ends with Configuration.

IDeviceNotifier interface will be used by the ControlPoint to send event notifications to the interface implemented class about device added or removed from network. Similarly IServiceNotifier interface will be used by the Service to send events when action or status variable are updated. Due to asynchronous nature of prism components this interface is required. When an action is called, its response cannot be received immediately rather event notification will be sent by service for the response to be received.

Status variable will be updated and event will be fired on either when a request is made to update the status variable or notification has been sent by the server side.
Client Side Architecture

Control Point

Device 1
Service 1

Device n
Service n

SSDP Connector

Discovery Handler

Http Client Connector

Connector

Device Description Handler

Http Server Connector

Control Handler

Event Handler

Http Client Connector

Figure 3-4 Client side Architecture
Figure 3-5  Client side Class diagram
3.3 Problems in Implementation

The main problem we faced was the asynchronous nature between the Prism-MW components and connectors [7].

All the communication between components and connectors is made by sending the events. And we cannot synchronize the request and response events in a sequence. So we have made the solution that adds all the required values in to the event arguments. For example when a request is made on the HttpServerConnector, it is not copying only request data to the event also it is attaching the client socket on which response will be sent to the event argument. The same event will be passed from all components. And when it return with response data HttpServerConnector will detach the socket and data will be sent on that socket.

3.4 Evaluation

We have made demonstrator applications for server side and for client side. Evaluation has been made on number of tests which are:

- Server side application is able to send notifications of its online/offline status.
- Client side is able to get notifications of presence/non-presence of a UPnP device on a network.
- Server side application is able to describe its services when a request is made.
- Client side application is able to request for server side applications’ services.
- Client side application is able to get data from server side.
- Client side application is able to get server side event notifications.
- Client side application is able to detect multiple server side devices running on the network.

Intel has provided UPnP tools for developers which run on windows. The tools include both client side and server side applications.

The above test scenarios are also used with the Intel tools. We have tested and verified our UPnP implementations’ correctness by coordinating of our client side applications with Intel’s server side applications and vice versa. For example, to test first scenario we have enabled Intel’s server side application to send offline/online notifications and then accordingly our client side application is getting the notifications of the status (2\textsuperscript{nd} scenario). In result, all the tests are running successfully.

### Figure 3-6 IDeviceNotifier and IServiceNotifier Class Diagram

<table>
<thead>
<tr>
<th>IDeviceNotifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>*deviceAdded(in device : Device)</td>
</tr>
<tr>
<td>*deviceRemoved(in device : Device)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>IServiceNotifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>*actionUpdated(in action : Action)</td>
</tr>
<tr>
<td>*statusVariableUpdated(in statusVariable : StateVariable)</td>
</tr>
</tbody>
</table>
All the tests can be viewed in more detail using the following use case scenarios:

### 3.4.1 Use Case 1: Send Notifications

<table>
<thead>
<tr>
<th>Use Case Name</th>
<th>Send Notifications</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Brief Description</strong></td>
<td>Server will send notifications, and accordingly our control point on the android mobile will get notification</td>
</tr>
<tr>
<td><strong>Flow Of Events</strong></td>
<td>Run the server device (like Intel provided media server or our device) on windows (By running the server device, the device will automatically be on network). Our client side application running on android platform will get the online status of the server device</td>
</tr>
<tr>
<td><strong>Actors</strong></td>
<td>Developers</td>
</tr>
<tr>
<td><strong>Pre Conditions</strong></td>
<td>Client side application running</td>
</tr>
<tr>
<td><strong>Post Conditions</strong></td>
<td>None</td>
</tr>
</tbody>
</table>

