

# 國立交通大學

## 資訊科學系

### 碩士論文

一個鉅量多人連線遊戲開發及管理系統架構

A Framework of MMOG Development and Management

System

研究生：蘇科旭

指導教授：袁賢銘 教授

中華民國九十三年六月

一個鉅量多人連線遊戲開發及管理系統架構

A Framework of MMOG Development and Management System

研究生：蘇科旭

Student : Ko-Hsu Su

指導教授：袁賢銘

Advisor : Shyan-Ming Yuan

國立交通大學

資訊科學研究所



A Thesis

Submitted to Institute of Computer and Information Science

College of Electrical Engineering and Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer and Information Science

June 2004

Hsinchu, Taiwan, Republic of China

中華民國九十三年六月

# 一個鉅量多人連線遊戲開發及管理系統架構

研究生:蘇科旭

指導教授: 袁賢銘

國立交通大學資訊科學研究所

## 摘要

近年來，隨著多人線上遊戲市場不斷的擴大，有越來越多的廠商都相繼的投入大量的資源來進行線上遊戲的研發，所以整個遊戲開發及伺服器管理方式的研究變的越來越重要。根據目前在多人線上遊戲開發過程普遍被使用到的訊息導向多人線上遊戲平台，本篇論文主要提出一個建構於平台之上的開發及管理系統的架構，讓整個開發以及管理的工作變的更簡單且更具有彈性。

# **A Framework of MMOG Development and Management System**

Student:Ko-Hsu Su

Advisor: Shyan-Ming Yuan

Department of Computer and Information Science  
National Chiao Tung University

## **Abstract**

In recent years, by the massively multiplayer online game (MMOG) market become growing continuously, there are more and more game development factories to invest large resource in MMOG development. So the MMOG development and management system research become more important. According to the message oriented MMOG platform which be commonly used in the development flow, this paper present a framework of development and management system under the MMOG platform. Under the framework, the MMOG development and management work will become more simply and with more elasticity

## ACKNOWLEDGEMENTS

本篇論文的完成，首先我要感謝我的指導教授 袁賢銘教授 給予我許多寶貴的意見以及思考方向，還有分散式系統實驗室博士班學長 鄭明俊 吳瑞祥 高子漢 提供給我許多的意見，以及其他實驗室的同仁給予我程式開發上的協助，讓我可以順利的完成碩士論文的研究。

另外 我要特別感謝一同參與研究計畫的 蕭存喻學長 以及 陸振恩同學，在我論文研究期間，常常向你們請教有關於程式設計及論文寫作上的問題，很感謝你們在自己研究之餘能夠給予我適時的幫忙及協助，讓我學習了許多書本上學習不到的東西。

最後我要感謝我的父母 蘇惠敏 和 黃甚 以及我的兩位兄長 冠中 和 冠華 還有我的女友 怡容，感謝你們給予我物質及精神上最大的鼓勵，讓我可以堅持我的研究，願與你們分享這份榮耀及喜悅。



# CONTENTS

<b>ABSTRACT IN CHINESE</b> .....	I
<b>ABSTRACT IN ENGLISH</b> .....	II
<b>ACKNOWLEDGEMENTS</b> .....	III
<b>CONTENTS</b> .....	IV
<b>LIST OF FIGURES</b> .....	VI
<b>CHAPTER 1 INTRODUCTION</b> .....	1
<b>1.1 MMOG PLATFORM OVERVIEW</b> .....	1
<b>1.2 MMOG DEVELOPMENT</b> .....	4
<b>1.2.1 DEVELOPER’S VIEW FOR MMOG</b> .....	5
<b>1.2.2 MMOG SERVER MANAGEMENT</b> .....	6
<b>1.3 OVERVIEW</b> .....	6
<b>CHAPTER 2 FRAMEWORK DESIGN</b> .....	8
<b>2.1 MMOG DEVELOPMENT SYSTEM</b> .....	8
<b>2.1.1 DESCRIBE THE MMOG CONTENT BY XML</b> .....	10
<b>2.1.2 CODE GENERATION</b> .....	12
<b>2.2 MMOG MANAGEMENT SYSTEM</b> .....	13
<b>2.2.1 JMX OVERVIEW</b> .....	14
<b>2.2.2 MMOG MANAGEMENT SYSTEM DESIGN</b> .....	16
<b>CHAPTER 3 SYSTEM IMPLEMENTATION</b> .....	19
<b>3.1 DOIT PLATFORM OVERVIEW</b> .....	19
<b>3.2 DEVELOPMENT SYSTEM IMPLEMENTION</b> .....	20
<b>3.3 SERVER MANAGEMENT SYSTEM IMPLEMENTATION</b> .....	27
<b>3.3.1 GAME SERVER LAYER</b> .....	28
<b>3.3.2 CENTRAL MANAGEMENT SERVER LAYER</b> .....	32

