Automated software upgrade for reconfigurable mobile devices.

Zhang, Hui

Automated Software Upgrade for Reconfigurable Mobile Devices

Swansea University
Prifysgol Abertawe

Hui Zhang

College of Engineering

Swansea University

Submitted to Swansea University in fulfillment of the requirements for the degree of

Master of Philosophy (M.Phil)

December, 2011

Abstract

Due to rapid advancement of the mobile communication technologies and the existence of multi-standards, the demands for managing mobile device effectively to fulfill various functionalities are on the rise. With the development of firmware over-the-air (FOTA) technology, the reconfigurable feature of mobile device provided by Software Defined Radio (SDR) technology offers a new way to realize the multi-functionalities of mobile devices. Therefore, based on the reconfigurable feature provided by SDR devices and the FOTA technology standardized by Open Mobile Alliance Device Management (OMA DM) standard, in this thesis, a framework of Modulation Module Update (MMU) is proposed for upgrading the modulation module on the mobile device over-the-air.

As the modulation module is a vital module for radio devices, the upgrade or reconfiguration of modulation module can be extremely helpful for the realization of multi-functionalities. Therefore, in this thesis, the management object for updating modulation module are defined based on OMA DM standard, and three operation phases are defined in this framework as well. Compared with the original framework provide by OMA DM standard, the MMU framework is significantly improved from both the logical structure design aspect and the operation mechanism design aspect. The download operation of the firmware upgrade is optimized by the download mechanism proposed in this thesis, which is an over-the-air (OTA) software download mechanism combining the centralized download method and the decentralized download method for different circumstances. An important module named Central Information Controller (CIC) is also proposed to realize the control of the decentralized download method.

The implementation of the MMU is achieved in this thesis as well. The server side and the client side of the framework as well as the CIC are simulated to test and verify the framework we proposed for the modulation module OTA upgrade.

Declarations and Statements

DECLARATION

This work has not previously been accepted in substance for any degree and is not being concurrently submitted in candidature for any degree.

Signed... ... (candidate)

Data.....14/.12/.2011.....

STATEMENT 1

This thesis is the result of my own investigations, except where otherwise stated. Where correction services have been used, the extent and nature of the correction is clearly marked in a footnote(s).

Other sources are acknowledged by footnotes giving explicit references. A bibliography is appended.

Signed... ... (candidate)

Data.....14/.12/.2011.....

STATEMENT 2

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organizations.

Signed... ... (candidate)

Data.....14/.12/.2011.....

Table of Contents

Acknowledgement

First of all, I want to thank Dr. Xinheng Wang, my supervisor, for his friendly and patient supervision and many inspiring discussions.

I am also grateful to all my friends, for their support in any respect during the completion of my study.

Finally I would like to give my special thanks to my parents. Without their support it would not have been possible for me to study in the UK.

List of Figures

List of Tables

List of Abbreviations

ACL	Access Control List
AMF	Airborne, Maritime Fixed
ASIC	Application-specific Integrated Circuit
ATC	Air Traffic Services
CDMA	Code Division Multiple access
CIC	Central Information Controller
DM Tree	Device Management Tree
DSP	Digital Signal Processor
EDGE	Enhanced Data rates for GSM Evolution
EMS	Emergency Medical Service
FOTA	Firmware Over The Air
FPGA	Field Programmable Gate Array
FUMO	Firmware Update Management Object
GPP	General Purpose Processor
GPRS	General Packet Radio Service
GPS	Global Positioning System
GSM	Global System for Mobile Communications
HMAC	Hash-based Message Authentication Code
IEEE	Institute of Electrical and Electronic Engineers
ITS	Intelligent Transportation Systems
MO	Management Object
MMU	Modulation Module Update
NIC	Network Interface Card

OMA	Open Mobile Alliance
OMA DM	Open Mobile Alliance Device Management
OTA	Over-The-Air
SDR	Software Defined Radio
SoC	System on Chip
WCDMA	Wideband Code Division Multiple Access
WLAN	Wireless Local Area Network
UMTS	Universal Mobile Telecommunications System
URL	Uniform/Universal Resource Locator

List of Publications

Published

H. Zhang, H. X. Wang, and M. Iqbal, "An OMA DM Based Framework for Updating Modulation Module for Mobile Devices", International Journal of Adaptive, Resilient and Autonomic Systems, 2(3), pp. 13-23, 2011.

M. Iqbal, H. X. Wang, and H. Zhang, "Load-Balanced Multiple Gateway Enabled Wireless Mesh Network for Applications in Emergency and Disaster Recovery", International Journal of Adaptive, Resilient and Autonomic Systems, 2(4), pp. 35-52, 2011.

Submitted

X. H. Wang, and H. Zhang, "Development and Implementation of a Decentralized Download Method for Updating Firmware on Wireless Devices", EURASIP Journal on Wireless Communications and Networking.

1. Introduction

1.1 Overview

Nowadays, multiple standards have co-existed in the world mobile communication environment and mobile wireless technology has gained tremendous popularity due to its ability to provide ubiquitous information access to users on the move [1][2]. Consequently, the demands for more functionalities and higher intelligence of the mobile devices are on the rise. Traditional radio devices are primarily based on hardware, which enables the devices to be modified through physical intervention. With the requirement of the versatile functionalities, this significantly leads to higher cost and reduces the flexibility of the devices [3].

The emergence of software defined radio (SDR) technology has significantly changed the situation. The physical layer functions of a SDR device are software defined. It means that the operation or behaviour of the device can be changed only by changing the software [4]. This provides a chance to implement the multi-mode, multi-band or multi-functional functionality by upgrading software. In addition, the upgrade of SDR device can be achieved by downloading software to realize a series of functionalities, such as fixing bugs, offering new services, changing air interface, etc.

Although the download and upgrade of software for SDR devices can be realized by several methods, over the air (OTA) software upgrade shows more advantages than others in the long run [5]. Today, OTA upgrade of software is not restricted to application software any more. With the development of firmware over the air (FOTA) technology, firmware can also be updated by OTA method. Actually, FOTA technology conspicuously improves the development of the technologies for upgrading SDR devices. Nowadays, open mobile alliance device management (OMA DM) standard has become a universal standard for FOTA, especially the FOTA update specified by FUMO [6]. Therefore, it is also adopted for the firmware upgrade of SDR based

devices.

However, the OMA DM protocol and the FUMO defined by OMA DM Standard for firmware upgrade only provide a common specification to satisfy most of the mobile devices. They do not consider the specified design of management object and management operations which are directly related to the firmware upgrade operation.

Moreover, the firmware download method applied by FUMO is limited in specific circumstances. At present, the traditional client/server architecture used for software download is the well-known means of distributing software, which is also applied by OMA DM standard. Therefore, the server is the only device that can provide firmware to mobile devices for update. Although this centralized download method is the most traditional way to download firmware, it can hardly carry out the firmware download for a large number of mobile devices in a limited timeframe only with a few central servers. Thus, when there are lots of devices in a specific area, such as sensors in a wireless sensor network, waiting for software download at the same time, the traditional method exposes its limitations.

Accordingly, the research on the firmware upgrade of SDR devices based on the FOTA technology and the OMA DM Standard is of great significance. In this thesis, we proposed the MMU framework to solve the above problems.

1.2 Contributions

In this thesis, a framework named MMU is proposed, which is designed and implemented for updating the modulation module OTA on the mobile device based on OMA DM. The detailed contributions are organized as follows.

- The logical design of MMU is presented in this thesis. We elaborate the overall design of MMU in Chapter 3, including the design of management architecture, the management object and the management operations. Although the design of MMU

is based on the basic framework provided by OMA DM standard, the logical structure and the operation mechanism is significantly improved in MMU. Compared with the original framework of OMA DM standard, the download operation is optimized by the download mechanism we designed, which is an OTA software download mechanism combining the centralized download method and the decentralized download method for different circumstances. This download mechanism is more efficient than the regular download mechanism provided by FUMO, especially when there are a lot of mobile devices waiting for the firmware to be updated. Besides, the FUMO protocol only defines the logical flow of firmware update, which does not specify the specific operations or the detailed message exchange flow between the server and clients. Therefore, in the design of MMU, the management operations of modulation module upgrade are exclusively extended and designed in detail. The firmware upgrade operations we proposed are divided into three phases, which are initialization phase, download phase and update phase, and each phase is designed in detail. In addition, the nodes for the management object of modulation module are also designed in detail, including the type, format, name, size, as well as the Access Control List (ACL) property of each node.

- MMU is implemented in this thesis. The detailed implementation of MMU is elaborated in Chapter 4 from a coding point of view. MMU is implemented by three parts which are the DM Server side, Agents in mobile devices and the CIC. The DM Server is implemented by the traditional five layer structure which includes User Interaction Layer, DM Protocol Layer, SyncML Representation Layer, Transport Layer, and Network Layer. However, we have implemented the DM Protocol Layer and SyncML Representation Layer in a more flexible way. The implementation of these layers in MMU is based on relatively independent classes which can be seen as the different components in these layers. These components realized by different classes are respectively related to the operation commands or the elements in syncML Messages, which can be combined into different operations

or different syncML messages based on different requirements. Besides, the detailed implementation of the clients in mobile devices is also realized, including DM Tree, DM Agent, FUMO Agent, and DL Agent. The implementation of these client agents are also component based. Each part can be seen as a component in the mobile device, which is relatively independent to each other. Each component provides interfaces to other components and invokes the interfaces provided by other components to realize the communications between different components. The change of each component will have less influence on the other components, and new functionalities can be added by adding new component. This implementation method to design MMU is more effective than traditional implementation method and makes the framework more flexible. Moreover, the implementation of CIC is achieved as well. CIC provides the control functions for the decentralized download method we proposed. As we mentioned in the last section, the traditional download method is deficient in some situations. CIC we proposed plays a vital role to solve this problem.

- The server side and the client side of the framework as well as the CIC are simulated to test and verify the framework proposed, and the test and verification results show the feasibility and accuracy of MMU.

1.3 Outline

The rest of the thesis is organized as follows: Background of SDR, FOTA, and OMA DM standard is presented in Chapter 2. The detailed logical design of MMU framework is elaborated in Chapter 3. The implementation of MMU framework is presented in Chapter 4. In Chapter 5, the test and verification results are presented. Finally, the conclusion and future works are presented in Chapter 6.

2. Background

2.1 Software Defined Radio

2.1.1 Introduction of SDR

Multiple standards have become a common phenomenon in today's mobile device networking environment. Many mobile communication standards co-exist in the world mobile communication environment [7], such as GSM, IEEE 802.11 standards for WLAN, as well as the 3G mobile technology standards, etc. Furthermore, the next-generation mobile standards are on their way.

Table 2.1 presents the features of several existing standards. Moreover, different standards are employed in different regions or countries. Take 3G mobile technology standards as an example, UMTS system is primarily used in Europe, Japan, and China with different radio interfaces while CDMA2000 system is applied in North America and South Korea [8].

The diversity of the mobile technology standards makes an increasing requirement on the versatility of mobile devices. Traditional radio devices are primarily based on hardware. It means these devices can only be modified through physical intervention, which significantly reduces the flexibility of the devices and leads to high cost. For years experts had been looking for an effective way to solve this problem. An ideal scenario is that the mobile device is able to download the software to realize the fully-functional, multi-band, multi-mode functionality and overcome the global roaming limitations imposed by today's multiple air interface standards [9]. The types of the software that can be downloaded and updated could not be restricted to application software and more functionality could be offered to mobile devices by software. Then, the term of "SDR" was proposed.

The SDR Forum, working in collaboration with the IEEE P1900.1 group, has worked to

establish a definition of SDR [10]. The definition of SDR is as follow:

Radio in which some or all of the physical layer functions are software defined [11].

Table 2.1 Mobile Communication Standards

Specification	Ultra-wideband	802.11 a/b/g	802.11 n	Wireless Broadband (WiBro)	3G LTE (cellular WAN)	Digital Video Broadcasting-Handheld	Digital Video Broadcasting -Terrestrial
Application	High-speed local interconnect, wireless USB	Medium-speed LAN	High-speed LAN	Mobile wireless access	Mobile data/voice	Mobile TV	Mobile TV
Range	10 m	80 m	50-150 m	1-5 km	1+ km	-------	-------
Rate	480 Mbps	11 Mbps (b), 54 Mbps (a/g)	100-600 Mbps	3-50 Mbps (downlink)	100 Mbps (downlink)	384 Kbps	7 Mbps
Frequency	3.1-10.6 GHz	2.45/5.8 GHz	2.45/5.8 GHz	2-6/2, 3 GHz	1.25/2.2/5/10/20 GHz	0.8 MHZ, 1.6 GHz	0.8 MHZ, 1.6 GHz
Modulation	Orthogonal frequency division multiplexing	Direct-sequence spread spectrum, orthogonal frequency division multiplexing	Orthogonal frequency division multiplexing	Orthogonal frequency division multiplexing	Orthogonal frequency division multiplexing, etc	Orthogonal frequency division multiplexing	Orthogonal frequency division multiplexing, coded orthogonal frequency division multiplexing

Traditional radio devices can only be modified through physical intervention. By contrast, SDR technology provides an efficient solution to these problems. An SDR device can change transmitter and receiver characteristics such as modulation type, radiated power, wideband and narrowband operations, and air interfaces only by changing software [4]. This technology provides a chance to implement the multi-mode, multi-band or multi-functional functionality by only upgrading the software.

A series of hardware and software are defined by SDR, where physical layer processing is implemented through modifiable software or firmware operating on programmable processing technologies [12][13]. There are many kinds of these devices, including FPGA, DSP, GPP, SoC or other application specific programmable processors.

2.1.2 Reconfiguration of SDR

The reconfiguration of SDR based devices refers to the operation of software download that is for transferring the reconfiguration data into the devices. The data needs to have

the capability to change the operation or behaviour of the SDR device. The type of the reconfiguration data is as follows [14]:

- A piece of software, such as DSP algorithm, operating system device driver, application software, etc.
- Parameters, such as DSP processing parameters, etc.
- An FPGA configuration bitstream.
- Parameter values for parameterized ASICs.

The reconfiguration of the device could be at any layer of mobile devices from the application to hardware. The download and update operations may be applied to a series of software including application software, system software, radio software, etc. By the support of the hardware and software architecture provided by SDR device [15][16], the reconfiguration operation can realize a series of functionalities, such as fixing bugs, offering new services, changing air interface, etc. Figure 2.1 shows the reconfiguration feature of SDR devices.

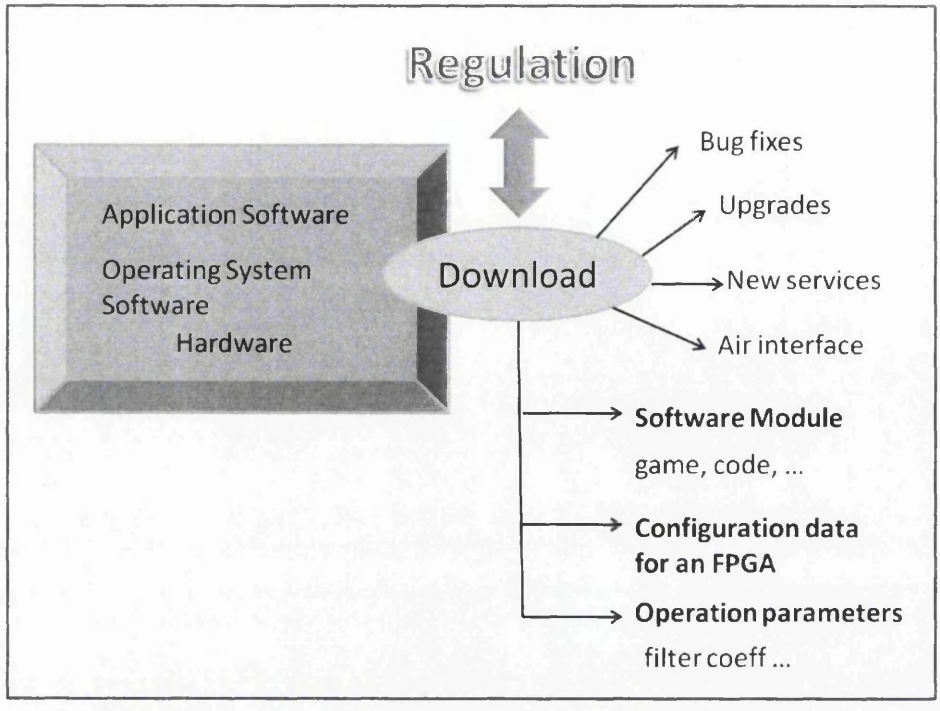

Figure 2.1 Reconfiguration Feature of SDR Devices

2.1.3 Development of SDR

SDR technologies as an effective way to solve the existing problems of current mobile devices have been significantly developed. The adoption of SDR technologies has spread to various fields. A survey by SDR Forum shows the market segmentation of SDR in Figure 2.2 [17], which presents the application of SDR technologies in different domains, such as in military, civilian and commercial domains.

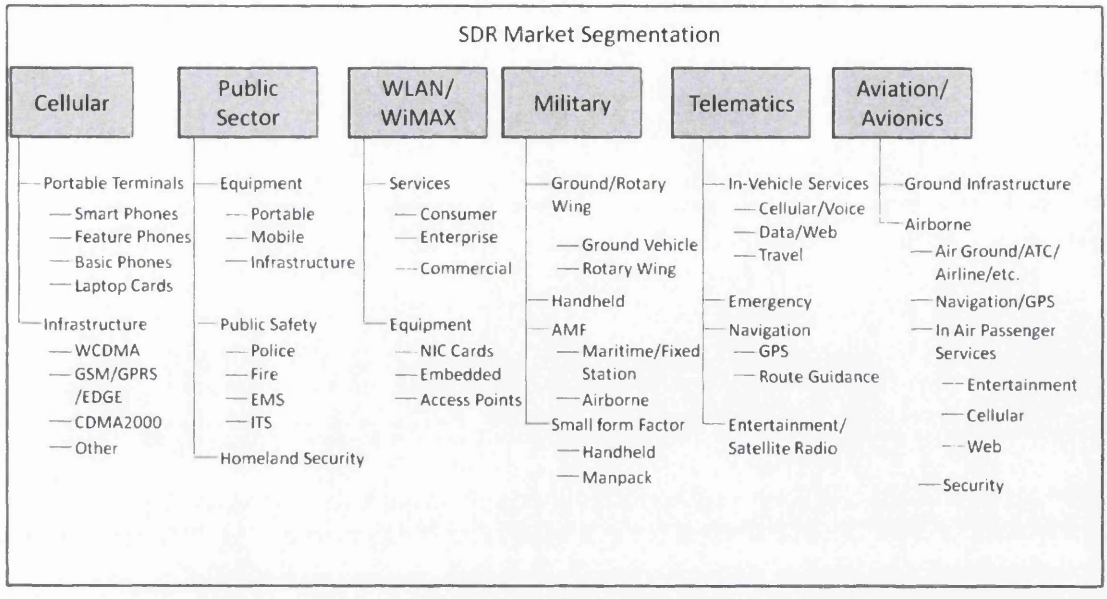

Figure 2.2 SDR Market segmentation

The reconfiguration feature provided by SDR technologies is one of the key reasons that makes it developed rapidly in the past few years. The research on SDR technologies is still on its way. The software upgrade on SDR based devices as a key point of the SDR has been improved by the evolution of the hardware technology. The development of the performance of hardware such as DSP and FPGA improves the hardware conditions for the dynamic reconfiguration of SDR devices [18][19].

2.1.4 Summary

This section introduces the generation and development of SDR technology and the reconfiguration feature of SDR devices. Actually, in this thesis, our research focuses on the software upgrade on mobile devices, which lays particular emphasis on the management and realization of the software download and update between server side and mobile devices. Although our research is not directly related to the lower layer development of SDR system, note that the reconfigurable feature provided by SDR technology is the precondition of our research.

2.2 Firmware Over-the-air Upgrade

2.2.1 Firmware Upgrade Technology

The reconfiguration of the SDR based mobile devices related to the lower layer software is usually related to firmware. In electronic systems and computing, firmware is a term often used to denote the fixed, usually rather small, programs and/or data structures that internally control various electronic devices. Actually, the boundary between firmware and software is not quite strict [25]. As we all know, the upgrade of firmware especially the radio software in SDR devices can realize the multi-mode, even multi-functional functionalities. Not only that, it can also supply mobile device with additional capabilities as well as fix software bugs or deficiencies of the software in devices. All of these functionalities lead to two vital points of the reconfiguration, firmware download and update.

With the development of the mobile communication technologies, the methods of the software download to mobile devices have significantly improved and become various. The OTA software download technology has been applied from the initial application software download to the firmware download [5][21]. Today, FOTA technology has been widely used due to its advantages. Table 2 show the comparison between FOTA and the other two download methods [22].

Table 2.2 Comparison of Different Download Methods

	Firmware OTA	Firmware (cabled)	Point-of-sale
Use cases	Batch updates of critical bug fixes, Push new features & functionalities	End-user maintenance, Large delta upgrades	Rectify known issues, Just-in-time updating of handsets via kiosk or through manufacturer update site
Applicable to	Most suited to operators who can integrate OTA within existing support environments. Also used by manufactures but end-user must bear airtime cost	Large mobile device manufactures who bundle necessary cable-ware with device.	All
Pros	Anytime, anywhere	Fast and low cost	Fast, low cost. Process can be removed from end-user if required
Cons	Impact on user experience	Requires necessary cable and Internet access	Consumes "sales person's" time
Standards	OMA, proprietary	Proprietary	OMA, proprietary, etc

The three methods for firmware download in Table 2.2 have their own advantages for different requirements. The obvious advantage of FOTA is that firmware in mobile devices can be updated wirelessly, anywhere, and any time. The FOTA technology allows the creation of the smallest possible firmware updates, which are then highly compressed and transmitted over-the-air to mobile devices, and then decompressed and applied on the device firmware [23]. This technology has been seen as the most cost-effective solution to update the firmware for those devices which are already in use. In recent years, the adoption of FOTA technology in handsets has increased rapidly. A survey by Wireless Informatics Forum presents the growth of the FOTA handsets [24], which is shown in Figure 2.3. As for the multi-mode, multi-band functionalities of SDR devices which may require the firmware upgrade anytime, anywhere, the function provided by FOTA technology is exactly what they need.

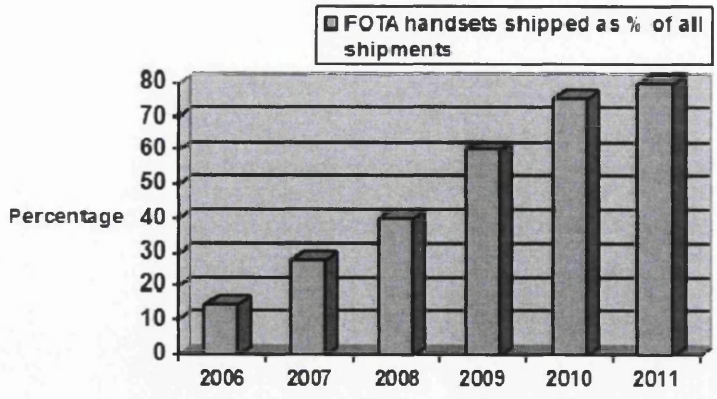

Figure 2.3 Growth of FOTA Handsets

2.2.2 Firmware Over-the-air

FOTA technology allows the firmware to be updated over-the-air, which benefits a lot of groups, such as the manufacturers, operators as well as the end users [25]. Therefore, the firmware update packages need to be downloaded to the mobile device to achieve the update for different functions. Figure 2.4 shows a general architecture of the FOTA update.

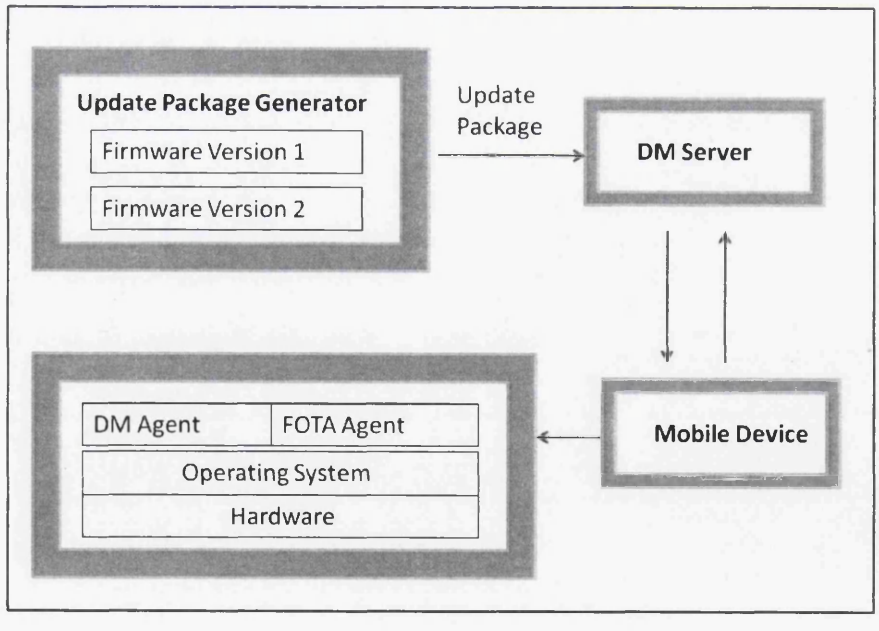

Figure 2.4 General architecture of the FOTA

A general implementation of FOTA mainly contains three parts [26], which are the update package generator, DM Server, the agents in mobile devices.