### 3.4.2 Use Case 2 (Receive Notifications)

<table>
<thead>
<tr>
<th>Use Case Name</th>
<th>Receive Notifications</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Brief Description</strong></td>
<td>Control Point on our client side wait for notification to be sent from media server.</td>
</tr>
<tr>
<td><strong>Flow Of Events</strong></td>
<td>Control Point starts the use case after getting the device that it is found(uses: Discover Devices) Control Point waits subscribes to media servers Server device saves the subscription Control point waits for notifications Server device send notifications(uses: Send Notifications) Control Point receives the notifications on our client side android platform.</td>
</tr>
<tr>
<td><strong>Actors</strong></td>
<td>Developers</td>
</tr>
<tr>
<td><strong>Pre Conditions</strong></td>
<td>Control point on our client side is ready to be run</td>
</tr>
<tr>
<td><strong>Post Conditions</strong></td>
<td>None</td>
</tr>
</tbody>
</table>
### 3.4.3 Use Case 3: Discover Device

<table>
<thead>
<tr>
<th>Use Case Name</th>
<th>Discover Device</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Brief Description</strong></td>
<td>Our client side application will detect server devices.</td>
</tr>
</tbody>
</table>
| **Flow Of Events**   | Our client side application sends search messages  
Server device matched to search message  
Our Client Application discovered the device. |
| **Actors**           | Developers      |
| **Pre Conditions**   | Client side application running and will ready to discover devices |
| **Post Conditions**  | None            |

![Image of Device Spy](image.png)

http://194.47.121.119:1124/

Media Server (SUNNY-1646)
### 3.4.4 Use Case 4: Describe Services

<table>
<thead>
<tr>
<th>Use Case Name</th>
<th>Describe Services</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Brief Description</strong></td>
<td>Server device will send description of its services on the request from client side application</td>
</tr>
</tbody>
</table>
| **Flow Of Events**   | Client Side application sends the request for the description server device’s services  
Server side application sends the description of its services |
| **Actors**           | Developers |
| **Pre Conditions**   | Client side application running |
| **Post Conditions**  | Control point have received the services description |

![Image of a media server interface](image)
3.4.5 Use Case 4: Call Services

<table>
<thead>
<tr>
<th>Use Case Name</th>
<th>Call Services</th>
</tr>
</thead>
<tbody>
<tr>
<td>Brief Description</td>
<td>Our android-based Client applications can call any service of any device on the network</td>
</tr>
<tr>
<td>Flow Of Events</td>
<td>Client Side application invoke/call the services to be performed on server device’s. Server will perform the service action and will update the status variables and will notify to the client for status variables. (uses: Send Notifications)</td>
</tr>
<tr>
<td>Actors</td>
<td>Developers</td>
</tr>
<tr>
<td>Pre Conditions</td>
<td>Client side application has discovered devices and service descriptions</td>
</tr>
<tr>
<td>Post Conditions</td>
<td>Control point have received the services description</td>
</tr>
</tbody>
</table>

![Diagram of network connection interface with options: GetCurrentConnectionIDs, GetCurrentConnectionInfo, GetProtocolInfo]
Although Cyberlink has given implementation of UPnP for android but it is not specially designed for mobiles. Our implementation especially built for mobile devices as mobile devices has memory and resource constraints. Cyberlink gives the implementation as Programming in Large but we has given the implementation in Programming in small and many. Also Cyberlink uses the Xerers parser to parse the xml and SOAP packets. But we are using SAX for parsing the xml and DOM is used for SOAP packets. These parsers are builtin for android platform.

If we use OSGi framework for these scenarios then we can find that it is difficult to have all the services be available for the computational part bundles before the services can be performed on bundled. But in the case of Prism-MW, all these scenarios can be easily handled after having the components connected.
4 Conclusion & Future Work

In this chapter we are discussing our conclusion and contributions about the thesis. Furthermore we are discussing about future improvements which can be done later.

4.1 Conclusion

This thesis attempts to take a survey of some coordination frameworks like UPnP and Jini. We have found that Jini is more implementation dependent like it requires Java on both sides of its devices to making a remote method invocation. This thing makes Jini protocol independent. And also it doesn’t have support of the Eventing. On the other hand UPnP is protocol dependent and not implementation dependent. UPnP uses protocols which run on TCP/IP can be implemented in any language. Also UPnP has support for eventing so that interested devices can subscribe to the services of other devices and can get event notifications when service state changes. This thing gives advantage to UPnP over Jini.