3.3.3	MANAGER LAYER.....	35
CHAPTER 4	CONCLUSIONS .....	36
CHAPTER 5	FUTURE WORKS.....	38
BIBLIOGRAPHY	.....	40



# LIST OF FIGURES

<i>Figure 1-1 the message-oriented MMOG Platform architecture.....</i>	<i>3</i>
<i>Figure 1-2 Developer's view for MMOG Development.....</i>	<i>5</i>
<i>Figure 2-1 MMOG message script document.....</i>	<i>11</i>
<i>Figure 2-2 MMOG message xml schema document.....</i>	<i>12</i>
<i>Figure 2-3 MMOG Code generation flow.....</i>	<i>13</i>
<i>Figure 2-4 JMX Management Architecture.....</i>	<i>15</i>
<i>Figure 2-5 MBean Server Invoke Mechanism.....</i>	<i>15</i>
<i>Figure 2-6 MMOG Management System Architecture.....</i>	<i>16</i>
<i>Figure 3-1 DOIT Platform Architecture.....</i>	<i>19</i>
<i>Figure 3-2 DOIT Platform Inner Message Process Flow.....</i>	<i>21</i>
<i>Figure 3-3 Examples of the Message Format.....</i>	<i>22</i>
<i>Figure 3-4 The Code of MoveMessage and LoginMessage.....</i>	<i>23</i>
<i>Figure 3-5 The Code of LoginMessageHandler and LoginMessageFactory.....</i>	<i>26</i>
<i>Figure 3-6 DOIT Platform Code Generation Engine.....</i>	<i>27</i>
<i>Figure 3-7 DOIT Platform Management System Architecture.....</i>	<i>28</i>
<i>Figure 3-8 Management Plug-in Work Flow.....</i>	<i>29</i>
<i>Figure 3-9 Central Management Server Work Flow.....</i>	<i>32</i>
<i>Figure 3-10 Overview of the ServerManagementFactory Class.....</i>	<i>34</i>



# **CHAPTER 1 INTRODUCTION**

## **1.1 MMOG PLATFORM OVERVIEW**

MMOG is an acronym that stands for Massive Multiplayer Online Game. This is the kind of online game that allows a huge number of players (greater than 1,000) to play in the same game concurrently. Unlike other small online games which the game states are maintained by a few players, the MMOG states are maintained by all of the players. There are some kinds of MMOG like MMORPG (Massively Multiplayer Online Role-Playing Game) and MMOFPS (Massively Multiplayer Online First Person Shot) and etceteras.

According to different kinds of MMOG, the necessary of game server's Scalability, Availability, Flexibility and Simplicity will be different. Each player in the game world can make interaction with each others by making move, trade, attack or other actions. That is all the requests from the clients will be received by servers and will be processed by game servers immediately, and client will receive the server's response. The development of high performance MMOG server has become one of the most important thing when design a MMOG. Therefore the MMOG Platform has been Introduced.