The Update Package Generator is for generating the update package. As we know, the download of firmware is to download the updating programs or parameters which are used for the update process. These programs or parameters are usually provided by the manufactories of service providers. For some kinds of firmware, the update package for download is actually not the programs of the intended target version. It is a package generated by specific algorithm, which is comprised of a set of data instructions for transforming the software image form the source version to the intended target version [23].

DM Server is the device management server used to send update packages to mobile devices and manage the FOTA download and update operations. OMA DM standard has been considered as the most widely applied standard for FOTA. This will be introduced in later sections.

Agents in mobile devices are the clients responsible for realizing the firmware download and update operations in mobile devices. DM Agent is responsible for managing the communications and the exchange of management messages between the devices and DM Server. FOTA Agent is for managing the operations of download and update of firmware and processing the downloaded packages. Generally, the FOTA download and update procedure is as follows. First, the Update Package Generator creates the update packages. Then, the update package is downloaded to the mobile devices from DM Server under the management of DM Server. After that, the downloaded update package is processed by FOTA Agent to achieve the update operation.

2.2.3 Research on FOTA of SDR Devices

The research on FOTA has been for a long time and the adoption of FOTA technology in mobile devices has increased rapidly. The adoption of FOTA technology in SDR has developed for years as well. Downloading firmware packages to mobile devices to realize the multi-functionalities is one of the major applications of FOTA technology for SDR based devices. SDR Forum proposed a firmware OTA download flow which is shown in Figure 2.5.

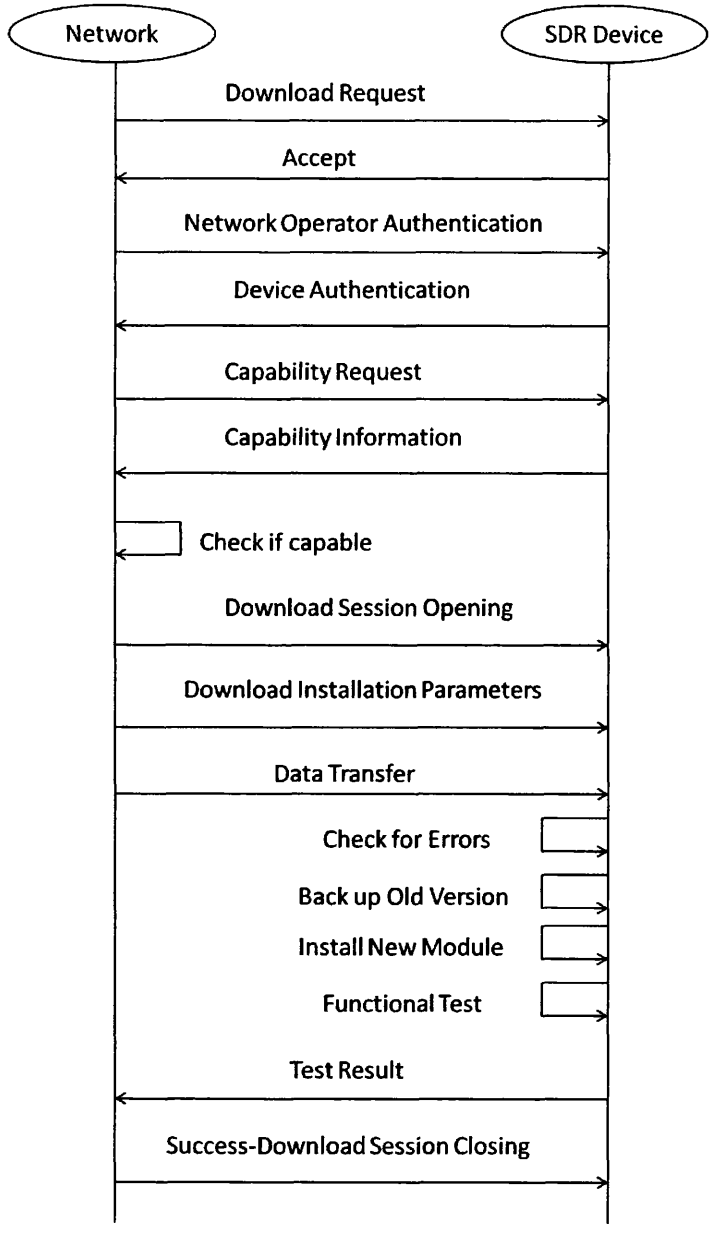

Figure 2.5 Firmware Over-the-air Download Flow

Figure 2.5 shows a download operation procedure initiated by the network. The procedure includes the following steps:

1. Initiation.

2. Authentication: to identify the authentication of both sides of communication to guarantee the identity legitimacy.

3. Capability exchange: to check if the device is capable of downloading update packages.

4. Data transfer: to download the data to mobile devices and check errors.

5. Installation and Test: to process the downloaded data, install the update, and verify correct functionality.

6. Closing.

The procedure of download operation proposed by SDR forum specifies the flow to realize the FOTA download operation for SDR based devices. Base on the flow, there has been a series of further research on FOTA download and update.

In [27], a framework Smart Box Management (SBM) was proposed. It is an end-to-end remote management framework for Internet enabled device. The SBM server provides the basic set of services to the SBM clients over the Internet, such as remote activation, remote configuration, dynamic updates (downloads), and device diagnostic uploads, based on a set of protocols like device registration protocol, configuration protocol, upload protocol, and download protocol. SBM realizes these management functions by using their own protocols defined for the system which limits the generality of the system. The specified protocols can hardly be widely used for other systems.

In [28] the authors presented the work on a demonstration platform for a SDR proof-of-concept and how OMA DM protocol and Functional Description Language

can be used to support RAT reconfiguration on SDR terminals. This paper introduced the OMA DM standard to SDR firmware over-the-air download.

As we know, the firmware upgrade of SDR device applies the FOTA technology. Nowadays, FOTA has been supported by the Mobile Device Management technologies which are standardized by the OMA. The FUMO protocol in OMA DM Standard has been the most widely used protocol for FOTA technologies. Today, the OMA DM Standard is applied for the firmware download as well.

2.2.4 Summary

In this session, we compare the current firmware upgrade technologies and explain the advantages of firmware over-the-air upgrade for SDR devices. We also analyse former researches which have been done for the FOTA upgrade of SDR devices. Today, FOTA technology has been adopted by the upgrade of SDR devices. Therefore, OMA DM standard applied by FOTA has also been considered as the standard for SDR devices. In next section, we will introduce the OMA DM Standard and explain how it works as the foundation of our research.

2.3 OMA Device Management

2.3.1 Open Mobile Alliance

Open Mobile Alliance is founded in 2002, which was formed by WAP Forum and Open Mobile Architecture [29]. Later, a number of organizations that committed to promote the standardization of mobile business. For example, Location Interoperability Forum (LIF), SyncML Forum, MMS Interoperability Group (MMS-IOP), and Wireless Village, have joined the OMA. At present, there are more than 400 members of OMA, including mobile operators, equipment suppliers, and software provider, etc. The aim of OMA is to establish a uniform and global industry standard for mobile services.

So far, OMA has developed a series of uniform standards, most of which have been widely deployed, such as OMA Multimedia Messaging Service, OMA Download over the air, OMA Management, etc.

2.3.2 Device Management Standard

The earliest standard for mobile device management was made by SyncML Forum, which is syncML Device Management V1.1.1. This standard supports the functions of parameters configuration management, device firmware update and device data collection. Afterwards, SyncML joined OMA. In 2003, OMA released the OMA Device Management V1.1.2 based on the syncML Device Management standard, which is known as the OMA DM V1.1.2. At present, the standard version of OMA device management standard is OMA DM V1.2 .

The definition of Device Management (DM) given by OMA means the third party can configure or update the mobile devices instead of terminal users. The third party is the organizations who have the permission to manage the mobile devices as well as terminal users. Based on device management standard, third party can remotely run various management operations of mobile devices, such as configuration, fault monitor and diagnosis, software or firmware update, etc [30][31].

In fact, OMA DM includes a series of protocols to realize the various functions of device management. OMA DM standard provides a framework, which provides the basic functions for mobile device management, such as the flow of the management sessions and the format of the management messages [32]. Then, based on the basic framework, other functions can be added to it by other protocols, such as FUMO protocol, DiagMon protocol for mobile device diagnosis and monitor, etc.

2.3.3 Device Management Mechanism

2.3.3.1 DM Tree

To ensure the generality, Management Object (MO) is defined by OMA DM standard [33]. The management of the mobile device can be achieved by managing the management objects. The benefit of the definition of management object is that the generality of the protocol could not be impacted by the difference of the formats or actions of the objects. There is a tree structure of these management objects, which is known as DM Tree [34]. Each device supporting OMA DM Standard has a DM Tree. Each node in a DM Tree has a unique Uniform Resource Identifier (URI) [35]. The structure of DM Tree is shown in Figure 2.6.

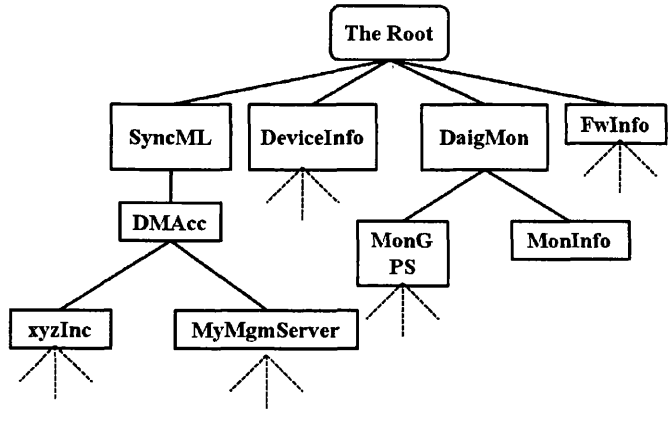

Figure 2.6 Structure of a DM Tree

OMA DM protocol defines a series of management commands which can be used for the operations on tree nodes. Some of the tree nodes could represent the information of the software or firmware on mobile devices. Nodes are divided into interior node and leaf node. An interior node can have child nodes but a leaf node can not. OMA DM protocol defines that an interior node can be expanded by management commands. Besides, each node has a series of properties which are used to define the specified features of it. The basic properties of nodes are shown in Table 2.3 [36].

Table 2.3 DM Tree Properties

Property Name	Requirement	Usage
ACL	Must	Access Control List
Format	Must	Data format of the current node value
Name	Must	Name of the node
Size	Must	Current size of the node value
Title	May	A string that provides the information about the node
TStamp	May	The time of the last change in value of the node
Type	May	The type of the data in programming languages

2.3.3.2 Package and Message

OMA DM protocol requires that the management operations should be sent through syncML packages [37]. When one syncML package is too large to be sent at a time, it can be divided into several syncML messages. The way of the communication between server and mobile devices is request/response. Due to the limited resource of mobile devices, server is not allowed to send a new message to mobile device before the former one has been successfully processed. The procedure of message exchange is shown in Figure 2.7.

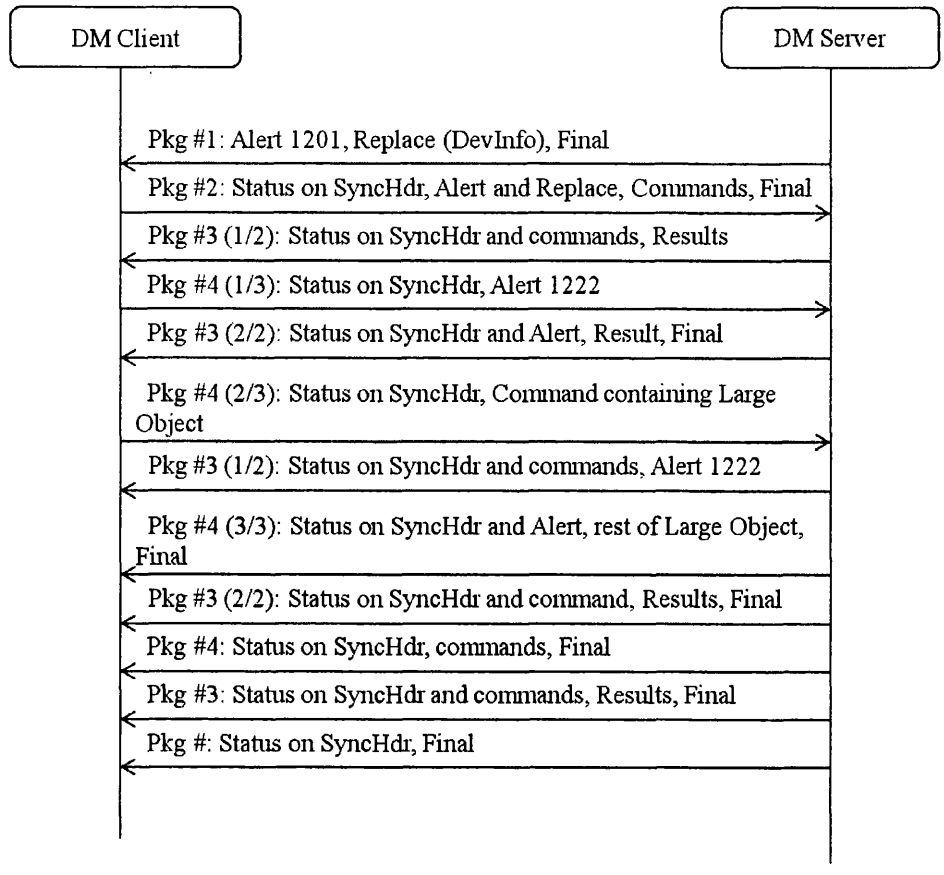

Figure 2.7 Exchange of Operation Message

2.3.3.3 Format of Message

The management operation between server and mobile device is based on messages. SyncML Representation protocol specifies the format of the syncML messages.

1. SyncML Message

OMA DM protocol defines that the container of operation messages is SyncML. A syncML message is composed of two parts, SyncHdr and SyncBody [38]. SyncHdr is used to record the basic information of the message. SyncBody is used to record the main content of the management operation. Both SyncHdr and SyncBody contain a series of elements. Figure 2.8 shows the basic structure of a syncML message. Figure 2.9 is a sample of syncML message.

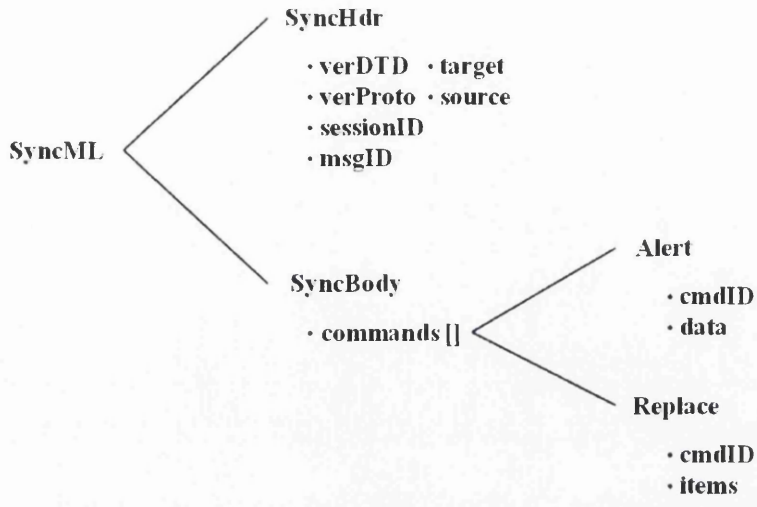

Figure 2.8 Structure of SyncML

```
- <SyncML xmlns="SYNCML:SYNCML1.2">
  - <SyncHdr>
      <VerDTD>1.2</VerDTD>
      <VerProto>DM/1.2</VerProto>
      <SessionID>1</SessionID>
      <MsgID>2</MsgID>
    - <Target>
        <LocURI>XXXX</LocURI>
      </Target>
    - <Source>
        <LocURI>XXXXXX</LocURI>
      </Source>
    </SyncHdr>
  - <SyncBody>
    - <Status>
        <MsgRef>2</MsgRef>
        <CmdID>1</CmdID>
        <Cmd>SyncHdr</Cmd>
        <Data>212</Data>
      </Status>
      <!-- XXXXXXX -->
    - <Replace>
        <CmdID>2</CmdID>
      - <Meta>
          <Format xmlns="syncml:metinf">b64</Format>
          <Type xmlns="syncml:metinf">application/XXXXXX</Type>
        </Meta>
      - <Item>
        - <Target>
            <LocURI>./data</LocURI>
          </Target>
        - <Data>
            <!-- Base64-coded XXX file -->
          </Data>
        </Item>
      </Replace>
      <Final />
    </SyncBody>
  </SyncML>
```

Figure 2.9 Sample of SyncML Message

2. Operation Command

Operation command is the vital part for the operation management. Operation commands are recorded in syncML message and sent to mobile devices to realize the management operation. Table 2.4 shows some common commands in device management.

Table 2.4 Common Commands in Device Management

Command	Usage	Position
Add	Add nodes	Contained in SyncBody
Delete	Delete nodes	Contained in SyncBody
Get	Get value of node	Contained in SyncBody
Replace	Replace value of node	Contained in SyncBody
Exec	Execute the operation represented by node	Contained in SyncBody

2.3.3.4 Operation Procedure

Generally, the operation procedure has two phases, Setup Phase and Management Phase [39]. Setup Phase is to exchange the authentication information and device information exchange. Management Phase is for the specified management operations, which can be repeated several times. The Setup Phase and Management Phase are shown in Figure 2.10 and Figure 2.11 respectively.

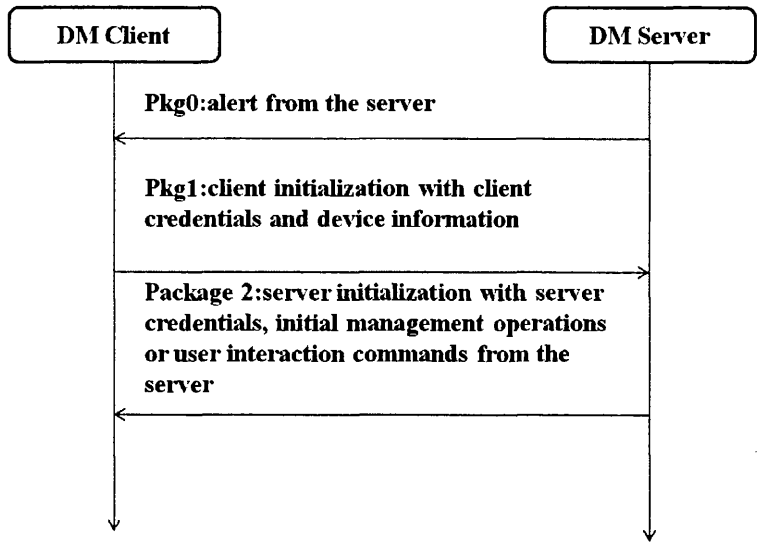

Figure 2.10 Setup Phase

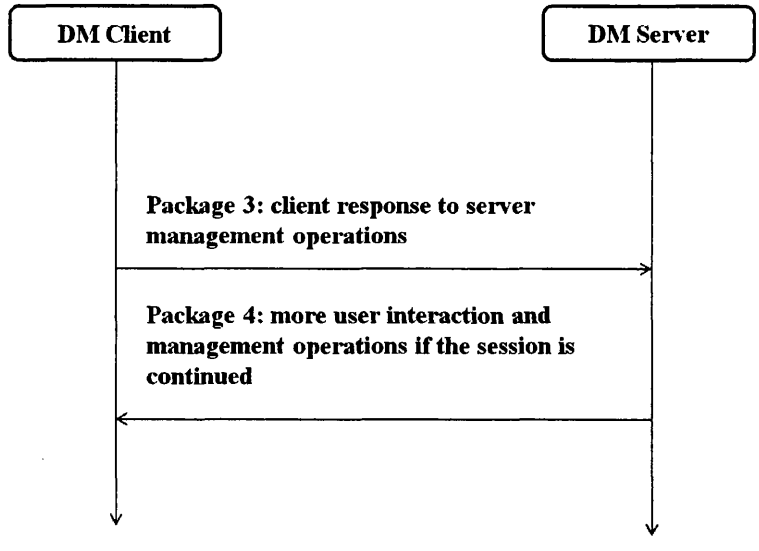

Figure 2.11 Management Phase

2.3.3.5 Security

For the data security consideration, the management sessions based on OMA DM protocol are all encrypted. The security of data is very important because many

information are included in management messages. Therefore, rules have been made in OMA DM protocol to ensure the security of data in management messages.

Verification

Before the DM Server and mobile device start the management operations, verification is necessary. The contents to be verified include:

1. Server ID, the unique identification of Server.

2. User Name, the unique identification of user.

3. Password, both the DM Server side and device side need to pass the password verification before the management operation session starts.

4. Nonce, used for calculating the Hash value to prevent from attack.

Integrity

The integrity of management message is realized by HMAC-MD5. The integrity verification of message is not necessary. But when the device side or server side requires for integrity verification, it could be realized by HMAC-MD5.

2.3.4 Firmware Update Management Object

FUMO is an OMA specification for updating the firmware of mobile devices over-the-air, which allows mobile devices to be updated over-the-air using the industry-standard protocol OMA DM. FUMO has been seen as the standard of firmware upgrade for mobile devices [6]. In fact, this is a protocol for firmware update in mobile devices based on the framework provided by Device Management protocol. Figure 2.12 shows the overall flow of the management operation mechanism for firmware update [40].

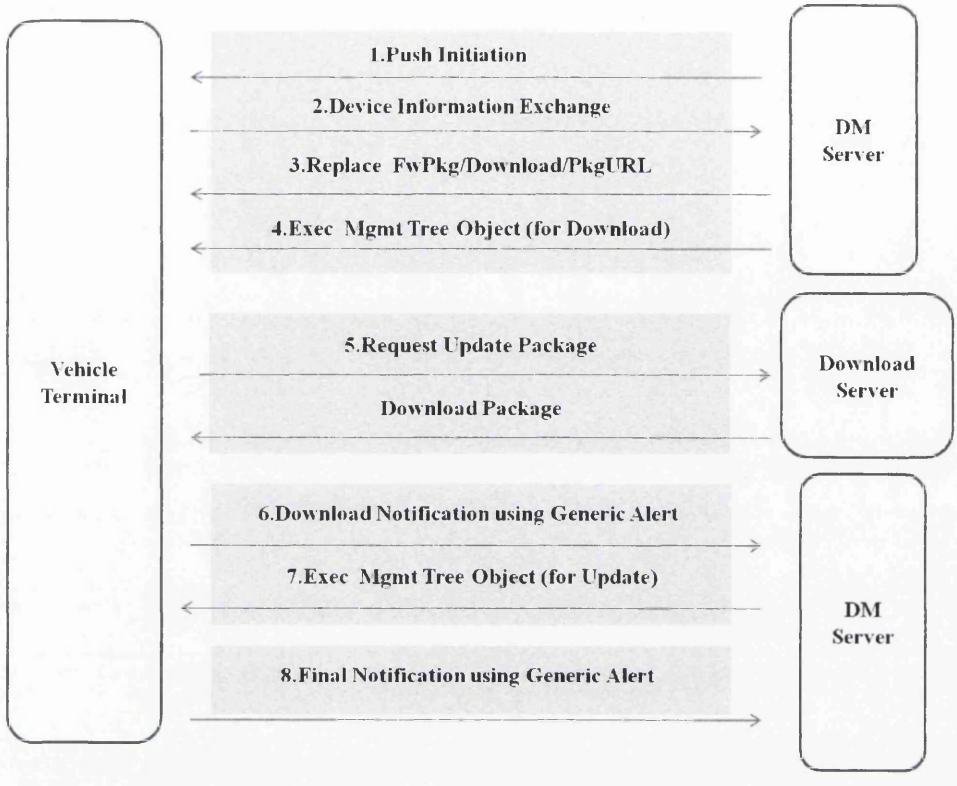

Figure 2.12 Flow of Firmware Update

The detailed steps of firmware update of FUMO are as follows:

1. DM Server sends notification to the mobile device in order to make the device connect to DM Server.
2. Mobile device connects to DM Server and starts the management session. Meanwhile, the mobile device sends client credential and device information to DM Server.
3. DM Server sends the URL information of the firmware to be downloaded to mobile device by using "Replace" command.
4. DM Server asks the mobile device to execute the download operation by using "Exec" command.
5. Mobile device starts to execute the download operation by using the appropriate way in different circumstances.
6. When mobile device finishes the download operation, it sends a message to notify DM Server that the download operation is finished by using "Generic Alert". Then

DM Server can start new management operations.

7. DM Server asks the mobile device to execute the update operation by using "Exec" command.
8. When mobile device finishes the update operation, it sends a message to tell DM Server that the update operation is finished by using "Generic Alert". Then DM Server can start new management operations.

2.3.5 Summary

In this session, we introduce the OMA DM standard and explain how it works as the foundation of our research. As we introduced in this session, OMA DM standard for firmware upgrade only provides a common framework to satisfy all kinds of firmware and all kinds of mobile devices. However, as the vital factor, the specified design of management object and management operations is not involved. In fact, the proper design of management object and efficient design of management operations have a huge impact on the firmware upgrade operations. Besides, as we mentioned in last chapter, the download mechanism adopted by FUMO is the traditional centralized download method which has the limitations in some circumstances. Therefore, in this thesis, we research these problems and propose our solutions by developing a decentralised download method and implementing the system using a component based mechanism. This will be introduced in next chapter.

3. Design of Modulation Module Update

3.1 Introduction

OMA DM Standard includes a series of protocols in order to specify the implementation of the functions for device management. The protocols of OMA DM Standard provide a framework for device management, which specifies the basic rules of device management, including the basic flows of the management session and the format of the management messages, etc. Based on the basic framework provided, other management functions can be added into the framework by other protocols in OMA DM Standard, such as FUMO, DiagMon, etc. FUMO protocol for the firmware update in mobile device has a lot of advantages and has been constantly improved.