As we are implementing the coordination framework for mobile devices we have also research on programming styles for mobile devices. For this purpose OSGi and Prism-MW frameworks has been investigated.

OSGi architecture framework has following problems:
For the devices to operate on bundles (components), bundles’ software must be available [23]. For example for an device the device driver must be available in the form of bundle otherwise in distributed system this will be very difficult to operate on the device by other bundles.

While in Prism-MW framework, the components are connected before they are used through connector object.

So for UPnP device architecture devices should have the bundles provided before they can operate in OSGi framework but it is an issue which must be handled professional developers. In Prism-MW UPnP device architecture is easily mapped by implementing/connecting the components (computation parts) with devices.

4.2 Contributions

We have searched how we can coordinate between mobile devices. And we have found that UPnP has advantages among others.

For UPnP to be used on the mobile we have explore the Prism-MW framework. Because Prism-MW framework is style architecture used to overcome the complexity of large software on mobile devices. We have implemented UPnP protocol stack on top of the Prism-MW framework for the android platform. We have transformed the UPnP protocol stack into Prism-MW style components and connectors.

Internal communication between the components and connectors is done by Prism events and we also have written the Configuration classes which store the connectors and components settings.

We have provided the implementations for both client side and server side device applications.

We have made both server side and client side very generic so that any UPnP device and its services and control point can be implemented.

4.3 Future Work

A comparison can be made between UPnP implementation over Prism-MW and without Prism-MW framework for speed and reliability.
More research can be done on the following situations:

- How Multicast scenario will be handled in the case of its unavailability on the network routers.
- How the mobile device will be discovered in the case of long distance network.
5 References

8. http://csse.usc.edu/~softarch/Prism/, reviewed May 24, 2010
17. http://www.osgi.org/Main/HomePage
18. http://www.osgi.org/About/WhatIsOSGi
23. http://www.cs.stir.ac.uk/~kjt/research/match/resources/tutorial/HomeCare_Networks/OSGi/OSGi_5.html
24. http://www.cybergarage.org/cgi-bin/twiki/view/Main/CyberLinkForJava#CyberLinkForJavaAndroid
Appendix A

Device Description

“Some of these placeholders would be specified by a UPnP Forum working committee (colored red) or by a UPnP vendor (purple). For a non-standard device, all of these placeholders would be specified by a UPnP vendor.” [6]

The following XML template is being quoted from UPnP Architecture 1.0. Green elements will be used in XML file as they are appeared in this form. More information can be seen at UPnP Architecture 1.0 [6]

```xml
<?xml version="1.0"?>
<root xmlns="urn:schemas-upnp-org:device-1-0">
   <specVersion>
      <major>1</major>
      <minor>0</minor>
   </specVersion>
   <UDN>urn:uuid:UUID</UDN>
   <UPnP:modelURL>URL to model site</modelURL>
   <UPnP:manufacturer>Manufacturer's name</manufacturer>
   <UPnP:model>Model name</model>
   <UPnP:serialNumber>Serial number</serialNumber>
   <UPnP:UDN>Global unique ID</UDN>
   <UPnP:friendlyName>Friendly name</friendlyName>
   <UPnP:manufacturerURL>URL to manufacturer site</manufacturerURL>
   <UPnP:description>Long user-friendly title</description>
   <UPnP:service>
      <serviceType>urn:schemas-upnp-org:service:serviceType</serviceType>
      <serviceId>urn:schemas-upnp-org:serviceId:serviceId</serviceId>
      <SCPDURL>URL to service description</SCPDURL>
      <controlURL>URL for control</controlURL>
      <eventSubURL>URL for eventing</eventSubURL>
   </service>
</root>
```
Appendix B

Service Description

"Some of these placeholders would be specified by a UPnP Forum working committee (colored red) or by a UPnP vendor (purple). For a non-standard device, all of these placeholders would be specified by a UPnP vendor." [6]

The following xml template is being quoted from UPnP Architecture 1.0. Green elements will be used in xml file as they are appeared in this form. More information can be seen at UPnP Architecture 1.0 [6]