As we all know, modulation module plays a significant role in radio devices, which is directly related to the operation or behavior of the radio device. As we mentioned in earlier chapters, the SDR technology offers us a chance to realize multi-functionalities based on its software defined feature on physical layer. Therefore, the reconfiguration of modulation module is a key point for the realization of multi-mode, multi-band, even multi-functionalities. Thus, in this thesis, the Modulation Module Upgrade (MMU) framework is proposed for the OTA upgrade of modulation module based on OMA DM standard.

This chapter presents the overall design of MMU, mainly including the design of management architecture, the management object and the management operations. The design of OTA software download mechanism for different circumstances is elaborated as well in this chapter.

3.2 Logical design of MMU

It is well known that the OMA DM standard plays a dominant role in mobile device management and firmware upgrade. It is a widely used standard that has been adopted

by almost all kinds of mobile devices. Therefore, the adoption of this standard in our framework can make the framework more universal and flexible. Although the design of MMU is based on the basic framework provided by OMA DM standard, the logical structure and the operation mechanism is significantly improved in MMU. Compared with the original framework of OMA DM standard, the download operation is optimized by the download mechanism we proposed, which is an OTA software download mechanism combining the centralized download method and the decentralized download method for different circumstances.

In this section, we introduce the logical design of the MMU. The specific design of MMU is as followed.

3.2.1 Management Architecture

MMU is a framework for updating the modulation module based on OMA DM. It is an improved framework based on the framework provided by OMA DM standard, and is designed for dealing with different download situations. As we mentioned in earlier chapters, the firmware download method applied by FUMO is limited in specific circumstances, and an effective download mechanism is proposed to solve this problem. The design of the MMU expands and optimizes the original framework of OMA DM standard. The modules for realizing the download mechanism are well designed and included in the MMU framework as well.

Figure 3.1 shows the management architecture of MMU we proposed. The architecture of MMU is mainly composed of five parts, which are DM Server, Content Server, Central Information Controller (CIC), the Agents and DM Tree in mobile devices. DM Server and DM Agent as well as DM Tree are used for achieving the basic functions required by OMA DM standard. Content Server and CIC are responsible for centralized download method and decentralized download method, respectively. The DL Agent and FUMO Agent aim to achieve the download and update operation based on the download mechanism and the management operation mechanism we proposed.

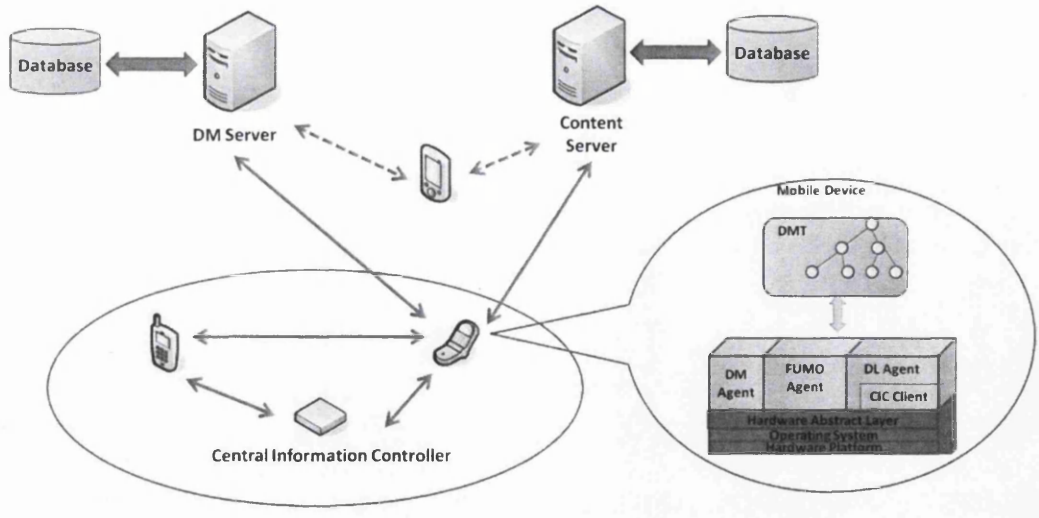

Figure 3.1 Management Architecture

DM Server: DM Server is the server-side implementation of the OMA DM protocol. Its task is to manage OMA DM compliant devices by using different management operations, e.g., provisioning, configuration of device, updating software, and fault management [41].

Content Server: Content Server is to provide and manage the software packages and multi-media contents for the software upgrades and bug fix of the mobile device. Content Server is mainly responsible for the centralized download method.

Mobile Devices: The device to be managed is composed of hardware components, software modules for managing the hardware components, and device management agent that performs software updates/management and firmware updates by connecting to the OMA DM server [42].

Device Management Tree (DMT): Each device that supports OMA DM contains a Management Tree. The Management Tree organizes all available management objects in the device in a hierarchical tree structure where all nodes can be uniquely addressed with a URI. DM Server realizes the management actions by manipulating the nodes in a device management tree [43].

DM Agent: DM Agent is a software component that resides in the mobile device. It is

used to process the messages received from the DM Server, including parsing messages from DM Server, interpreting OMA DM commands, and executing relevant actions in the device. In addition, the DM agent can also generate relevant responses and send them back to the DM Server [44].

FUMO Agent: FUMO Agent coexists with the DM Agent and DL Agent to provide firmware downloading and updating functions on the managed device. Generally, FUMO Agent and other special purpose agents can coexist to provide additional functions on a managed device [45][46].

DL Agent: DL Agent is a developed module proposed in this thesis. DL Agent coexisting with the DM Agent and FUMO Agent aims to manage the download operation in mobile devices by intellectively choosing the appropriate way to execute software download in different circumstances. The CIC Client included in this agent is used for communicating with CIC to realize the decentralized download operation.

CIC: CIC is the module proposed in this thesis to enable decentralized download and update operations. It is used for providing available software for mobile devices and managing the software download between mobile devices in the specified area.

The design of the logical operation flow of MMU is shown in Figure 3.2. The detailed steps of the flow are as follows:

- User who has the permission can log in the DM Server and set the management operation to manage the mobile devices.
- DM Server notifies the mobile device to start the software download operation.
- Mobile device can cancel the download operation by sending related message back to DM Server.
- If download operation is confirmed, mobile device will set the download

information. It will choose the appropriate download method and execute the download operation. Then, the mobile device will send the operation result back to DM Server.

- DM Server notifies the mobile device to start the software update operation.
- Mobile device can cancel the update operation by sending related message back to DM Server.
- If update operation confirmed, mobile device will execute the update operation and send the operation result back to DM Server.
- Management session closed.

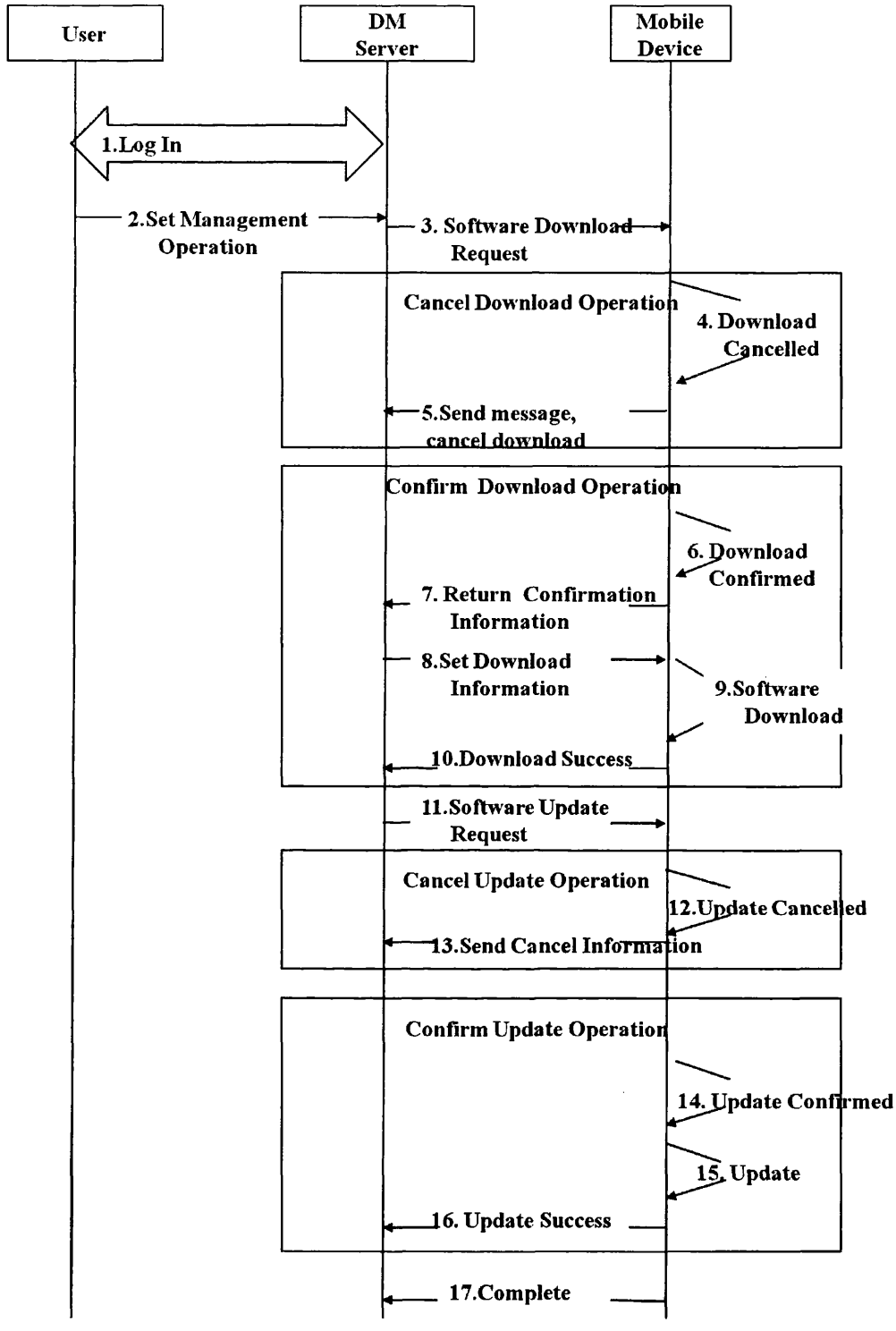

Figure 3.2 Flow of Management Operations

3.2.2 Management Object

Virtual tree structure built with the management objects is used for efficient device management. Through the DM Agent, DM Server may access individual node of the

virtual management tree, or, by accessing the parent node, it may reach all its child nodes [47][48]. Each firmware or software to be managed in a mobile device corresponds to a management object. The management object related to a certain firmware/software has a series of tree nodes, which contains the property information and the operation information of the firmware/software [40]. In this session, we define the firmware update management objects in the device management tree for the update of the modulation module based on FUMO.

Figure 3.3 shows the structure of the management object for the modulation module, which shows how the management object of the modulation module is defined, including its format, Access Control List (ACL) and Scope, and how they relate to other nodes. The management object is an extension of the DM Tree in the mobile device. Every managed software or firmware needs to be related to a management object in the DM Tree like this structure. DM Server performs the management operations through operating the nodes in the DM Tree.

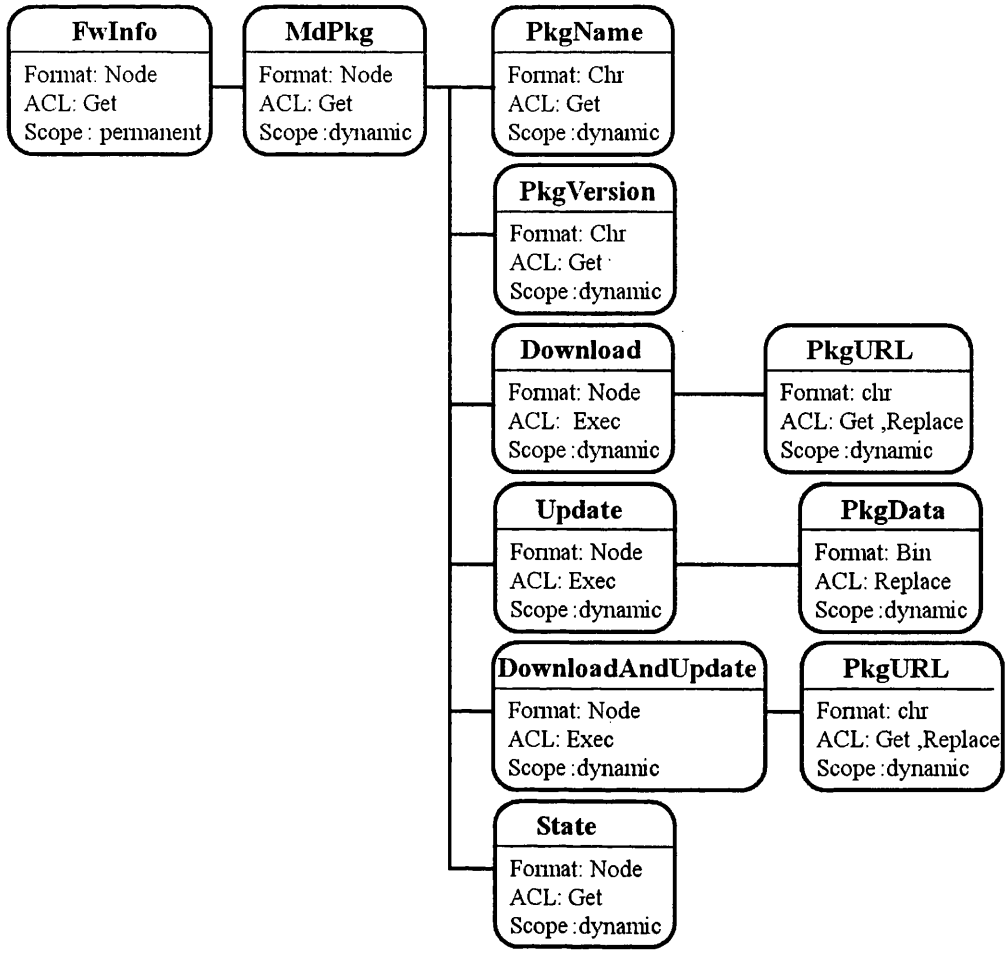

Figure 3.3 Structure of MO for Modulation Module

1. **FwInfo** is an interior node we designed to expand the root node, acting as the entrance of the firmware management object.

   Path: ./FwInfo

   ACL: Get, **Scope**: dynamic, **Format**: node.

2. **MdPkg** is an interior node acting as a placeholder for the modulation module's unique identifier. It stands for a software or firmware that can be managed by DM Server. Through accessing the children nodes of MdPkg, DM Server can realize a series of management operations on the mobile device.

   Path: ./FwInfo/MdPkg

   ACL: Get, **Scope**: dynamic, **Format**: node

3. **PkgName** is a leaf node that specifies the name associated with the modulation module. By using certain operation commands, DM Server can get the name

information of the modulation module.

Path: ./FwInfo/MdPkg/ PkgName

ACL: Get , **Scope**: dynamic, **Format**: chr

4. **PkgVersion** is a leaf node that specifies the version information associated with the modulation module. By using certain operation commands, DM Server can get the version information of the modulation module.

   Path: ./FwInfo/MdPkg/ PkgVersion

   ACL: Get , **Scope**: dynamic, **Format**: chr

5. **Download** is an interior node, which acts as the target of the "Exec" command. DM Server operates the node to perform the initialization of the modulation module download.

   Path: ./FwInfo/MdPkg/ Download

   ACL: Exec, **Scope**: dynamic, **Format**: node

6. **PkgURL** is a leaf node specifying the URL where the modulation module is located. In MMU, this node is used to record the address of the software to be downloaded when centralized method is applied. By using certain operation commands, DM Server can get or set the URL information of the modulation module.

   Path: ./FwInfo/MdPkg/ Download / PkgURL

   ACL: Get, Replace, **Scope**: dynamic, **Format**: chr

7. **Update** Update is an interior node, which acts as the target of the "Exec" command. DM Server operates the node to perform the update operation of the modulation module to the mobile device.

   Path: ./FwInfo/MdPkg/ Update

   ACL: Exec, **Scope**: dynamic, **Format**: node

8. **PkgData** is a leaf node acting as the target of a 'Replace' command when DM Server is used to directly provide the binary firmware update module.

   Path: ./FwInfo/MdPkg/ Update / PkgData

   ACL: Replace, **Scope**: dynamic, **Format**: Bin

9. **DownloadAndUpdate** is an interior node and the target of an 'Exec' command.

The update takes place as soon as the download finishes.

Path: ./FwInfo/MdPkg/ DownloadAndUpdate

ACL: Exec, **Scope**: dynamic, **Format**: node

10. **PkgURL** is a leaf node specifying the URL where the modulation module package to be downloaded locates. This node is used to record the address of the software to be downloaded when centralized method is applied as well. By using certain operation commands, DM Server can get or set the URL information of the modulation module.

    Path: ./FwInfo/MdPkg/ DownloadAndUpdate / PkgURL

    ACL: Get, Replace, **Scope**: dynamic, **Format**: chr

11. **State** is a leaf node that contains a value indicating the current state of the update of the modulation module. By using certain operation commands, DM Server can get the state information of the modulation module.

    Path: ./FwInfo/MdPkg/ State

    ACL: Get, **Scope**: dynamic, **Format**: node

3.3.3 Management Operation Mechanism

In order to realize the function of firmware update in mobile device, we designed the management operation mechanism for firmware update based on OMA DM and FUMO protocols.

Actually, the FUMO protocol only defines the logical flow of firmware update, which does not specify the specific operations or the detailed message exchange flow between the server and clients. Therefore, in this section, the management operations of modulation module upgrade are exclusively extended and designed in detail. The firmware upgrade mechanism we proposed are divided into three phase which are initialization phase, download phase and update phase. The three phases in this thesis are relatively independent. But the download phase and update phase are based on the phase before them. However, as long as the phases before them have been completed,

these two phases can be executed at any time. For example, the download phase can be executed right after the initialization phase or later. This kind of design makes the management operation to be more flexible. If any error happens to one phase, it just needs to re-execute this phase rather than the entire upgrade operation. The detailed design of each phases are as follow.

3.3.3.1 Initialization Phase

Initialization Phase is a phase to deploy the firmware information in the DM Tree. This phase needs to be executed only once. Once the related information exists in DM Tree, this phase can be skipped. The aim of initialization phase is to add a series of nodes related to the management object of modulation module to the DM Tree. DM Server can realize the management operation, such as the download operation and the update operation, by accessing these nodes.

At the beginning of the initialization phase, the server sends the Notification to the client to make the client connect to the server. After the client receives the notification message, it sends the initialization message with the device information back to the server to allow the server to identify the device. When all these identifications are completed, the server executes management operation for the initialization of the modulation module update. First, the DM Server sends operation message with "Add" command to create FwInfo as the entrance of the firmware management object. Then, the rest nodes related to the update management object are added by the "Add" commands within the next package. In the framework we proposed, the commands of adding the management objects are all in one operation message rather than the respective messages for higher efficiency [49]. Figure 3.4 shows the details of the initialization phase.

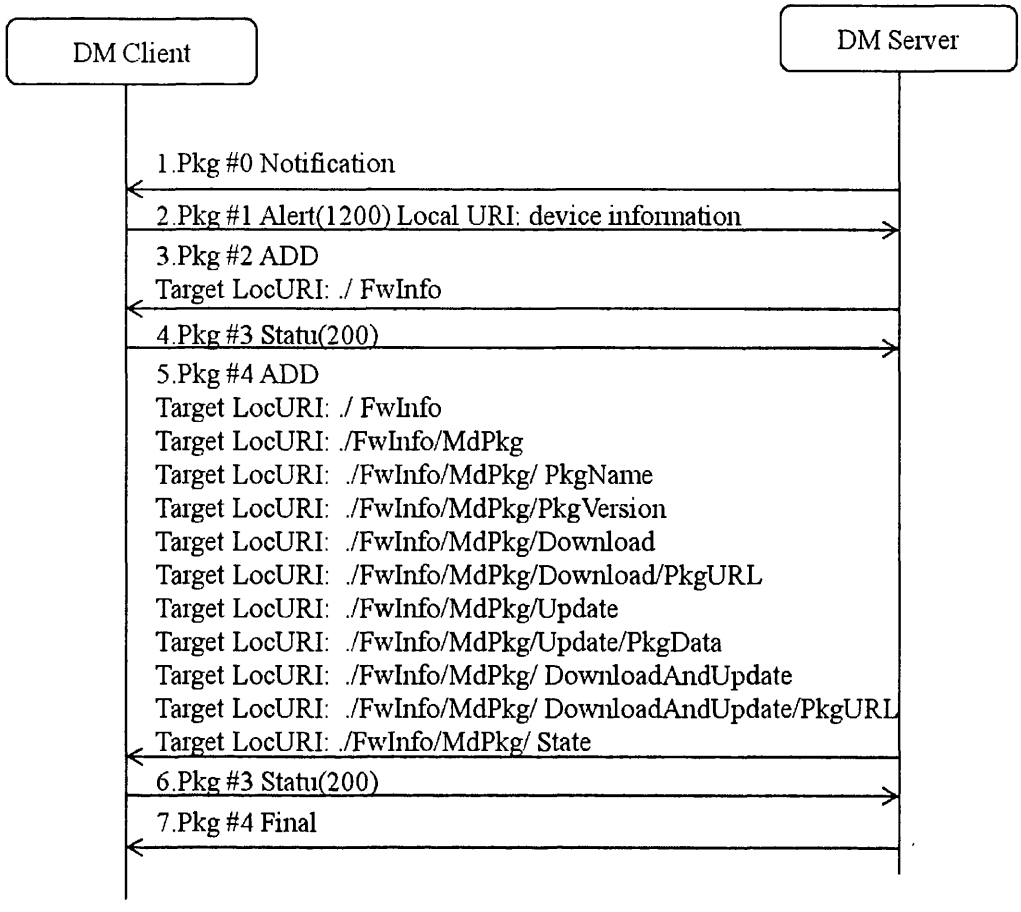

Figure 3.4 Initialization Phase

The detailed steps of the Initialization Phase are as follows:

1. DM Server sends notification to the mobile device in order to make the device connect to DM Server.
2. Mobile device connects to DM Server and starts the management session. Meanwhile, the mobile device sends client credential and device information to DM Server.
3. DM Server adds the entrance node "FwInfo" to the mobile device by using "Add" command. The entrance node needs to be added only once. After it is added to the DM Tree, other related nodes can be added to this node.
4. After the entrance node has been added successfully, the mobile device sends messages back to DM Server with the state information "200".
5. After DM Server receives the message with the success information, it starts to add other nodes to the mobile device.

6. After all of the nodes have been added successfully, the mobile device sends messages back to DM Server with the state information "200" which means the operation has been successfully completed.
7. After DM Server receives the message with the success information, it can start new management operations or terminate the current session.

3.3.3.2 Download Phase

Download Phase is a phase to realize the download operation by using proper management commands. The download phase can be activated right after the initialization phase or later. The precondition of download phase is that the management object of the firmware to be managed has already been existing in DM Tree.

The download of the modulation module phase is mainly based on Download node and the PkgURL node. By handling these two nodes, DM Server realizes the control of the download operation. Server sends operation message with "Replace" command to operate the ./Download/PkgURL for specifying the download descriptor of the modulation module. After that, the download procedure is started by the message received from the server with "Exec" command on Download node. Figure 3.5 shows the details of the download phase.

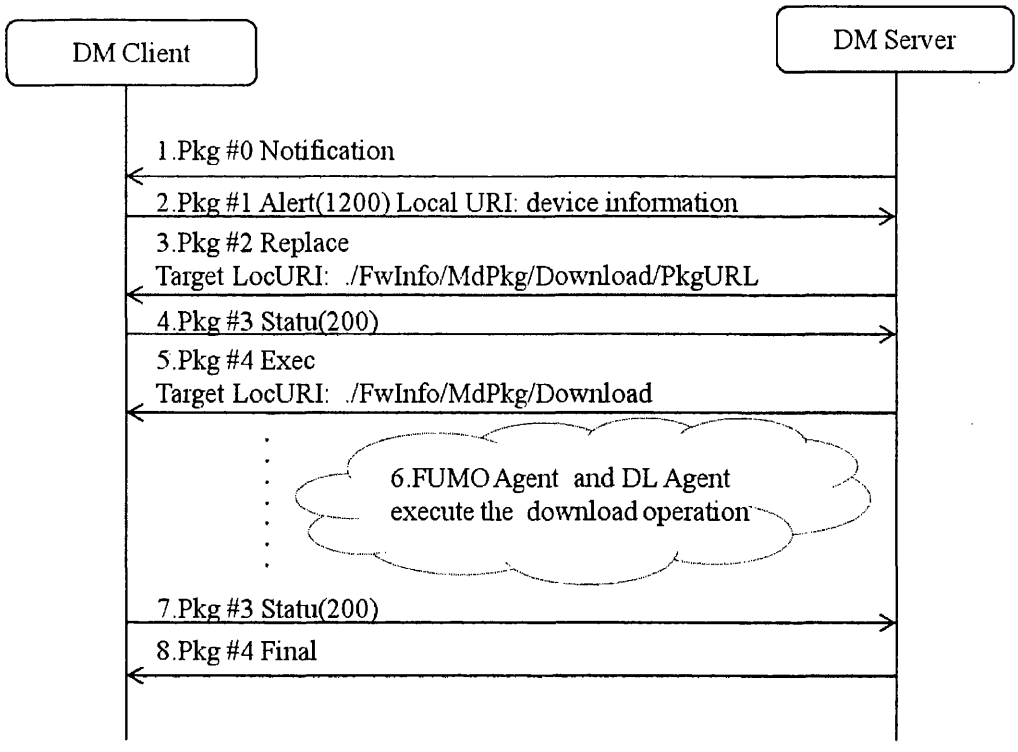

Figure 3.5 Download Phase

The detailed steps of the Download Phase are as follows:

1. DM Server sends notification to the mobile device in order to make the device connect to DM Server.

2. Mobile device connects to DM Server and starts the management session. Meanwhile, the mobile device sends client credential and device information to DM Server.

3. DM Server sends the URL information to mobile device by using "Replace" command. Mobile device gets the URL information and writes it into the node PkgURL.

4. If the mobile device receives the URL information and writes it into the node PkgURL successfully, it sends a message back to DM Server with the success state "200".

5. DM Server operates the Download node in the DM Tree of the mobile device by

using the "Exec" command to start the download operations.

6. Mobile device starts the download operation. The download of the firmware package has two methods, which are the centralized method and the decentralized method. In addition, the centralized method also includes two different ways. One way is to download the firmware package from the download server provided by other providers, using http protocol or other effective protocols. The other way is to download the firmware package from the device management download server by using the download protocol specified by DM Standard. In this thesis, we only take the former situation into consideration. The download mechanism we proposed will be introduced in next section.

7. If the download operation is successfully finished, the mobile device sends a message back to DM Server with the success state "200".

8. After DM Server receives the message with the success information, it can start new management operations or terminate the current session.

3.3.3.3 Update Phase

Update Phase is a phase to realize the update operation by using proper management commands. The update phase can be right after the download phase or not. The precondition of update phase is that the package of the firmware to be updated has already been downloaded to the mobile device.

Update phase is commenced by the message with "Exec" command on the update from the server. This phase is right after the client sends the message back to the server to notify the server that the download is finished. Client provides firmware update manager to manage the update operation of the modulation module on the device. Meanwhile, the interface related to the firmware layer is provided as well, which can be invoked to complete the modulation module update operation. Figure 3.6 shows the

details of the update phase.

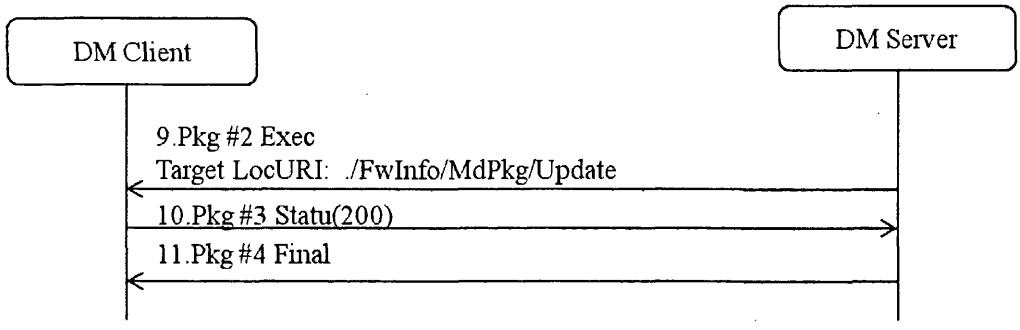

Figure 3.6 Update Phase

The detailed steps of the Update Phase are as follows. These steps are the extension of the Download Phase.

9. DM Server operates the Update node in the DM Tree of the mobile device by using the "Exec" command to start the Update operations.

10. Mobile device starts the Update operation. If the Update operation is successfully finished, the mobile device sends a message back to DM Server with the success state "200".

11. After DM Server receives the message with the success information, it terminates the current session. The management operation for the update of modulation module is finished.

3.3OTA Software Download Mechanism

The framework we proposed extends the download mechanism based on OMA DM, which combines the centralized and decentralized software download mechanism. Depends on different situations, the framework chooses the appropriate way to execute the software download.

3.3.1 Centralized Download Method

In present mobile networks, software upgrades are performed via downloading from central servers. The well-known client/server architecture as used in the mobile network is the underlying network concept for software distributions. As shown in Figure 3.7, the client requests to update the software from the central server, and then the server provides software to the client. Therefore, the server is the only element that can provide software to client for updating. Although the centralized download mechanism is the most traditional way to download software to mobile device, it is still difficult to deliver software to a large number of clients in a limited timeframe only with a few central servers.

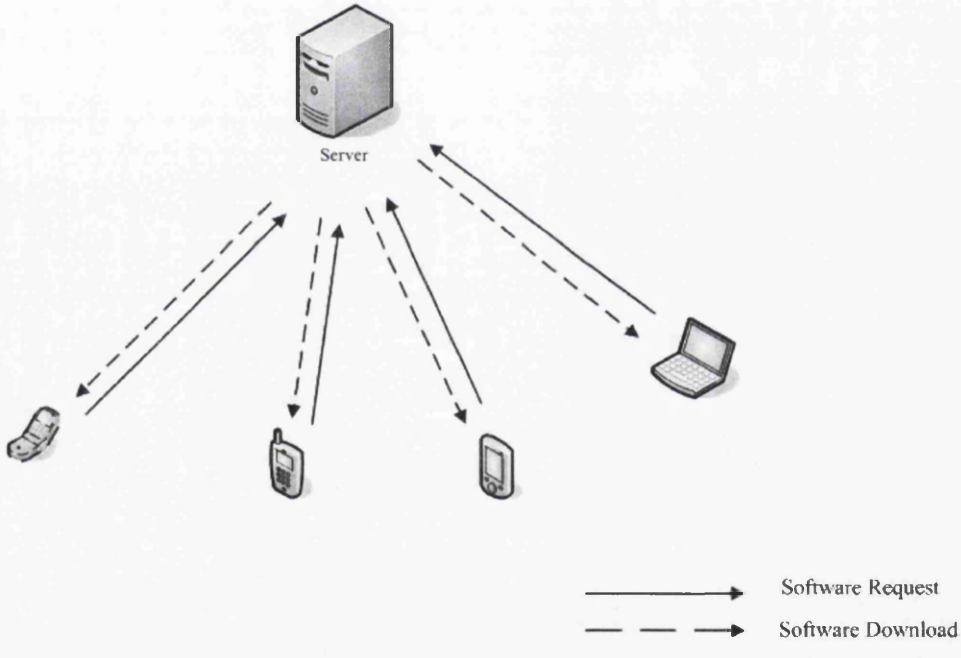

Figure 3.7 Centralized OTA Software Download

3.3.2 Decentralized Download Method

With the decentralized OTA software download method, software can be provided by mobile devices as shown in Figure 3.8, which means the software can be downloaded from the client nearby. Each mobile device serves as a possible software server. Therefore, software can be passed from one device to another through the connection

like Bluetooth, etc. This method for software distribution could cut down the network load and be more effective, especially when there are a lot of mobile devices waiting for the software to be updated.

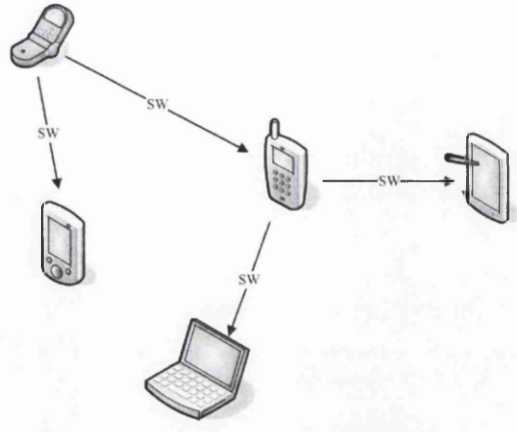

Figure 3.8 Decentralized OTA Software Download

3.3.3 Combined Download Mechanism

As mentioned before, this framework combines the centralized download method and the decentralized download method. When decentralized download is available, the framework will choose this way for software download. If not, centralized download is applied. Figure 3.9 depicts the combined download mechanism.

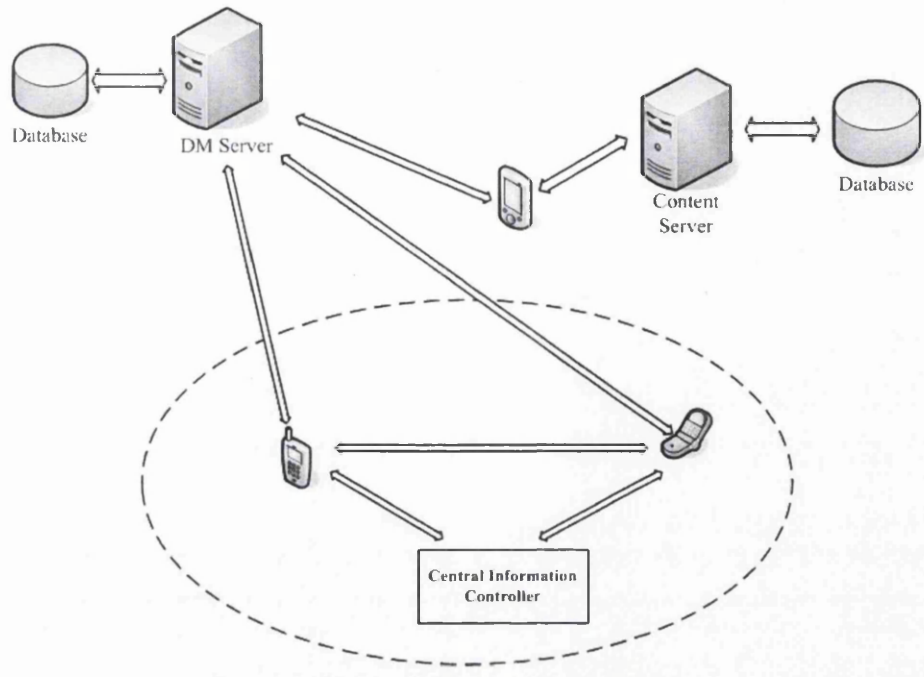

Figure 3.9 A Combined Download Mechanism

The decentralized part is dominated by the Central Information Controller (CIC) which plays a significant role in the entire architecture. CIC is responsible for managing the available software information. It broadcasts software information to mobile devices in specified area, which aims at informing the mobile devices with the information about the software needed in this area. In addition, CIC is also used to register the information of the mobile device with the available software, accept the software requirement messages from the devices which ask for the software to download and send the information needed back to those devices. Mobile devices carrying the related software register their information to the CIC when they receive the broadcast information from CIC asking for some software. Meanwhile, when one mobile device in the specified area needs some software, it requests CLC if there is software available in this area. If the software is available, the device gets the information of the device with the software from the CIC, then connects to that device and gets the software from it. If not, the device downloads the software from the central server by the centralized way. When the mobile device with the related software leaves the specified area, the CIC will delete the registered information of that device.

Figure 3.10 shows the firmware download flow of MMU with the combined centralized

and decentralized download method. When the download operation starts, the client, i.e. Client A, checks if it is in the area controlled by a CIC. If not, the modulation module will be downloaded by the centralized method from the content server. Otherwise, it checks if it is possible to obtain the modulation module locally from the other clients. It sends message to CIC to ask whether the modulation module is available for downloading by decentralized way. If it is available, CIC would send Client A the information about the client carrying the modulation module which is Client B. Then, Client A connects to Client B, and downloads the modulation module from Client B. If the modulation module is not available for decentralized way, Client A connects to the content server to download the modulation module after analyzing download descriptor by HTTP in the centralized way [50].

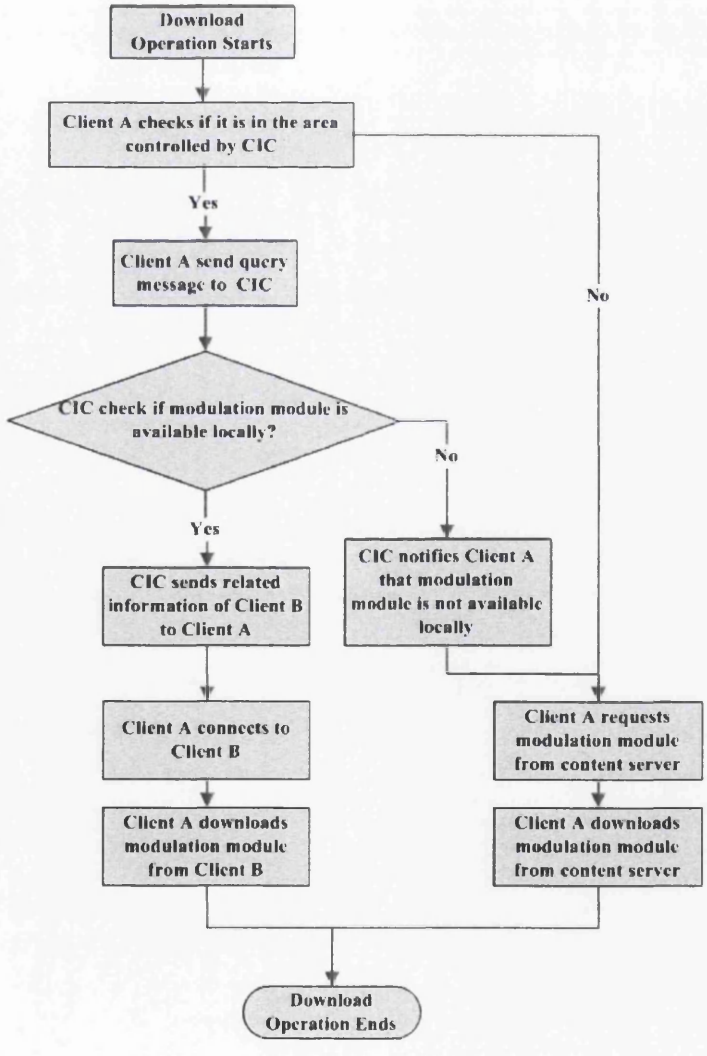

Figure 3.10 Flow of the download Mechanism of MMU

3.4 Summary

In this chapter, the design of the MMU is elaborated. This chapter presents the overall design of MMU, mainly including the design of management architecture, the management object and the management operations. The nodes for the management object of modulation module are designed in detail, including the type, format, name, size, especially the ACL of each node. The management operations are divided into three phase which are initialization phase, download phase and update phase. Each phase is elaborated in this chapter. The design of over-the-air software download mechanism which employs the centralized download method and decentralized download method for different circumstances is elaborated as well in this chapter.

4. Implementation of Modulation Module Update

4.1 Introduction

The design of the management objects, management operation mechanism and the download mechanism of MMU have been described in last chapter. Therefore, in this chapter, the implementation of MMU is presented from a coding point of view. Based on the management mechanism and download mechanism presented in last chapter, the implementation in this chapter is elaborated from three aspects, which are the DM Server side, the client side in mobile devices and the Central Information Controller side.

4.2 Implementation of DM Server

The main task of DM Server is the device management, which mainly depends on the message exchange between server side and client side. From a functional perspective, the design and implementation of device management function can be divided into five parts as shown in Figure 4.1, which are User Interaction Layer, DM Protocol Layer, SyncML Representation Layer, Transport Layer, and Network Layer. This hierarchical five-layer structure is a traditional one to implement DM Server. Each layer in this structure focuses on its own functions and communicating with other layers by interfaces. Based on the traditional five-layer structure, we implemented the DM Protocol Layer and SyncML Representation Layer in a more flexible way. The implementation of these two layers in this thesis is based on relatively independent classes which can be seen as the different components in these layers. These components realized by different classes are respectively related to the operation commands or the elements in syncML Messages, which can be combined into different operations or different syncML messages based on different requirements. This makes these layers more flexible and is easy for use and maintenance. The detailed

implementation of DM Server is as follows.

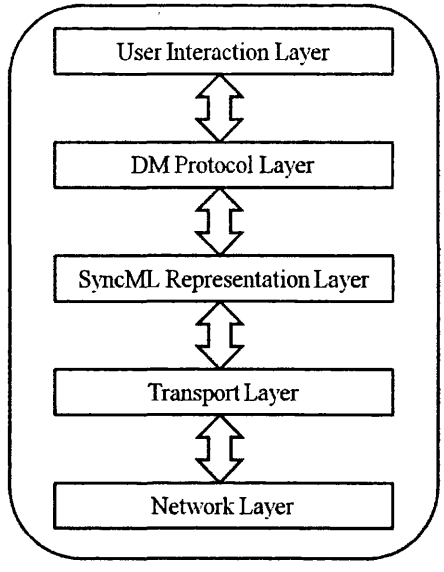

Figure 4.1 Functional Layers of Dm Server

4.2.1 User Interaction Layer

User Interaction Layer is the highest layer of device management. It is the layer that directly handles the management requirements of users. This layer provides user interface to get the management request from users, such as the request of software update or data collection, etc. It also analyses and processes these requests, then converts them into related management instructions and passes these instructions to the lower layer. Besides, it is also used to present related information to the users, such as the software upgrade results, and so on.

As for the modulation module update requirement, User Interaction Layer needs to deal with a series of commands, such as Add, Copy, Delete, Get, Replace, Exec, etc. According to the specific need, these commands can be logically combined so as to realize various complex and orderly management operations.

The implementation of User Interaction Layer mainly includes two packages which are dm.server.user and dm.server.rqmanager. The detailed design of dm.server.user and

dm.server.rqmanage is shown in Figure 4.2.

The dm.server.user package is for dealing with getting the user management requirements and presenting the management results. There are mainly two classes in this package, which are Requestproc and Resultsproc.

The dm.server.rqmanage package is used for converting the management requirements of users into related management instructions by the ordered combination of different commands. There are several classes in this package, including IRequest, RqConvert, IResult, RqResult, RqAdd, RqDel, RqGet, RqExec, and so on.

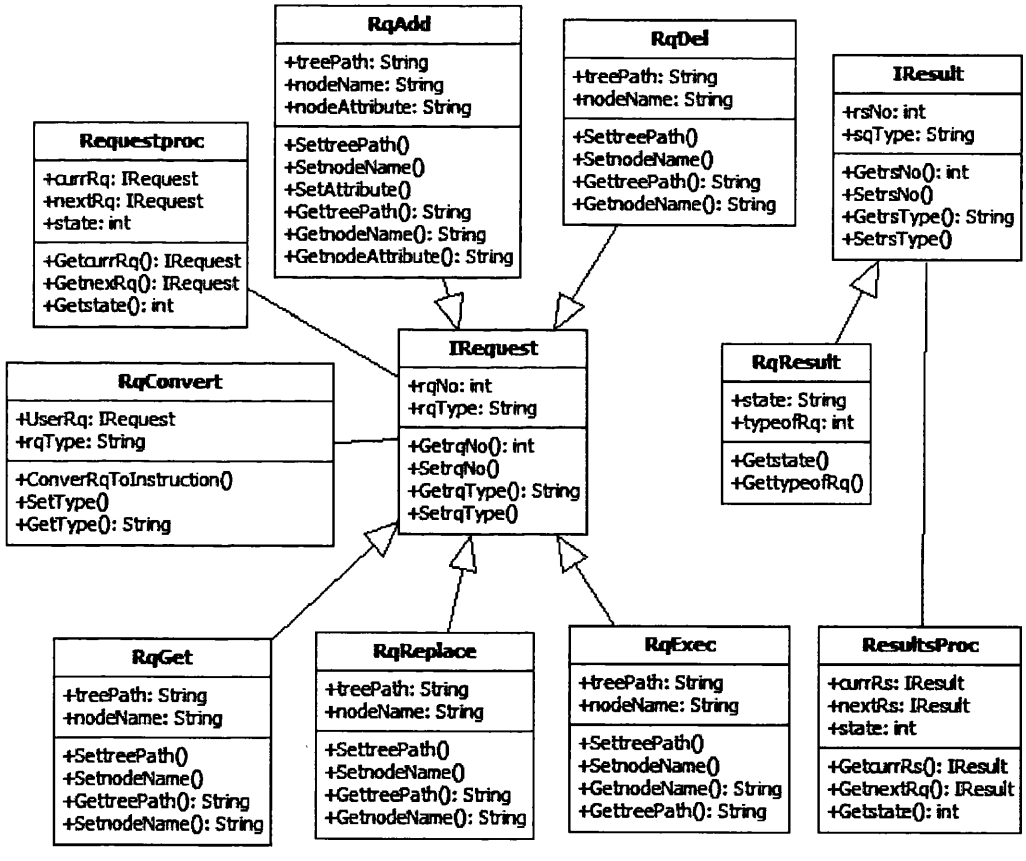

Figure 4.2 Design of User Interaction Layer

4.2.2 DM Protocol Layer

DM protocol is a significant layer, which is the highest layer to process the message

exchange between DM server and mobile devices. It is responsible for managing the process of device management. On the one hand, DM protocol layer gets the SyncML objects parsed by SyncML Representation Layer from SyncML messages as well as the identification information from Transport Layer. On the other hand, the SyncML objects processed by DM Protocol Layer will be passed to SyncML Representation Layer. Then, after processed by SyncML Representation, they will be sent to mobile device through Transport Layer and Network Layer.

DM Protocol Layer controls the entire procedure of the management operation between DM Server and mobile devices. Thus, it provides not only the management function of single session but also the control function of the entire session pool. This layer resides between User Interaction Layer and SyncML Representation Layer and interacts with them. It is used for processing the user management instructions from the User Interaction Layer, which encapsulates these instructions into SyncML objects and passes them to the SyncML Representation Layer for further processing. Meanwhile, it is also used for processing the SyncML objects passed from SyncML Representation Layer and feeding back the processing results to User Interaction Layer.

From the perspective of implementation, DM Protocol Layer includes two main packages, which are dm.server.engine and dm.server.dmoperation. The detailed design of these two packages is shown in Figure 4.3 and Figure 4.4.

Figure 4.3 shows the design of dm.server.engine, which aims to manage the sessions and the session pool, and control the initiation, operation, processing and termination of the sessions. There are mainly two classes in these two packages, which are dmSession and dmSessionPool.

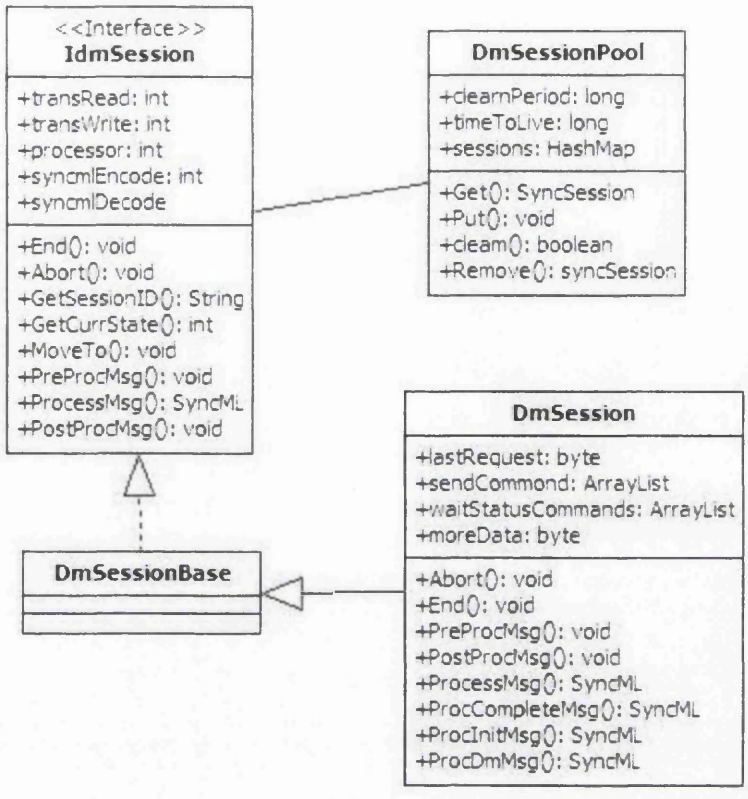

Figure 4.3 Design of dm.server.engine

Figure 4.4 shows the design of dm.server.dmoperation, which realizes the user instructions in SyncML objects. As mentioned, the operations, such as adding nodes and deleting nodes, are all encapsulated into related components. There are several classes in dm.server.dmoperation package, which inherits the IdmOperation interface. These classes are used to convert the user instructions into the elements of SyncML objects. As shown in Figure 4.4, these classes are corresponding to the operation commands, such as Add, Delete.etc.

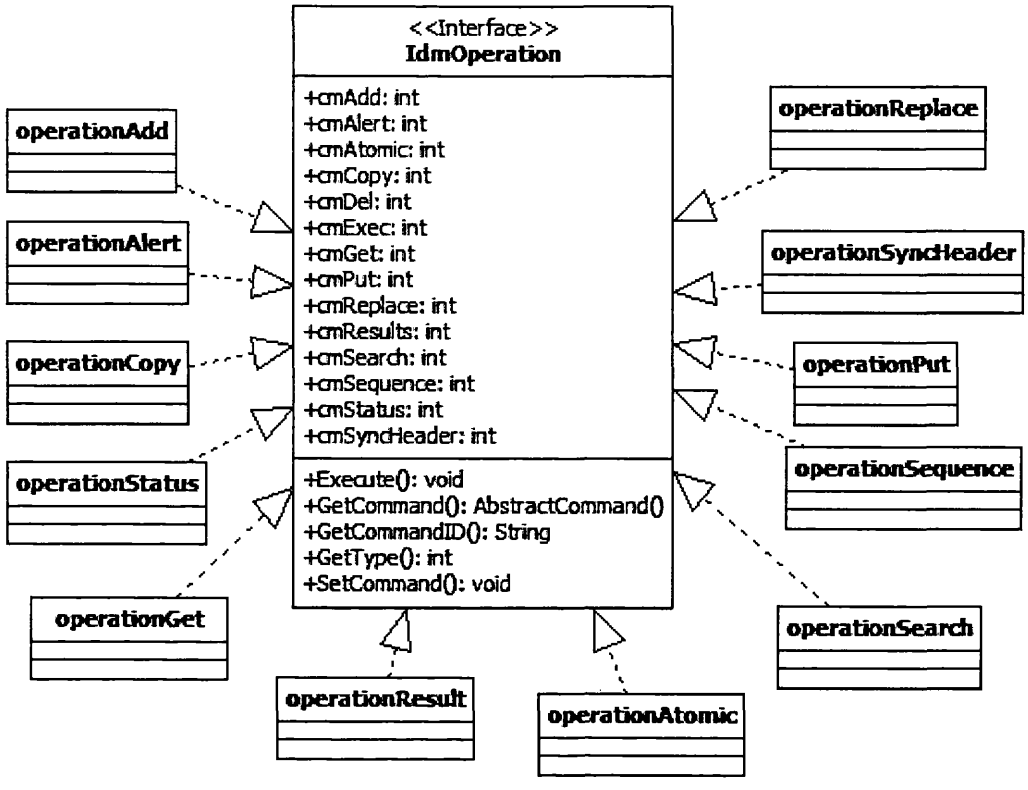

Figure 4.4 Design of dm.server.dmoperation

4.2.3 SyncML Representaion Layer

SyncML Representation Layer contains all the elements specified by SyncML protocol for message exchange. The layer realizes the mapping between the SyncML objects and the SyncML messages. Each element in the SyncML message can be mapped to an instantiation object. The management instructions set by users can be converted into SyncML message from SyncML object by SyncML Representation Layer. Meanwhile, the SyncML messages from the client side can be parsed into the SyncML object for further processing by this layer as well.

The mapping between syncML object and syncML message is realized by a series of classes called element class. Each element in SyncML message can be mapped to the related element class. SyncML Representation Layer contains two main packages,

which are dm.message.syncml and dm.message.coder.

The dm.message.syncml package provides the element class needed, which contains two parts, the syncML classes and management command classes. Each element class is a component in this layer, which can be combined into different operations or different syncML messages based on different requirements. The changing of the component, such as adding or deleting, will not influence other components.

Figure 4.5 shows the detailed design of syncML classes. SyncML classes realize the basic label elements of syncML messages, such as SyncHdr, SyncBody, Meta, VerProto, Source, Target, SessionID,VerDTD, Cred, etc.

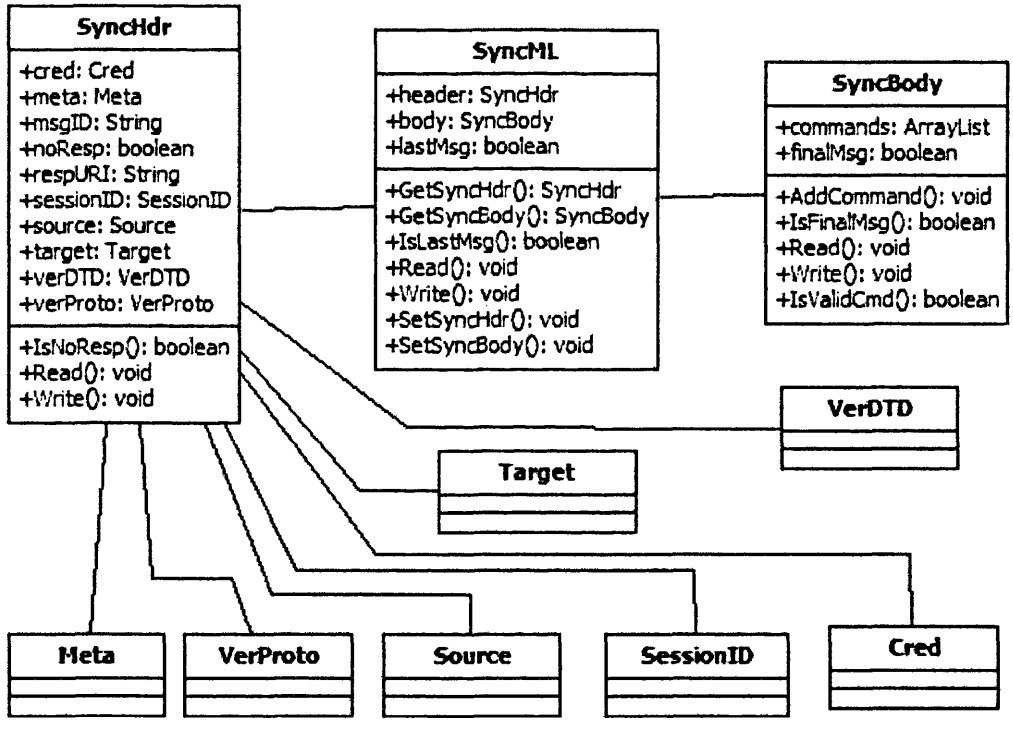

Figure 4.5 Design of SyncML Classes

Figure 4.6 shows the detailed design of management command classes. Command classes realize the operation elements in syncML messages, such as Add, Delete, Get,Replace, Exec, Alert, Sequence, etc.

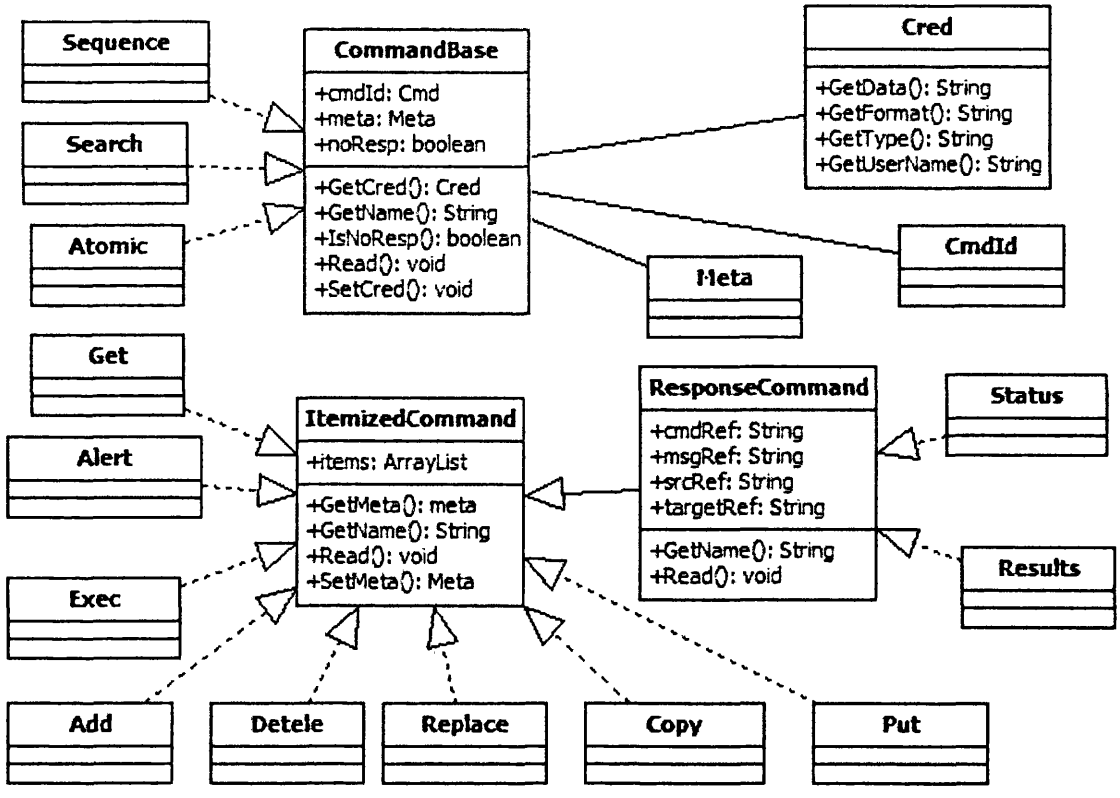

Figure 4.6 Design of Management Command Classes

The dm.message.coder package provides the rest functions which decode the message from the Transport Layer into SyncML message and encode SyncML message for the Transport Layer. The design of dm.message.coder is shown in Figure 4.7.

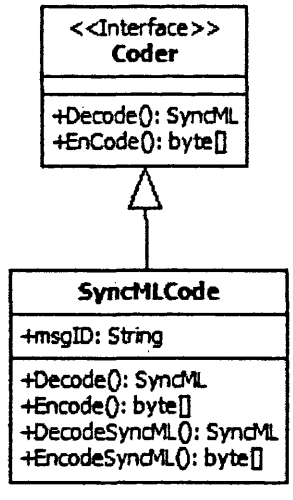

Figure 4.7 Design of the dm.message.coder

4.2.4 Transport Layer

Transport Layer is the carrier of message exchange between server side and client side of MMU. At present, there are three kinds of SyncML message exchange carrier in SyncML protocol, which are HTTP, WAP, and OBEX [51]. In this thesis, the MMU is mainly based on Http. The server side uses Jave Sevlet to process the HTTP requests from the client side. The business logic is encapsulated as a servlet, which can be invoked by HTTP request from client side.

Transport Layer includes two important classes, DmServlet and ProtocolInfo. DmServlet inherits javax.servlet.http.HttpServlet, aiming for receiving and sending SyncML messages. ProtocolInfo is used for saving the basic HTTP protocol information, including host, content-type, accept, etc. DmServlet gets the protocol information by dopost() and encapsulates it into protocolInfo, then passes it to SyncML Representation Layer for further processing. The design of DmServlet and ProtocolInfo is shown in Figure 4.8.

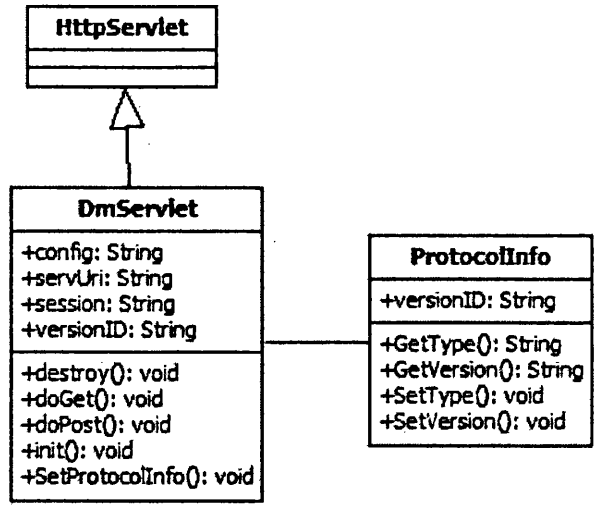

Figure 4.8 Design of Transport Layer

4.3 Implementation of Agents in Mobile devices

The main task of Agents in mobile devices is to communicate with DM Server to

achieve the mobile device remotely management operation. Therefore, due to the different aims of the management, the functions of the agents in mobile devices are different. In this thesis, we designed the Agents in mobile devices for the function of software upgrade over-the-air.

The implementation of these clients is also component based. Each part or function module can be seen as a component in the mobile device, which is relatively independent to each other. Each component provides interfaces to other components and invokes the interfaces provided by other components to realize the communications between different components. The change of each component will have less influence on other components, and new functionalities can be added by adding new component. This implementation to designMMU is more effective than traditional implementation method and makes the framework more flexible and easy to maintenance. In this session, the detailed design of these agents is presented.

4.3.1 Overall design

This section describes the overall design of the Clients of MMU from the aspect of function and the aspect of implementation. The specific design is as followed.

4.3.1.1 Functional Design

As we mentioned in last chapter, the client side of MMU is the clients residing in mobile devices, which is responsible for processing the management operations from server side. DM Server achieves the management function of mobile device by the message exchange between the server side and client side. Focusing on the software upgrade function, the client side is composed of several Agents in mobile devices, which are DM Agent, FUMO Agent, DL Agent and DM Tree. Figure 4.9 shows the functional structure of the client side of MMU.

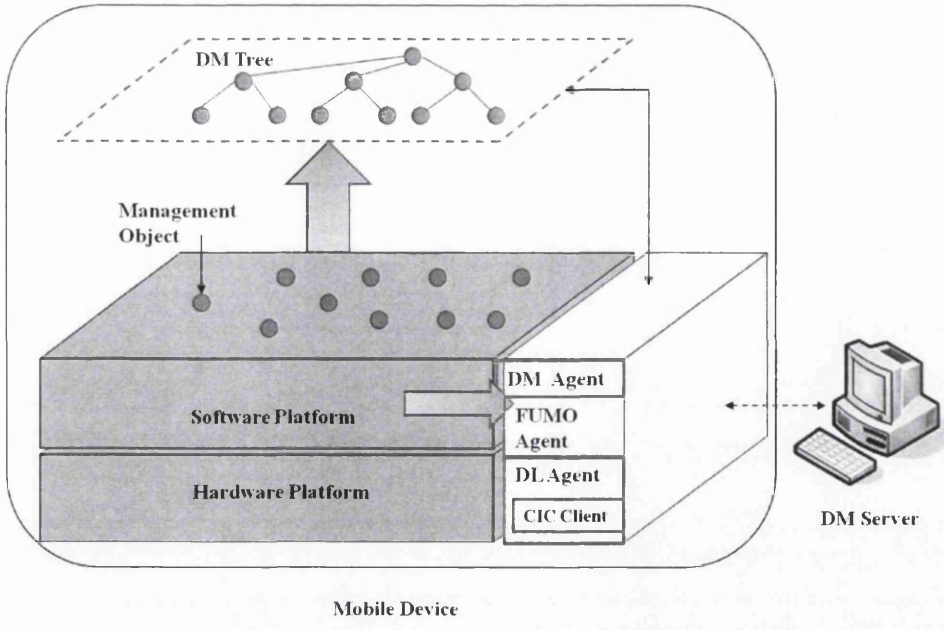

Figure 4.9 Functional Structure of Client Side

4.3.1.2 Programming Design

In this section, the design of the client side of MMU is presented from a programming point of view. As shown in Figure 4.10, the implementation of MMU is divided into four layers. They are, from the top to bottom, Application Layer, DM Core Layer, Hardware Abstract Layer, and Operating System.

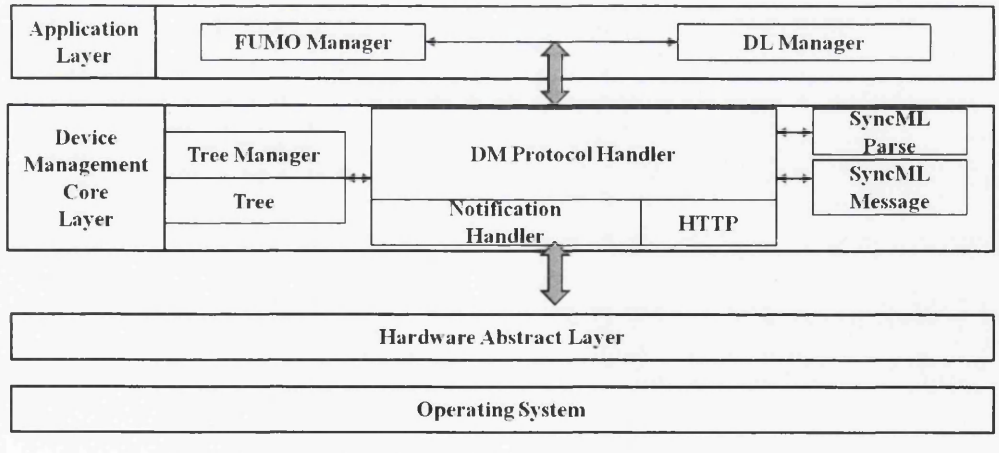

Figure 4.10 Programming Design of Agents in Mobile Device

4.3.2 Device Management Core Layer

DM Core Layer is the layer to realize DM Agent, which provides the basic device management functions based on OMA DM Standard. It parses the messages received from DM Server into the related operation instructions, and passes them to Application Layer for further processing. Besides, it is also responsible for encapsulating the management results into SyncML messages and passes them to the lower layer. DM Core Layer includes seven function modules which are DM Tree Manger, DM Tree Storage, SyncML Parse, SyncML Manager, DM Protocol Handler, Notification Handler, and HTTP Handler. DM Tree is implemented by the DM Tree Manger module and DM Tree Storage module. And DM Agent is realized by the rest modules. As mentioned, in this thesis, the implementation of each function module is based on the concept of component. The detail design is as follows.

4.3.2.1 DM Tree

From a coding point of view, DM Tree is a tree structure of the management objects. Each node in DM Tree is corresponding to a management object for the device management. Due to the management operation of devices is based on the operation of the DM Tree, the implementation of DM Tree is composed of two packages which are dm.client.tree and dm.client.treemanager.

The dm.client.tree package is used for realizing the basic elements of DM Tree, including the interior node and leaf node as well as the attributes of them. The dm.client.treemanager package is to implement the operation of DM Tree, such as adding nodes, deleting nodes, changing attributes, etc. There are five classes in these packages, including TreeNodeBase, DmACL, TreeManager, InteriorNode, Leafnode. The detailed design of these two packages is shown in Figure 4.11.

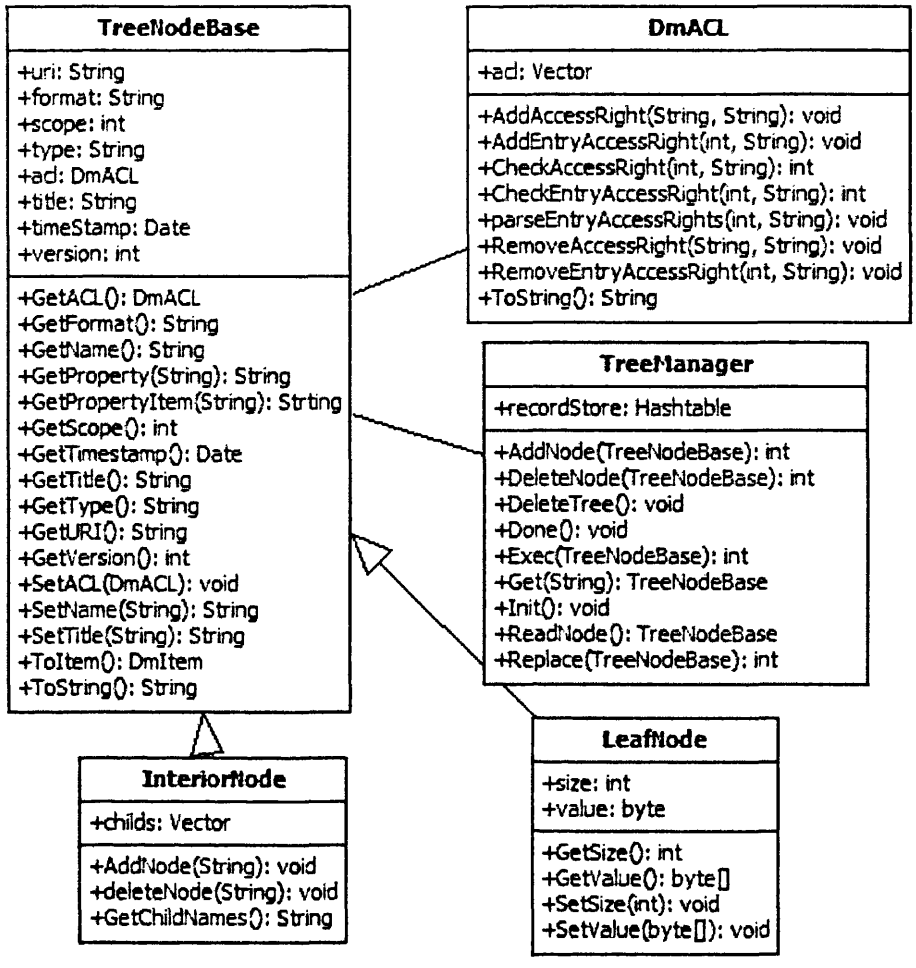

Figure 4.11 Design of DM Tree

4.3.2.2 DM Agent

DM Agent is used for realizing the basic functions of device management, which is the basis of software upgrade and other further functions. DM Agent provides the function of message exchange specified by OMA DM Standard, such as parsing the syncML messages, and so on. Therefore, other agents can exchange information with DM Server based on DM Agent so as to realize other new functions.

DM Agent is composed of three packages which are dm.client.protocolhander and, dm.client.syncmlparse, and dm.client.syncmlmsg.

The dm.client.protocolhander package contains a series of classes of DM management,

which is used for controlling the starting, operation, processing and termination of sessions. The dm.client.syncmlparse package is used for decoding the messages from DM Server or encoding the SyncML objects into the SyncML messages. The dm.client.syncmlmsg package is for implementing the classes of the elements in SyncML message, which has the same function as the SyncML Representation Layer in DM Server. The detail design of the DM Agent is shown in Figure 4.12.

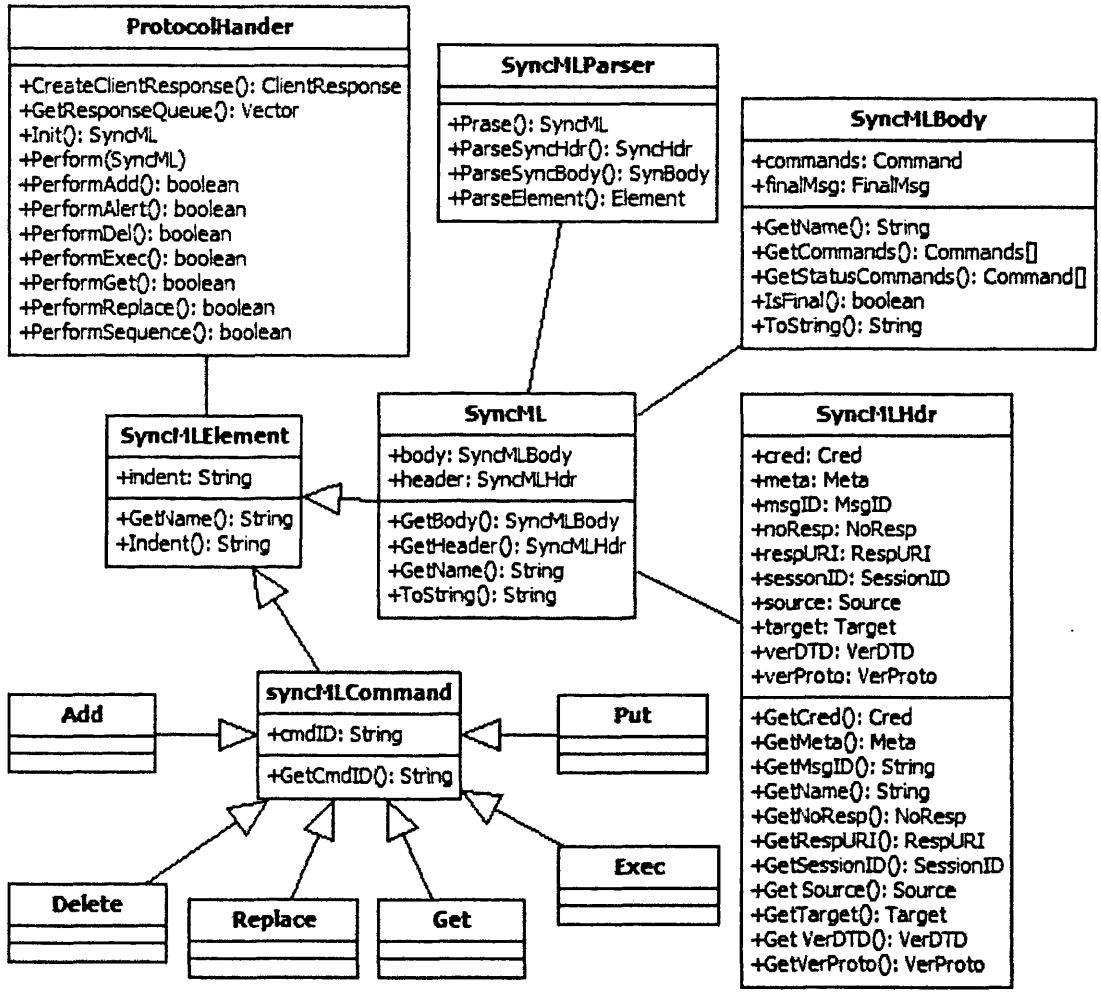

Figure 4.12 Design of DM Agent

4.3.3 Application Layer

Application Layer is the highest layer to directly process the management operation of software upgrade. It receives the operation instructions from DM Core Layer and realizes the specified management operations, such as software download, software

update, etc. Besides, it also sends back the operation results to DM Core Layer for further processing. Then, the results can be sent back to the server side. FUMO Agent and DL Agent are both in this layer.

4.3.3.1 FUMO Agent

As we mentioned in chapter 3, FUMO Agent is responsible for the firmware upgrade function for mobile devices. From the implementation point of view, the task of FUMO Agent focuses on the firmware upgrade, which means it is unnecessary for FUMO Agent to know how the message exchange is realized. In other words, the message exchange function is provided by DM Agent and FUMO Agent can just invoke the interface provided by DM Agent to exchange the firmware upgrade information with the DM Server.

The design and implementation of FUMO Agent is mainly composed of two packages which are dm.client.downloadmanager and dm.client.updatemanager.

The dm.client.downloadmanager is responsible for processing the software upgrade information parsed by DM Core Layer and managing the download of the software with the DL Agent. It is also responsible for the software download when centralized download method is employed. The dm.client.updatemanager package is to manage the update operation of the software needed to be updated. The detailed design of them is shown in Figure 4.13.

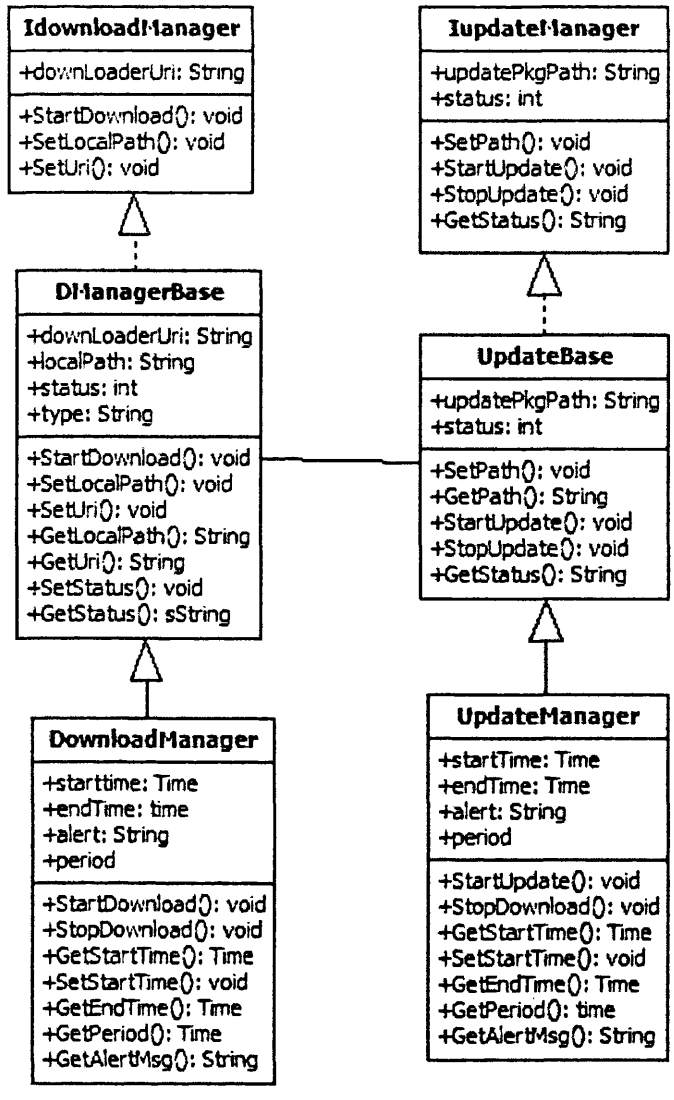

Figure 4.13 Design of FUMO Agent

4.3.3.2 DL Agent

DL Agent is designed for the software download mechanism we proposed. It works with FUMO Agent to realize the software download in different circumstances. In brief, it exchanges the information with CIC by the CIC Client module to make the decision of the download method and is responsible for achieving the decentralized software download when decentralized method is chosen.

The implementation of DL Agent is for making the decision of the download method and downloading software package by decentralized download method. Therefore, DL

Agent needs to have the capability for communicating with CIC to get the available software information. In addition, it also needs to have the function to communicate with the mobile device nearby for the software download by the decentralized method. The design and implementation of DL Agent is in one package named dm.client.dlagent.

The dm.client.dlagent is composed of five classes which are responsible for five different functions respectively. The DlManager class is the main part in DL Agent, which manages the operation of other classes and communicates with FUMO Agent to inform it the available download method. The RspforSwRQ class is used to require software information from CIC. The DlMethodDecider is responsible for deciding which download method is available based on the information getting from CIC. The Communication class is used for the communication between DL Agent and other devices, such as other mobile devices or the CIC. The decentralizedDownloader class is for the software download in decentralized method when decentralized download method is chosen. The detailed design of them is shown in Figure 4.14.

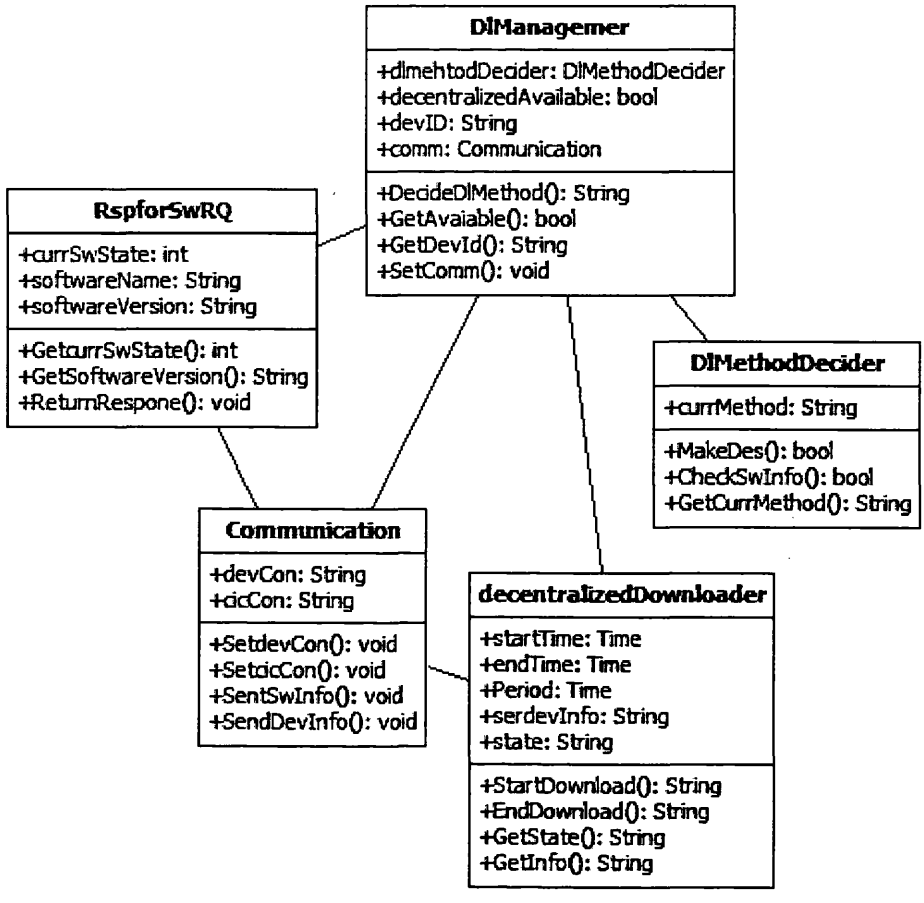

Figure 4.14 Design of DL Agent

4.4 Central Information Controller

CIC manages the available software information in the specific area and decides whether the decentralized download method will be used when the mobile devices request firmware to be updated. Therefore, from the coding point of view, the design and implementation of CIC is composed of five classes which are five function modules of CIC. These five classes are Broadcast, Register, DeviceManager, SoftwareManager and Communication. The detailed design of them is shown in Figure 4.15.

As CIC is a separate part out of the OMA DM framework, the detailed design and the way it works with DL Agent are described as follows.

Broadcast class uses 'Req4SW' to broadcast to its neighbors to request the software its

neighbors have and any neighbor who has the software returns the device and software information to CIC by the 'Req4Register' command from CIC. CIC will then register the mobile devices with the software by 'AddDev' function of Register module. If the device has left the area, the device information will be removed by 'RemoveDev' function. After registering the devices, CIC uses DeviceManager and SoftwareManager classes to manage specific device and software. When any device would like to upgrade/update its firmware, it will request the information from CIC, which is done by the Communication module. Once the information is found, the device will establish a direct link with the device having the software and download the software from that device. If CIC doesn't have the required information, the device will communicate with the Content Server directly and the traditional OMA DM download process will work.

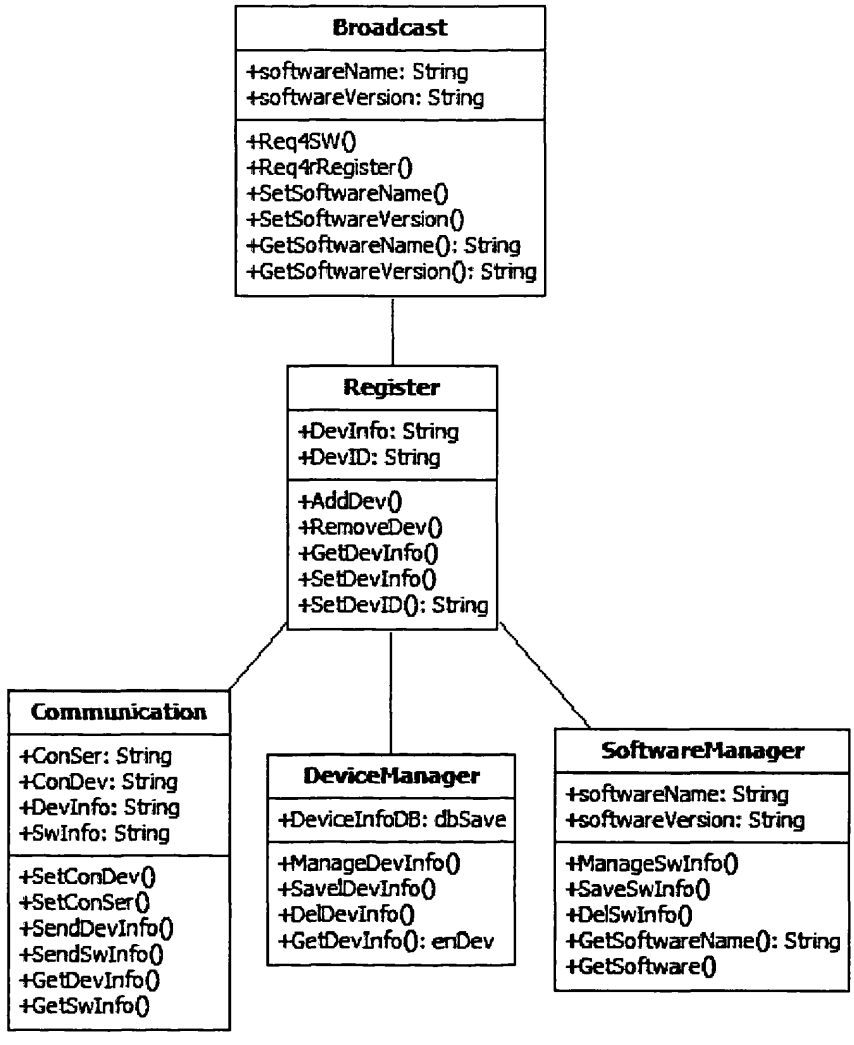

Figure 4.15 Design of CIC

4.5Summary

In this chapter, the implementation of MMU is elaborated from a coding point of view. The implementation of DM Server is realized by five parts which are User Interaction Layer, DM Protocol Layer, SyncML Representation Layer, Transport Layer, and Network Layer. The detailed implementation of the clients in mobile devices is also explained in this chapter, including the design of DM Tree, DM Agent, FUMO Agent, and DL Agent. Moreover, the detailed design of CIC is presented as well.

5. Test and Verification Results

5.1 Introduction

As we mentioned in earlier chapters, the MMU is an implementation of the framework we proposed for software upgrade. It is used for the update of the modulation modules in mobile devices. It implements the management objects and management operation mechanism as well as the download mechanism we proposed.

In this thesis, we simulated the server side and the client side of the framework, which are respectively MMU Server and MMU Client. MMU Server implements the basic functions of DM Server, which mainly aims at realizing the modulation module upgrade operations. MMU Client is a simulation of the mobile devices, which implements the agents we proposed for the software upgrade. CIC has also been implemented for the decentralized download method.

Two object-oriented programming languages, Java and C# are employed for the implementation of each part. The C# programming language is used to realize the user interaction part, such as the setting of the management operations, the result display, etc. The Java programming language is applied for the implementation of background processing.

MMU simulates both the centralized download method and decentralized download method for the modulation module upgrade. In the simulation, MMU Server, Content Server, CIC, as well as the MMU Clients are distributed in different laptops. The communications between them are based on wireless local area network. The experimental environment includes the IBM laptops with 1 GB memory. Windows XP is used as the operating system.

In this chapter, the results of the implementation of MMU are presented to verify the feasibility of the framework we proposed.

5.2 Test Procedure

In this section, the test method and procedure are elaborated. The test procedure is based on the three phases we proposed for the modulation module upgrade. Both of the centralized method and decentralized method are verified. The detailed test procedure for modulation module upgrade in centralized and decentralized scenarios is as follows.

5.2.1 Decentralized Download

The simulation environment we designed to test the decentralized download method is shown in Figure 5.1. MMU Server, Content Server, CIC, as well as the MMU Clients are distributed in different laptops. There are four MMU Clients with the labels 1-4 in this area, and one of them (MMU Client 1) carries the modulation module upgrade package. Therefore, for the other MMU Clients in this area, the download operation should be achieved by the decentralized method.

In this thesis, MMU Client 2 is used to test the decentralized download method. The detailed steps are as follows.

① Initialization phase. The tree nodes related to modulation module should be added to the DM Tree in MMU Client 2.

② The message exchange to start the download operation at the beginning of the download phase.

③ The information exchange between MMU Client 2 and CIC for decentralized download operation.

④ MMU Client 2 downloads the modulation module package from MMU Client 1 by decentralized download method.

⑤ MMU Client 2 returns the download result to MMU Server.

⑥ Update Phase. Update the modulation module in MMU Client 2.

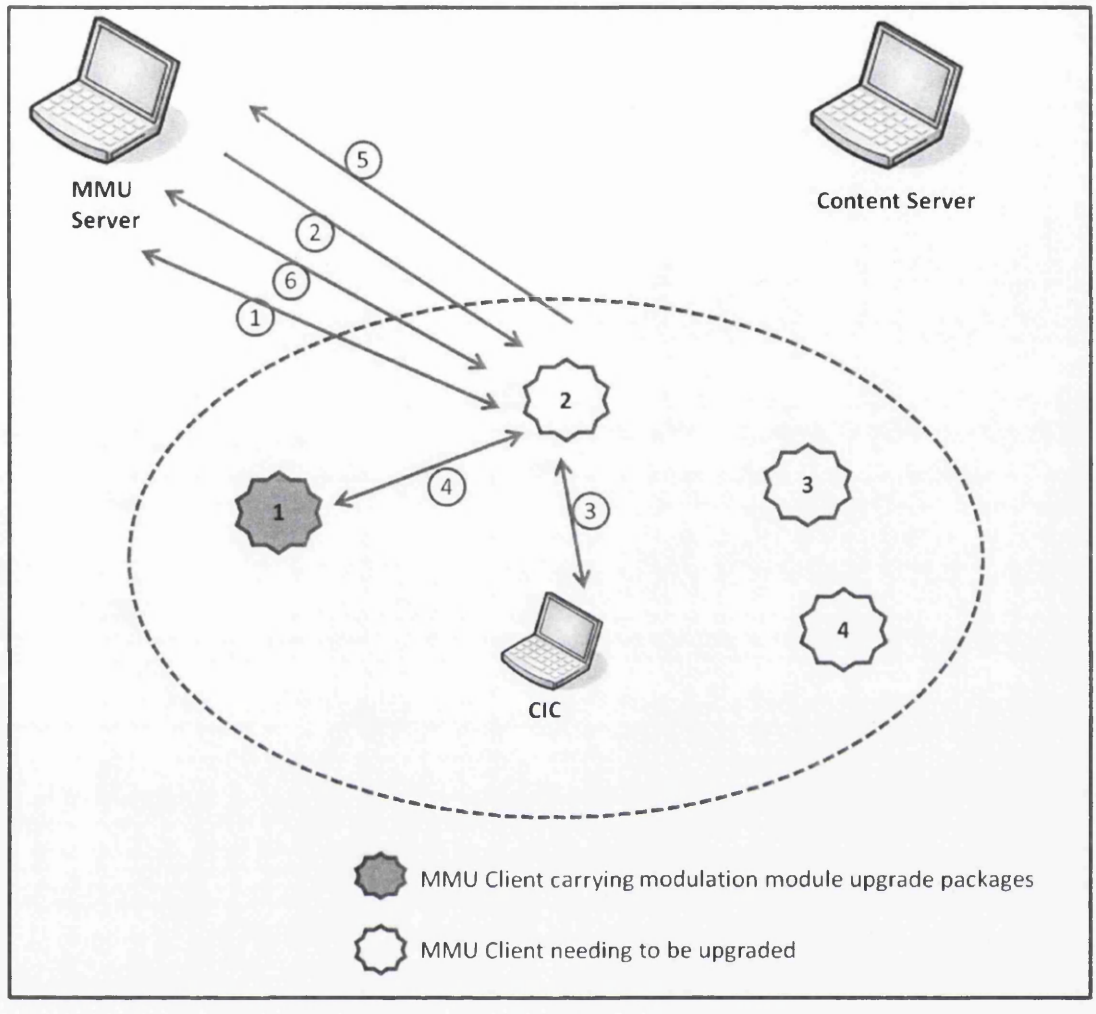

Figure 5.1 Decentralized Download Test

5.2.2 Centralized Download

The test for the centralized download method is after the test of decentralized method. The simulation environment we designed to test the decentralized download method is shown in Figure 5.2. The MMU Client 1 and MMU Client 2 have moved out of the area controlled by CIC. Therefore, the upgrade operation for MMU Client 3 and MMU Client 4 can only be achieved by centralized method.

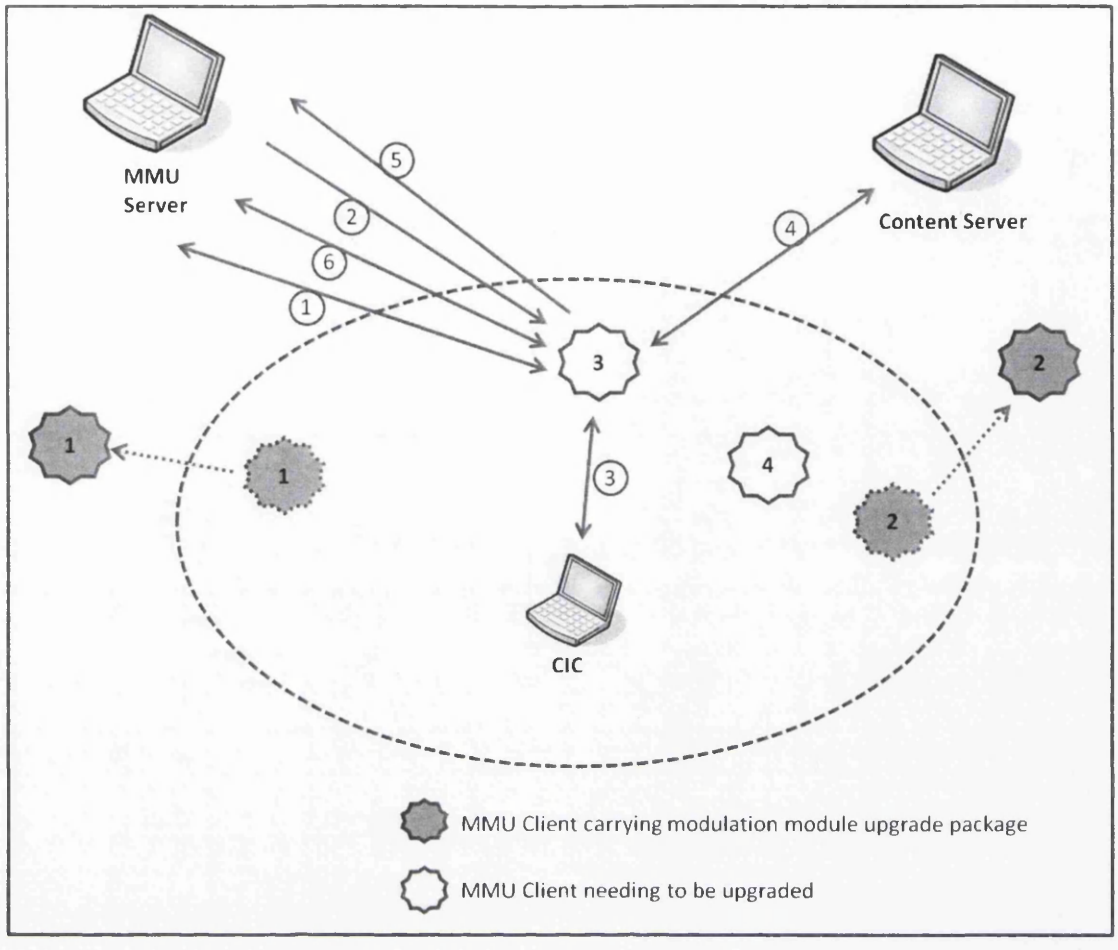

Figure 5.2 Centralized Download Test

In this thesis, MMU Client 3 is used to test the centralized download method. The detailed steps are as follows.

① Initialization phase. The tree nodes related to modulation module should be added to the DM Tree in MMU Client 3.

② The message exchange to start the download operation at the beginning of the download phase.

③ The information exchange between MMU Client 3 and CIC. CIC notifies MMU Client 3 that there are no other clients carrying the upgrade package in this area.

④ MMU Client 3 downloads the modulation module package from Content Server by centralized download method.

⑤ MMU Client 3 returns the download result to MMU Server.

⑥ Update Phase. Update the modulation module in MMU Client 3.

5.3 Test Results

The test results are presented in this section in the order of the three phases. As the initialization phase and update phase are the same as the centralized download test and decentralized download test, we will only show the results of the download phase for both tests.

5.3.1 Initialization Phase

The initialization phase is related to the step ① of both centralized download test and decentralized download test.

5.3.1.1 Configuration

To get connection with DM Server, a client needs to set a series of parameters properly. These parameters are the base of the save communication between MMU Server and MMU Client. The parameters include the device information, encoding mode and the account information of the server. Figure 5.3 shows the configuration interface of MMU Client 2.

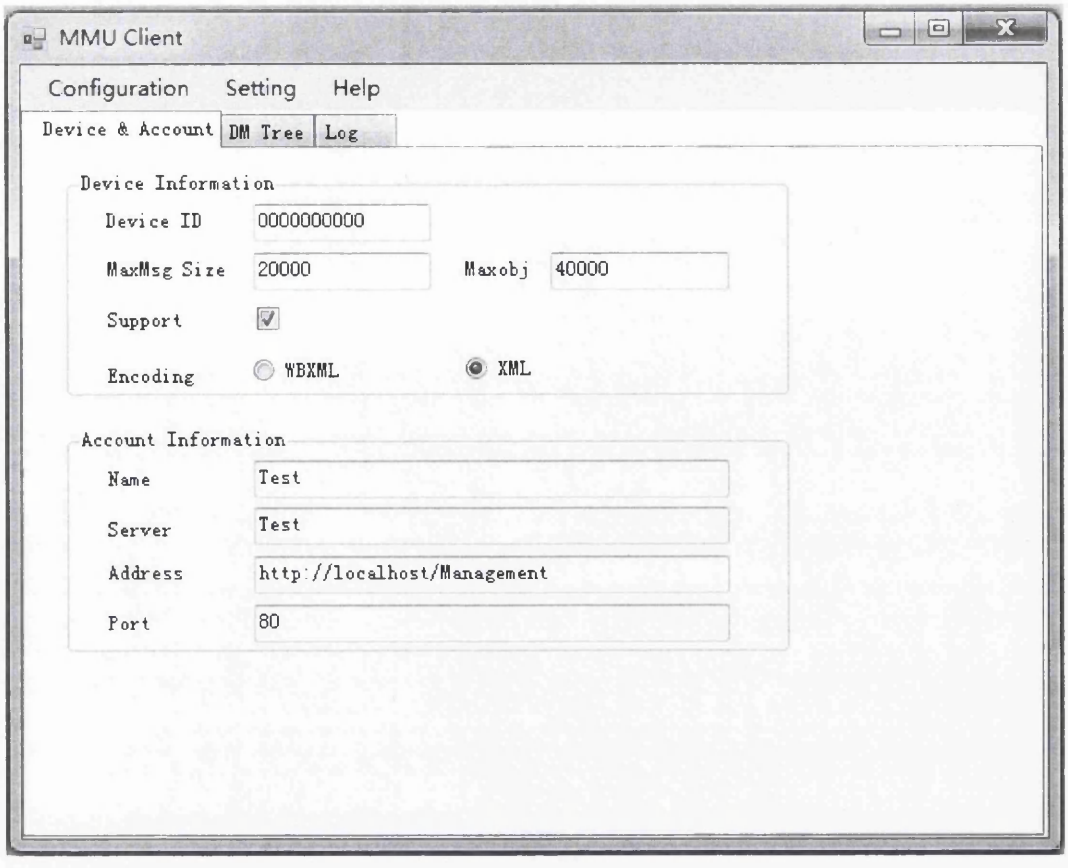

Figure 5.3 Configuration Interface in MMU Client 2

5.3.1.2 MMU Server

In MMU, the aim of initialization phase is to add the tree nodes which are related to modulation module to the DM Tree. Therefore, all nodes related to modulation module should be added before other phases start. The nodes should be added in a proper order as well which has been defined in Chapter 3. Figure 5.4 shows the information of MMU Server, which presents the operations defined by MMU Server and the related syncML messages sent to MMU Client 2. As shown in Figure 5.4, the node Mdpkg, which is the entrance node of other nodes, is added first. Then the rest nodes are added successively.

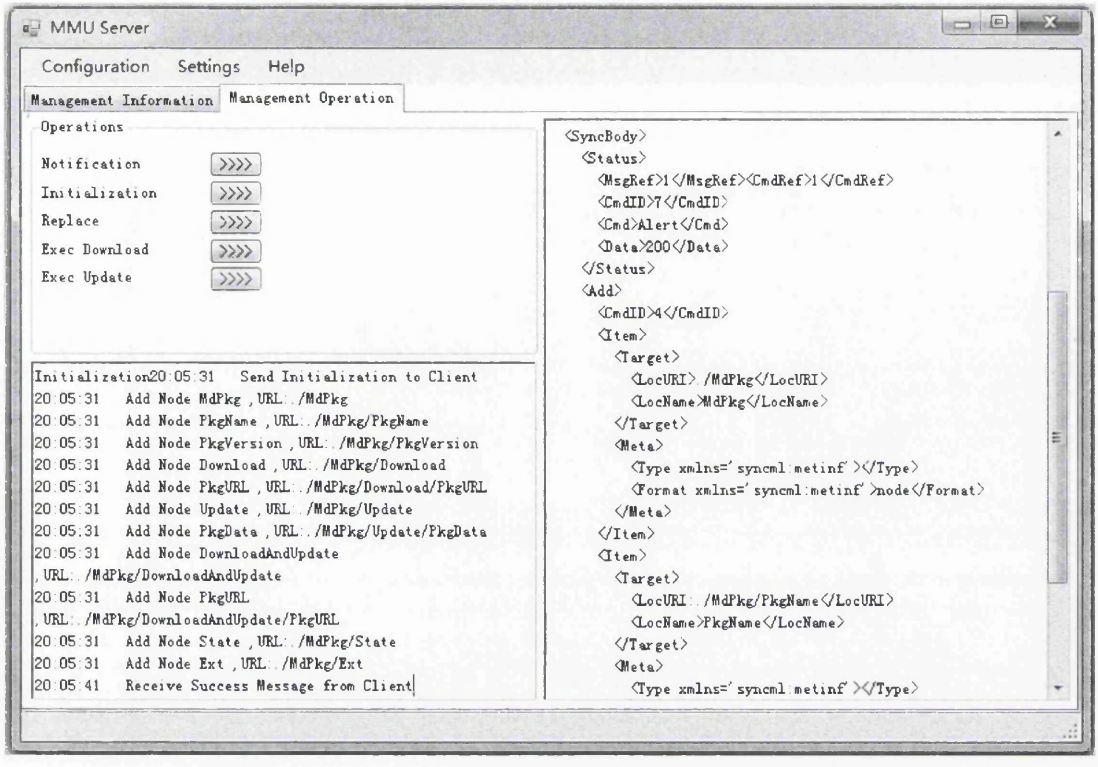

Figure 5.4 Log of Initialization in MMU Server

Table 5.1 shows the syncML message sent from MMU Server to MMU Client 2. This message is one of the messages in initialization phase, which is to add the FwInfo node into the DM Tree. The name and the path as well as the type of the node are included in this message under the command "Add".

Table 5.1 Message of Adding Management Object

```
<SyncML xmlns='SYNCML:SYNCML1.2'>
  <SyncHdr>
    <VerDTD>1.2</VerDTD>
    <VerProto>DM/1.2</VerProto>
    <SessionID>1</SessionID>
    <MsgID>1</MsgID>
    <Target>
      <LocURI>IMEI:000000000000000</LocURI>
    </Target>
    <Source>
      <LocURI>http://localhost/Management</LocURI>
    </Source>
  </SyncHdr>
  <SyncBody>
    <Status>
      <MsgRef>1</MsgRef><CmdRef>1</CmdRef>
      <CmdID>7</CmdID>
      <Cmd>Alert</Cmd>
```

```
        <Data>200</Data>
      </Status>
      <Add>
        <CmdID>4</CmdID>
        <Item>
          <Target>
            <LocURI>./FwInfo</LocURI>
            <LocName>FwInfo</LocName>
          </Target>
          <Meta>
            <Type xmlns='syncml:metinf'></Type>
            <Format xmlns='syncml:metinf'>node</Format>
          </Meta>
        </Item>
      </Add>
      <Final/>
    </SyncBody>
</SyncML>
```

5.3.1.3 MMU Client

After receiving the operation message from MMU Server, MMU Client 2 starts to execute the operations specified in the message. In initialization phase, the messages received from server are mainly for adding the nodes. As for the nodes adding, MMU Client 2 will check if the nodes exist already. If not, the operation of adding nodes will be executed.

Figure 5.5 shows the log information of the MMU Client 2, which presents the operations executed by MMU Client 2 and the related syncML messages sent back to MMU Server. As shown in Figure 5.5, the nodes related to modulation module are added successively.

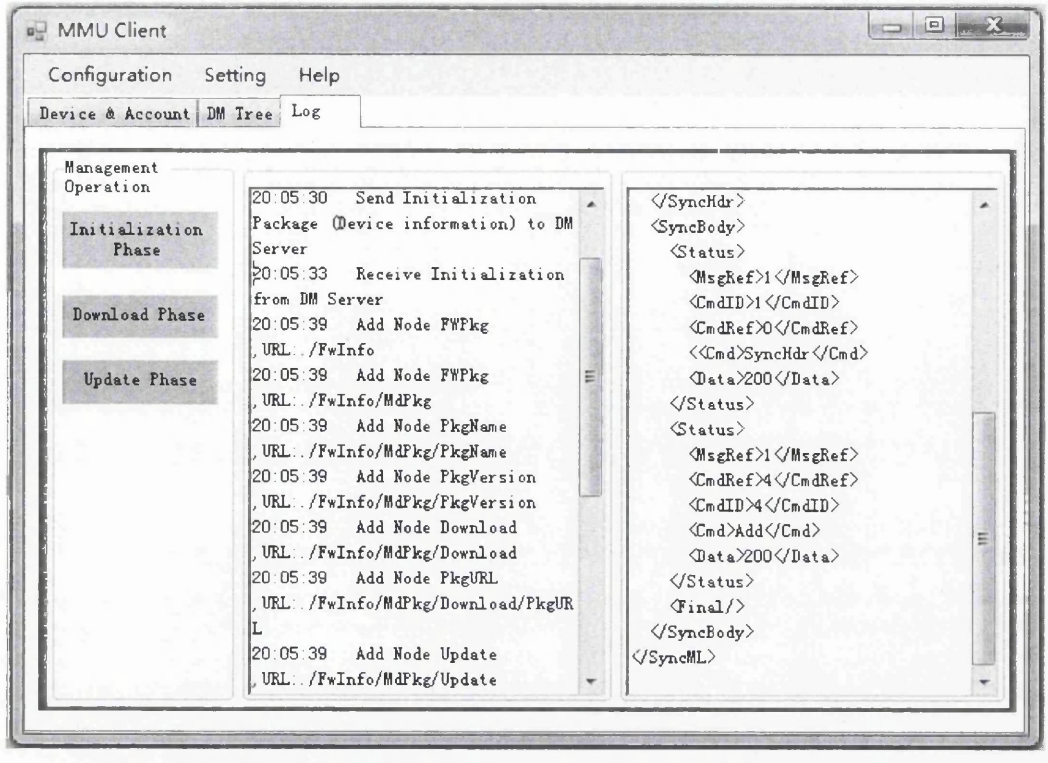

Figure 5.5 Log of Initialization in MMU Client 2

Table 5.2 shows the syncML message sent back to MMU Server from MMU Client 2. This message is one of the messages in initialization phase to inform the MMU Server that the node adding operation has been finished successfully. The status code 200 is included in this message under the element "Status".

Table 5.2 Message of Initialization Results of MMU Client 2

```
<SyncML xmlns='SYNCML:SYNCML1.2'>
  <SyncHdr>
    <VerDTD>1.2</VerDTD>
    <VerProto>DM/1.2</VerProto>
    <SessionID>1</SessionID>
    <MsgID>2</MsgID>
    <Target>
      <LocURI> http://localhost/Management </LocURI>
    </Target>
    <Source>
      <LocURI>IMEI:000000000000000</LocURI>
    </Source>
  </SyncHdr>
  <SyncBody>
    <Status>
      <MsgRef>1</MsgRef>
      <CmdID>1</CmdID>
      <CmdRef>0</CmdRef>
      <Cmd>SyncHdr</Cmd>
```

```
        <Data>200</Data>
      </Status>
      <Status>
        <MsgRef>1</MsgRef>
        <CmdRef>4</CmdRef>
        <CmdID>4</CmdID>
        <Cmd>Add</Cmd>
        <Data>200</Data>
      </Status>
      <Final/>
    </SyncBody>
</SyncML>
```

5.3.1.4 DM Tree

The tree nodes in DM Tree represent the management objects in mobile devices. Therefore, the adding of the node is not only just adding the node into the DM Tree, but also the attributes of the nodes. Figure 5.6 show the nodes of the DM Tree before the nodes of the modulation module are added. The nodes in Figure 5.6 are the basic nodes in a DM Tree. That means if a mobile device could be managed by a DM Server, these nodes are necessary. Figure 5.7 shows the nodes of the DM Tree after the nodes of modulation module are added in initialization phase. As shown in Figure 5.7, the nodes as well as their attributes are added into the DM Tree.

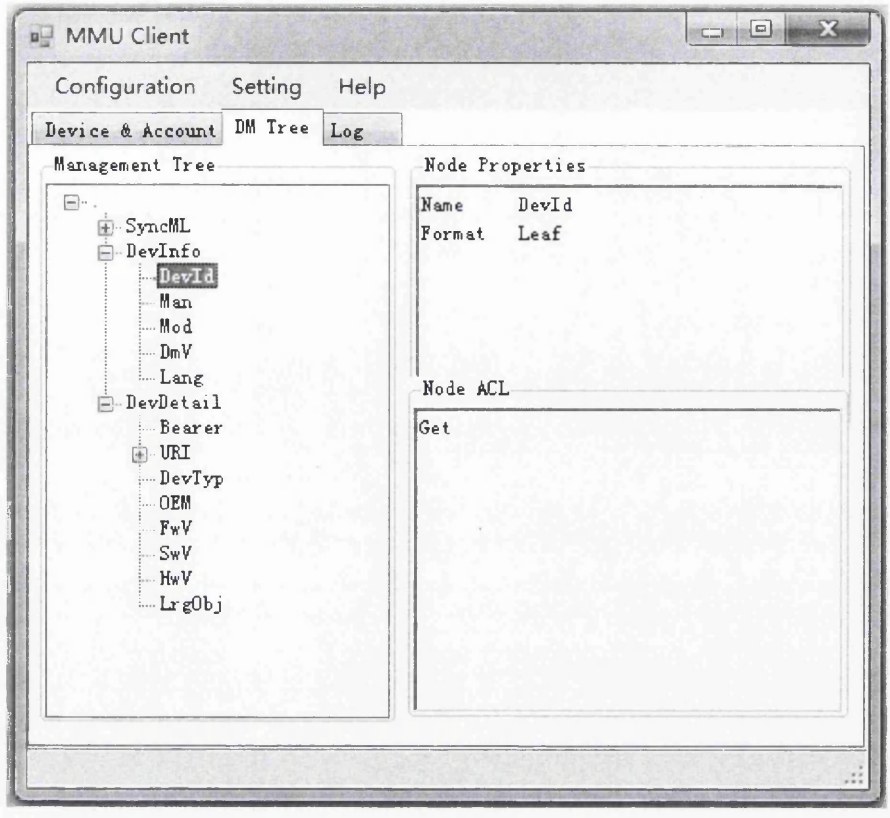

Figure 5.6 DM Tree before Initialization

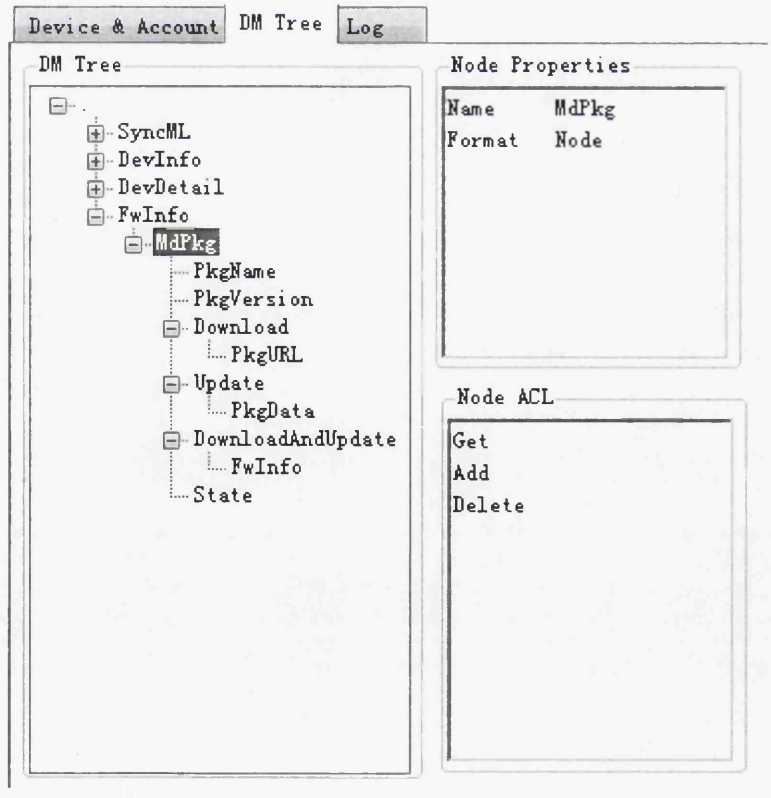

Figure 5.7 DM Tree after Initialization

5.3.2Download Phase

The update phase is related to the step ②③④⑤ of both centralized download test and decentralized download test.

5.3.2.1 Message Exchange in Download Phase

The message exchange at the beginning of download phase is related to the step ② of both centralized download test and decentralized download test.

As mentioned, the task of download phase is to realize the download execution of modulation module by proper method. Actually, the download method is decided by MMU Client. That means in MMU Server side, the management operation defined is the same as both download method. Therefore, in this section, we only present the test results between MMU Server and MMU Client 2 in this section.

Figure 5.8 shows the log information of the MMU Server, which presents the operations defined by MMU Server and the related syncML messages sent to MMU Client 2. As shown in Figure 5.8, during the download phase, there are two management operations need to be executed. The two operations are "Replace" and "Exec".

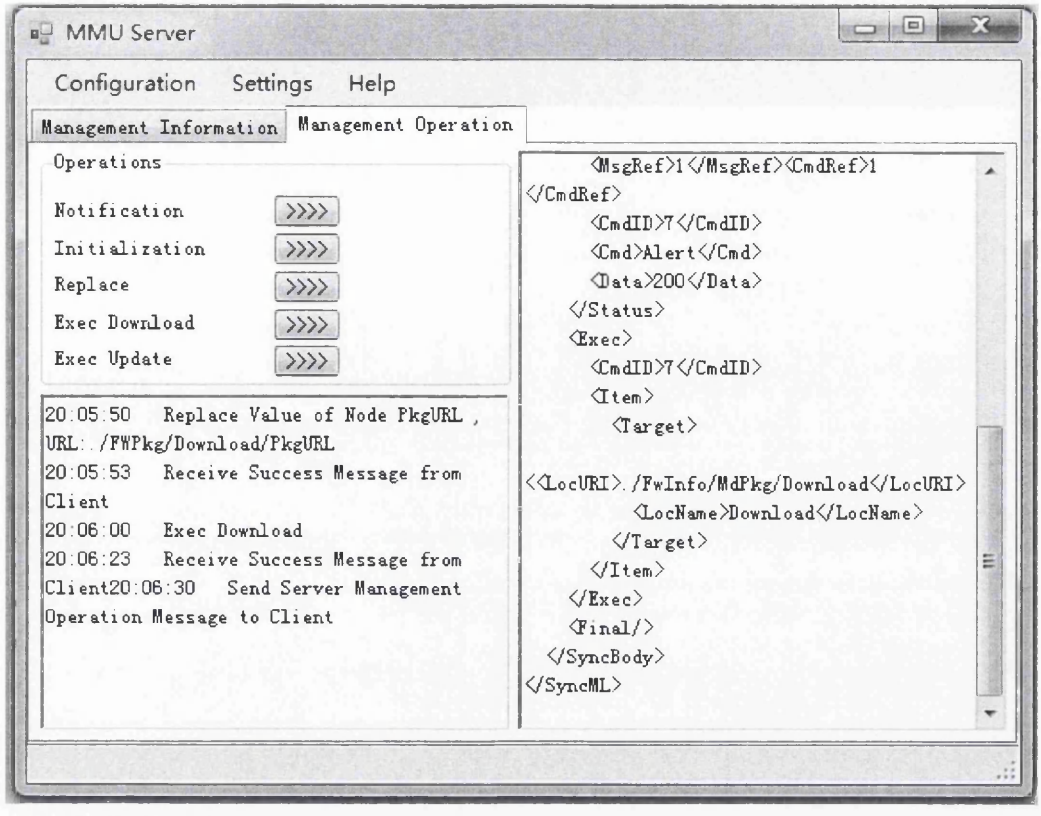

Figure 5.8 Log of Download Phase in MMU Server

Table 5.3 shows the syncML message sent from MMU Server to MMU Client 2 for the "Replace" operation. This message is one of the messages in download phase, which is to replace the attribute of PkgURL node in the path of "./FwInfo/MdPkg/Download/PkgURL". The PkgURL node records the download information of the modulation module for the centralization download method. When the centralization download method is chosen, this download address recorded in PkgURL will be used. As shown in Table 5.3, the name and path as well as the value of the node are included in this message under the command "Replace".

Table 5.3 Message of Configuration of PkgURL Node

```
<SyncML xmlns='SYNCML:SYNCML1.2'>
  <SyncHdr>
    <VerDTD>1.2</VerDTD>
    <VerProto>DM/1.2</VerProto>
    <SessionID>1</SessionID>
    <MsgID>2</MsgID>
    <Target>
    <LocURI>IMEI: 000000000000000</LocURI>
    </Target>
```

```
    <Source>
    <LocURI> http://localhost/Management </LocURI>
    </Source>
  </SyncHdr>
  <SyncBody>
    <Status>
      <MsgRef>2</MsgRef>
      <CmdID>1</CmdID>
      <Cmd>SyncHdr</Cmd>
      <Data>212</Data>
    </Status>
    <Replace>
      <CmdID>6</CmdID>
      <Item>
        <Target>
          <LocURI>./FwInfo/MdPkg/Download/PkgURL </LocURI>
          <LocName> PkgURL </LocName>
        </Target>
        <Data> http://localhost:1234 </Data>
      </Item>
    </Replace>
    <Final/>
  </SyncBody>
</SyncML>
```

Table 5.4 shows the syncML message sent back to MMU Server from MMU Client 2. This message is one of the messages in download phase to inform the MMU Server that the "Replace" operation has been finished successfully. The status code 200 is included in this message under the element "Status".

Table 5.4 Message of Configuration Results of MMU Client 2

```
<SyncML xmlns='SYNCML:SYNCML1.2'>
  <SyncHdr>
    <VerDTD>1.2</VerDTD>
    <VerProto>DM/1.2</VerProto>
    <SessionID>1</SessionID>
    <MsgID>2</MsgID>
    <Target>
      <LocURI> http://localhost/Management </LocURI>
    </Target>
    <Source>
      <LocURI>IMEI: 000000000000000</LocURI>
    </Source>
  </SyncHdr>
  <SyncBody>
    <Status>
      <MsgRef>2</MsgRef>
      <CmdID>1</CmdID>
      <CmdRef>0</CmdRef>
```

```
        <Cmd>SyncHdr</Cmd>
        <Data>200</Data>
      </Status>
      <Status>
        <MsgRef>1</MsgRef>
        <CmdRef>6</CmdRef>
        <CmdID>3</CmdID>
        <Cmd>Replace</Cmd>
        <Data>200</Data>
      </Status>
      <Final/>
    </SyncBody>
</SyncML>
```

Table 5.5 shows the syncML message sent from MMU Server to MMU Client 2 for the "Exec" download operation. This message is one of the messages in download phase, which is to initiate the download execution in mobile devices. The target of the Exec command is the Download node in DM Tree. As shown in Table 5.5, the name and the path of the target node are included in this message under the command "Exec".

Table 5.5 Message of Execution Command for Download

```
<SyncML xmlns='SYNCML:SYNCML1.2'>
  <SyncHdr>
    <VerDTD>1.2</VerDTD>
    <VerProto>DM/1.2</VerProto>
    <SessionID>1</SessionID>
    <MsgID>3</MsgID>
    <Target>
      <LocURI> http://localhost/Management </LocURI>
    </Target>
    <Source>
      <LocURI>IMEI:000000000000000</LocURI>
    </Source>
  </SyncHdr>
  <SyncBody>
    <Status>
      <MsgRef>2</MsgRef>
      <CmdID>1</CmdID>
      <Cmd>SyncHdr</Cmd>
      <Data>200</Data>
    </Status>
      <Exec>
      <CmdID>7</CmdID>
      <Item>
        <Target>
          <LocURI>./FwInfo/MdPkg/Download</LocURI>
          <LocName>Download</LocName>
        </Target>
      </Item>
```

```
    </Exec>
    <Final/>
  </SyncBody>
</SyncML>
```

5.3.2.2 CIC for Decentralized Download

The decentralized download step is related to the step ③ of decentralized download test. After receiving the download operation message from MMU Server, MMU Client 2 or MMU Client 3 starts to execute the operations specified in the message. In the download phase, the "Exec" message received from MMU Server is only for initiating the download operation. Therefore, MMU Clients 2 and MMU Client 3 need to decide which download method is applied in different circumstances. In this section, the results show the operations of CIC and the communications between CIC and MMU Clients 2.

CIC keeps the available software information and is responsible for providing this information to the mobile devices. Figure 5.9 shows the log information of CIC, which presents the operations in CIC and the communication procedures between CIC and MMU Client 2. As shown in Figure 5.9, CIC requests software 024124523, which is the modulation module by broadcasting, and gets the device information of MMU Client 1 carrying the software. When CIC receives the request from MMU Client 2, it sends the information of the MMU Client 1 to MMU Client 2. Therefore, the decentralized download can be applied by MMU Client 2.

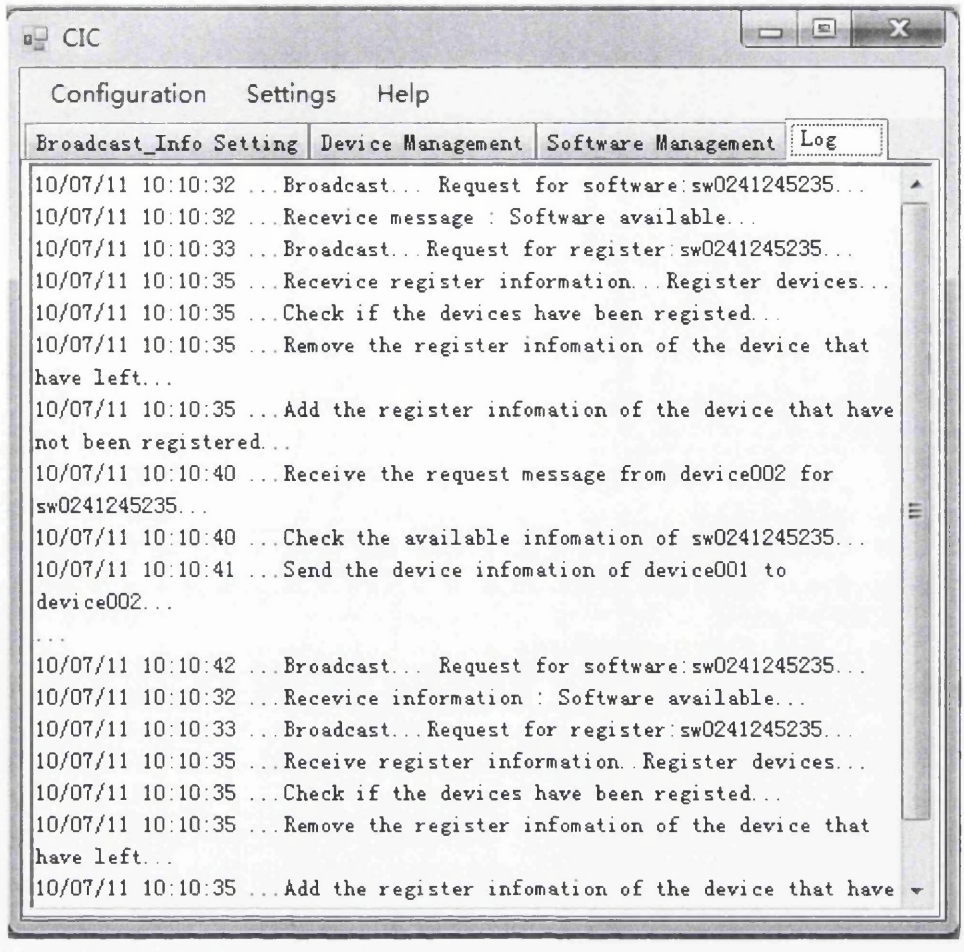

Figure 5.9 Log of CIC

5.3.2.4 Decentralized Download

The decentralized download step is related to the step ④⑤ of decentralized download test. When decentralized download method is chosen, the modulation module will be downloaded from the MMU Client 1 nearby locally. Figure 5.10 shows the log information of the MMU Client 2, which presents the operations executed by MMU Client 2 and the related syncML messages sent back to MMU Server. As shown in Figure 5.10, the download operation in decentralized download method is executed successively.

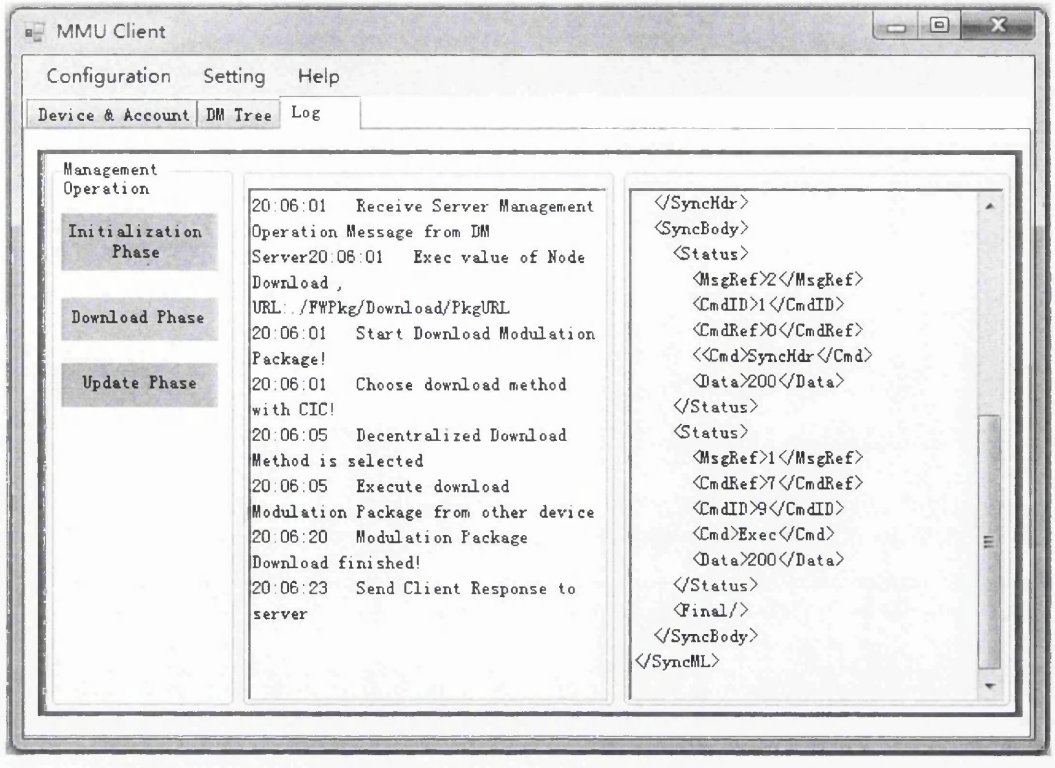

Figure 5.10 Log of Decentralized Download in MMU Client 2

Table 5.6 shows the syncML message sent back to MMU Server from MMU Client 2. This message is one of the messages in download phase to inform the MMU Server that centralization download has been finished successfully. The status code 200 is included in this message under the element "Status".

Table 5.6 Message of Decentralization Download Results of MMU Client 2

```
<SyncML xmlns='SYNCML:SYNCML1.2'>
  <SyncHdr>
    <VerDTD>1.2</VerDTD>
    <VerProto>DM/1.2</VerProto>
    <SessionID>1</SessionID>
    <MsgID>2</MsgID>
    <Target>
      <LocURI> http://localhost/Management </LocURI>
    </Target>
    <Source>
      <LocURI>IMEI:000000000000000</LocURI>
    </Source>
  </SyncHdr>
  <SyncBody>
    <Status>
      <MsgRef>2</MsgRef>
      <CmdID>1</CmdID>
      <CmdRef>0</CmdRef>
```

```
        <Cmd>SyncHdr</Cmd>
        <Data>200</Data>
      </Status>
      <Status>
        <MsgRef>1</MsgRef>
        <CmdRef>5</CmdRef>
        <CmdID>9</CmdID>
        <Cmd>Exec</Cmd>
        <Data>200</Data>
      </Status>
      <Final/>
    </SyncBody>
</SyncML>
```

5.3.2.3 Centralized Download

This section is related to the step ④⑤ of centralized download test. When centralized download method is chosen, the modulation module will be downloaded from the content server whose address has been obtained by the "Replace" operation in the earlier message. Figure 5.11 shows the log information of the MMU Client 3, which presents the download operations executed by MMU Client 3 and the related syncML messages sent back to MMU Server. As shown in Figure 5.11, the download operation in centralized download method is executed successively.

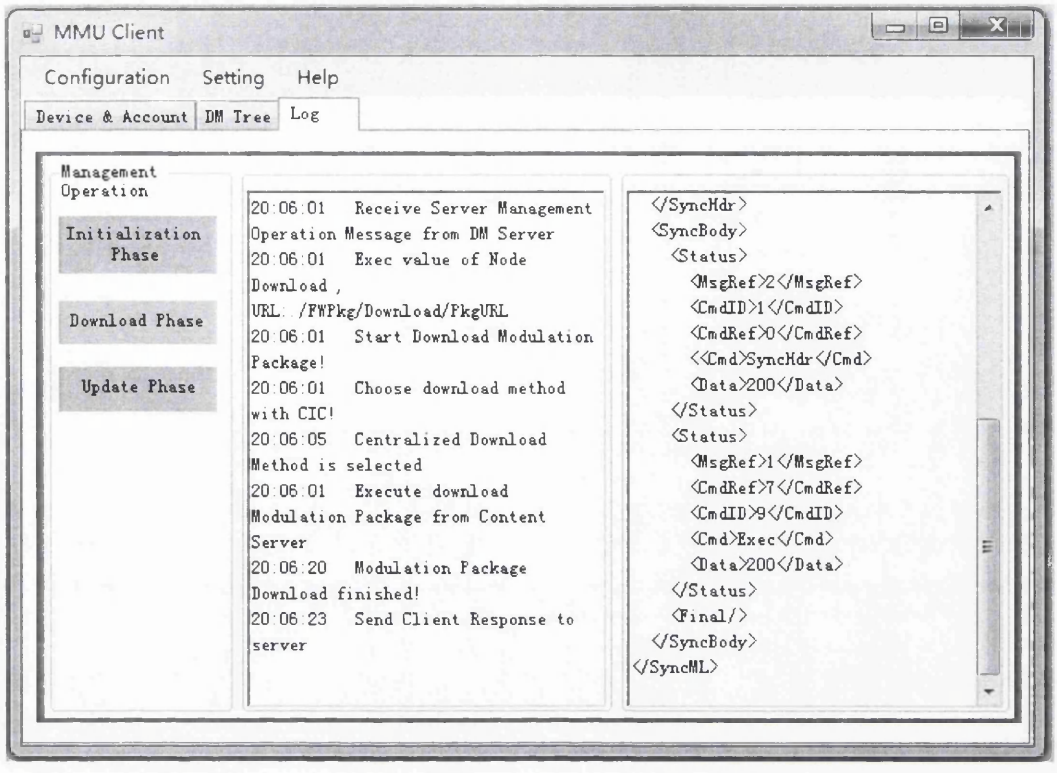

Figure 5.11 Log of Centralized Download in MMU Client 3

Table 5.7 shows the syncML message sent back to MMU Server from MMU Client 3. This message is one of the messages in download phase to inform the MMU Server that centralization download operation has been finished successfully. The status code 200 is included in this message under the element "Status".

Table 5.7 Message of Centralization Download Results of MMU Client 3

```
<SyncML xmlns='SYNCML:SYNCML1.2'>
  <SyncHdr>
    <VerDTD>1.2</VerDTD>
    <VerProto>DM/1.2</VerProto>
    <SessionID>1</SessionID>
    <MsgID>2</MsgID>
    <Target>
      <LocURI> http://localhost/Management </LocURI>
    </Target>
    <Source>
      <LocURI>IMEI:000000000000000</LocURI>
    </Source>
  </SyncHdr>
  <SyncBody>
    <Status>
      <MsgRef>2</MsgRef>
      <CmdID>1</CmdID>
      <CmdRef>0</CmdRef>
```

```
        <Cmd>SyncHdr</Cmd>
        <Data>200</Data>
      </Status>
      <Status>
        <MsgRef>1</MsgRef>
        <CmdRef>7</CmdRef>
        <CmdID>9</CmdID>
        <Cmd>Exec</Cmd>
        <Data>200</Data>
      </Status>
      <Final/>
    </SyncBody>
</SyncML>
```

5.3.3Update Phase

The update phase is related to the step ⑥ of both centralized download test and decentralized download test. Due to the update phase are the same as the two tests, we only present the test results between MMU Server and MMU Client 2 in this section.

5.3.3.1 MMU Server

As mentioned, the task of update phase is to finish the update execution of modulation module. The precondition of update execution is that the download execution has been finished successfully. That means the modulation module package is downloaded into the mobile device. Figure 5.12 shows the log information of the MMU Server, which presents the operations defined by MMU Server and the related syncML messages send to MMU Client 2. As shown in Figure 5.12, the update operation is initiated by the command "Exec" on node Update.

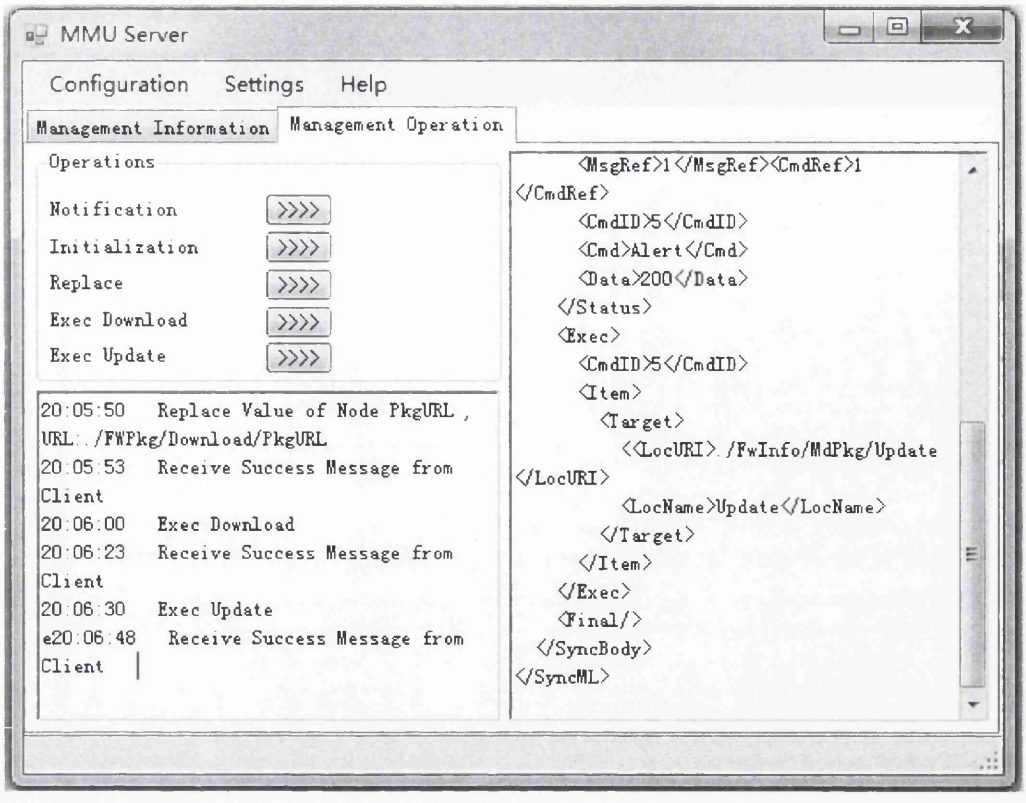

Figure 5.12 Log of Update Phase in MMU Server

Table 5.8 shows the syncML message sent from MMU Server to MMU Client 2 for the "Exec" update operation. This message is one of the messages in update phase, which is to initiate the update execution in mobile devices. The target of the Exec command is the Update node in DM Tree. As shown in Table 5.8, the name and the path of the target node are included in this message under the command "Exec".

Table 5.8 Message of Execution Command for Update

```
<SyncML xmlns='SYNCML:SYNCML1.2'>
  <SyncHdr>
    <VerDTD>1.2</VerDTD>
    <VerProto>DM/1.2</VerProto>
    <SessionID>1</SessionID>
    <MsgID>3</MsgID>
    <Target>
      <LocURI> http://localhost/Management </LocURI>
    </Target>
    <Source>
      <LocURI>IMEI:000000000000000</LocURI>
    </Source>
  </SyncHdr>
  <SyncBody>
    <Status>
      <MsgRef>2</MsgRef>
```

```
        <CmdID>1</CmdID>
        <Cmd>SyncHdr</Cmd>
        <Data>200</Data>
      </Status>
        <Exec>
        <CmdID>5</CmdID>
        <Item>
          <Target>
            <LocURI>./FwInfo/MdPkg/Update</LocURI>
            <LocName>Update</LocName>
          </Target>
        </Item>
      </Exec>
      <Final/>
    </SyncBody>
</SyncML>
```

5.3.3.2 MMU Client

After receiving the operation message from MMU Server, MMU Client 2 starts to execute the update operation specified in the message. The update of modulation module is realized through the interface provided hardware abstract layer. Figure 5.13 shows the log information of the MMU Client 2, which presents the update operations executed by MMU Client 2 and the related syncML messages sent back to MMU Server. As shown in Figure 5.13, the update operation is executed successively.

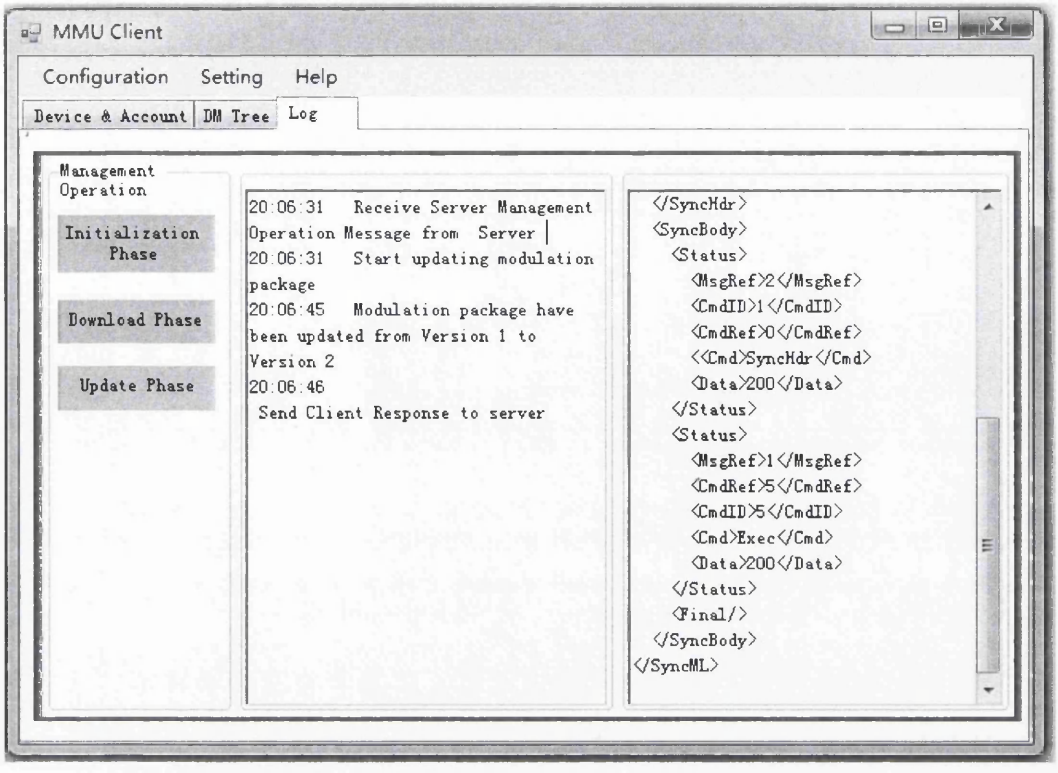

Figure 5.13 Log of Update in MMU Client 2

Table 5.9 shows the syncML message sent back to MMU Server from MMU Client 2. This message is one of the messages in update phase to inform the MMU Server that the "Exec" update operation has been finished successfully. The status code 200 is included in this message under the element "Status".

Table 5.9 Message of Update Results of MMU Client 2

```
<SyncML xmlns='SYNCML:SYNCML1.2'>
  <SyncHdr>
    <VerDTD>1.2</VerDTD>
    <VerProto>DM/1.2</VerProto>
    <SessionID>1</SessionID>
    <MsgID>2</MsgID>
    <Target>
      <LocURI> http://localhost/Management </LocURI>
    </Target>
    <Source>
      <LocURI>IMEI:000000000000000</LocURI>
    </Source>
  </SyncHdr>
  <SyncBody>
    <Status>
      <MsgRef>2</MsgRef>
      <CmdID>1</CmdID>
      <CmdRef>0</CmdRef>
```

```
        <Cmd>SyncHdr</Cmd>
        <Data>200</Data>
      </Status>
      <Status>
        <MsgRef>1</MsgRef>
        <CmdRef>5</CmdRef>
        <CmdID>9</CmdID>
        <Cmd>Exec</Cmd>
        <Data>200</Data>
      </Status>
      <Final/>
    </SyncBody>
</SyncML>
```

5.4 Summary

In this chapter, we simulated the server side and the client side of the framework as well as the CIC to test and verify the framework we proposed for the modulation module OTA upgrade. The test and the verification results in this chapter show the feasibility and accuracy of MMU.

6. Conclusion and Future Work

In this chapter, we conclude the contributions and present the possible future work of this thesis.

6.1 Conclusion of Contributions

In this thesis, a framework MMU for modulation module upgrade in mobile devices based on OMA DM Standard is proposed. This thesis presents the overall design of MMU, including the design of management architecture, the management object and the management operations. A combined download mechanism which combines the decentralized download method and centralized download method is proposed, which allows the firmware not only to be downloaded from central servers but also to be passed from one device to another. The implementation of MMU is elaborated from a coding point of view in this thesis, including the DM Server side and the Agents of the Client side in mobile devices. Moreover, we simulated the server side and the client side of the framework as well as the CIC to test and verification the framework we proposed for the modulation module OTA upgrade. The test and the verify results show the feasibility and accuracy of MMU.

6.2 Future Work

In this thesis, we propose a framework MMU for modulation module upgrade in mobile devices based on OMA DM standard. Our research focuses on the software upgrade on mobile devices especially the modulation module, which lays particular emphasis on the management and realization of the software download and update between server side and mobile devices. Although our research is based on the reconfigurable feature provided by SDR technology, it has not directly referred to the lower layer development of SDR system and the implementation of MMU is still in a simulation phase. Therefore, the future work can be related to the lower layer implementation.

Bibliography

[1] A. Sandeep, M. Sangita, et al, "Universal Manager: seamless management of enterprise mobile and non-mobile devices," 2004 IEEE International Conference on Mobile Data Management, pp.320-331, 2004.

[2] F. Siddiqui, and S. Zeadally, "Mobility management across hybrid wireless networks: Trends and challenges, Computer Communications," 29(9), pp.1363-1385, 2006.

[3] J. Hoffmeyer, P. Il-Pyung, et al, "Radio software download for commercial wireless reconfigurable devices, Communications Magazine," IEEE, 42(3), pp. S26-S32, 2004.

[4] "A technology overview of Software Defined Radio," Wipro Technologies White Paper, August 2002

[5] "Underatanding firmware over the air-FOTA," Innopath White Paper, 2007

[6] "The Critical Role of Interoperability standards for mobile device management," Innopath White Paper, 2007

[7] U. Ramacher, "Software Defined Radio Prospects for Multi-standard Mobile Phones," IEEE, 40(10), pp.62-69, October 2007

[8] 3GPP. http://www.3gpp.org/

[9] John D.Ralston, Peter G. Bier, "The emergence of software phone," SDR Forum, 1998

[10] SDR Forum. http://www.sdrforum.org

[11] "SDRF Cognitive Radio Definitions," SDR Forum working document, November, 2007

[12] A. Kountouris, C. Moy, "Reconfiguration in software radio systems", Karlsruhe Workshop on Software Radios, 2002

[13] J.P. Delahaye, G. Gogniat, et al, "Software radio and dynamic reconfiguration on a DSP/FPGA platform", in Rykaczewski, P. and Schmidt, M. (Eds.): in special issue on Software Defined Radio of Frequenz, May-June, No. 58, pp.152–159, 2004

[14] "Base Station System Structure", SDR Forum Document, January. 2002

[15] J.P. Delahaye, C. Moy, P. Leray, J. Palicot, "Managing dynamic partial reconfiguration on heterogeneous SDR platforms", SDR Forum, November,2005

[16] J.P. Delahaye, C. Moy, P.Leray, J. Palicot, "Partial Reconfiguration of FPGAs for Dynamical Reconfiguration of a Software Radio Platform", in Proc. of IST Mobile and Wireless Communications Summit, Budapest, Hungary, June 2007

[17] "SDR Market Study Task 1: Market Segmentation and Sizing", SDR Forum Document, SDRF-05-A-0003-V0.00,March, 2005

[18] P. Sedcole, B. Blodget, T. Becker, J. Anderson, P. Lysaght, "Modular dynamic reconfiguration

in Virtex FPGA," Computers and Digital Techniques, IEE Proceedings - , vol.153, no.3, pp. 157-164, 2 May 2006

[19] M. Ullmann, M. Hübner, B. Grimm, J. Becker, "A FPGA Run-Time System for Dynamical On-Demand Reconfiguration," In Proc. of the 18th Int. Parallel and Distributed Processing Symposium, 2004

[20] A.P.Ershow, "Fourth-generation software", Cybernetics and systems analysis,1973

[21] Sharon Peleg, "Redefining FOTA deployment in EMEA--FOTA's role in user experience management," Wireless Informatics Forum.

[22] C. Andreas, L. Matt, " Firmware Over-the-air: From Hype to Market Reality", ARCchart research paper sponsored by Red Bend Software, October,2006

[23] "Firmware Over the Air (FOTA) and Mobile Device Management (MDM)", HP White Paper, March 2009

[24] "Firmware Management: Using Wireless Informatics to solidify the business case", Wireless Informatics Forum, 2007

[25] Sharon Peleg, "Principles of Updating Mobile Firmware Over-the-Air (FOTA)", Red Bend Software White Paper.

[26] "Key Criteria for Evaluating Technologies for Effective Firmware Over-The-Air (FOTA) Updating of Mobile Phones", Red Bend Software White Paper.

[27] R. Chakravorty, H. Ottevanger, "Architecture and Implementation of a Remote management framework for dynamically reconfigurable devices," Proc. of the 10th IEEE International Conference on Networks, pp.375-381, 2002.

[28] Shi, Z., C. Dolwin, et al. "An OMA DM based Software Defined Radio Proof-of-Concept Demonstration Platform," Personal, IEEE 18th International Symposium on Indoor and Mobile Radio Communications, pp.1-5, 2007.

[29] Open Mobile Alliance. http://www.openmobileallianve.org

[30] S. Husain, T. Alonso, et al. "Remote device management of WiMAX devices in multi-mode multi-access environment," IEEE International Symposium on Broadband Multimedia Systems and Broadcasting, pp.1-14, 2008.

[31] Ma, J., J. Liao, et al. "Device Management in the IMS, Journal of Network and Systems Management," 16(1), pp.46-62, 2008.

[32] B. Steinke, and K. Strohmenger. "Advanced Device self Management through Autonomics and Reconfigurability, Mobile and Wireless Communications Summit," 16th IST. pp.1-4, 2007.

[33] Open Mobile Alliance, "OMA Device Management Standardized Objects", 2007.

[34] Open Mobile Alliance, "OMA Device Management Tree and Description", Version 1.2, 2008.

[35] IETF. "Uniform Resource Identifiers (URI)", RFC 2396, 1998

[36] Open Mobile Alliance, "OMA Device Management Tree and Description Serialization", Version 1.2, 2005.

[37] Open Mobile Alliance, "OMA Device Management Representation Protocol", Version 1.2, 2005.

[38] Open Mobile Alliance, "SyncML Representation Protocol", Version 1.2.1, 2007

[39] Open Mobile Alliance, "OMA Device Management Protocol", Version 1.2.1, 2008

[40] Open Mobile Alliance, "Firmware Update Management Object", Version 1.0.2, 2009

[41] Open Mobile Alliance, "Device Management Architecture", Version 1.3, 2009

[42] R. State, O. Festor, et al., "An extensible agent toolkit for device management," Proc. of the 2004 IEEE/IFIP Int'l Symp. on Network Operations and Management Symposium, pp.845-858, 2004

[43] K. Joon-Myung, J. Hong-Taek, et al., "OMA DM-based remote software fault management for mobile devices," Int. J. Netw. Manag, 19(6), pp.491-511, 2009

[44] L. Hun-Jung, P. Seon-Ho, et al., "u-MoDEM : ubiquitous Mobile Device Environment Manager based on OMA-DM," The 10th International Conference on Advanced Communication Technology, pp.283-287, 2008.

[45] S. Jaeyoung, C. Youngwoo, et al., "Design and implementation of the management agent for mobile devices based on OMA DM," Proceedings of the 2nd international conference on Ubiquitous information management and communication, 2008

[46] Open Mobile Alliance. "Firmware Update Management Object Architecture", Version 1.0, 2007

[47] Oommen, P. "A framework for integrated management of mobile-stations over-the-air," 2001 IEEE/IFIP International Symposium on Integrated Network Management Proceedings, pp.247-256, 2001.

[48] D. Krishnaswamy, T. Pfeifer, et al. "OMA DM Based Remote RF Signal Monitoring of Mobile Devices for QoS Improvement," Real-Time Mobile Multimedia Services, Springer Berlin/Heidelberg, pp.76-87, 2007.

[49] Kang, JM, Ju HT, Choi MJ, Hong JWK, "OMA DM Based Remote Software Debugging of Mobile Devices", Lecture Notes in Computer Science, 4773: 51 - 61, 2007

[50] H.Zhang, H.X.Wang, et al., "An OMA DM Based Framework for Updating Modulation Module for Mobile Devices, " International Journal of Adaptive, Resilient and Autonomic Systems, 2(3), pp. 13-23, 2011

[51] Open Mobile Alliance, "Device Management HTTP Binding", Version 1.3, 2010