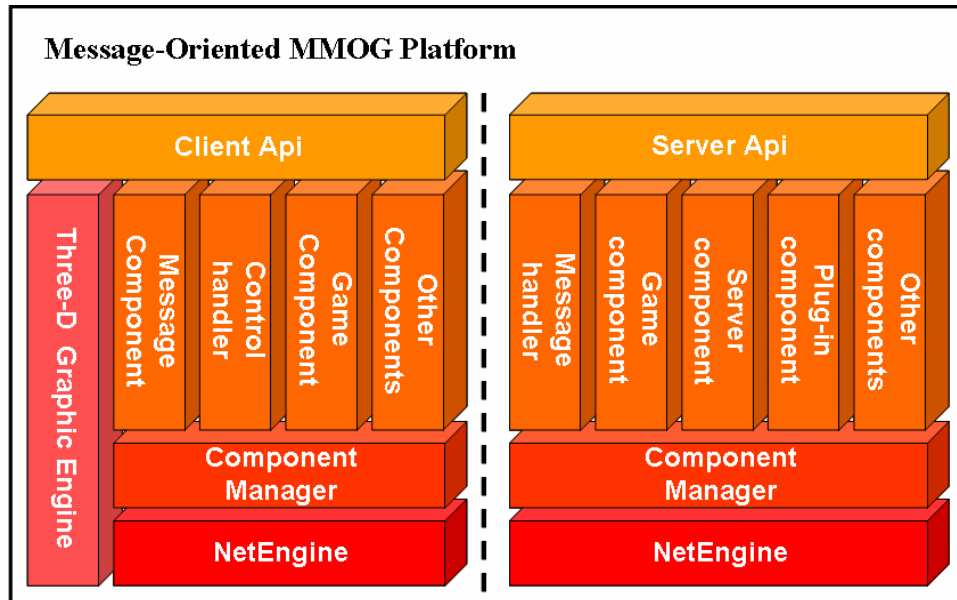
The MMOG Platform is a middleware solution which can solve the connection problem between client and server, and it also provide a effective inner message process mechanism and handle fault tolerance. So it will reduce the game developer's load, they just only implement the message protocols, NPC logics, MAP configure file, server configure file and then insert the game logics into the message handlers, then the server could be work. It can curtail the development time, and the game

developers can keep their mind on game content design. Therefore, the MMOG Platform plays a very important role of a MMOG, whether it has powerful ability or not it will directly affect the full game's performance. The MMOG Platform not only can be used in MMOG development, but also can be used for a message transformation middleware, so it can be used in other applications generally. For this reason, there are many factories invest large resource in MMOG Platform research like [ZONA], [BIGWORLD] and etc, and they put their focus on several topics like easy of development, easy of deployment and easy of maintenance.

*Figure 1-1* shows a normal message-oriented MMOG Platform framework. The message oriented is mean that the connection between server and client is accomplished by message transformation. At the server side the most underlay component is NetEngine, it is responsible for the physical network transformation work. And above the NetEngine ,the component manager provide components management service, all of the components must register to the manager and could be obtain by manager's component lookup. The message handler will process the message which sending from client and pass through NetEngine to handler. The game component contain some information like game objects, game NPCs, game map information and other game content information.

The server component contains some service like timer service, load balance service, log service and other services about server performance. The Plug-in component can be any game based plugging like server management Plug-in, encryption Plug-in. The other components are platform based components and they may register to the manager like fault tolerance component, persistent database storage component and other components defined by developer. The server API is a set of API which

provided by platform include the component lookup API, server setup API, and server management API. By using the server API the developer can implement the server control and management applications.



*Figure 1-1 the message-oriented MMOG Platform architecture*

At the client side the most underlay component is NetEngine too, and it also have a component manager which doing the same work as server side component manager. The 3D graphic engine is responsible for the 3D model display draw, and the message component defines the message protocols. The control handler will handle the entire player's control action which produced by the mouse action or keyboard action or others, and according to different control action that will lead to a kind of message be produced and then be sent to server. The game component is game base component, it include the game rule, role, event, object, environment and sound files or information which the player action will reference to. Other components can be a game plug-in or encryption program, or other platform based component. The client API defines the development API like 3D Engine and all components control API.

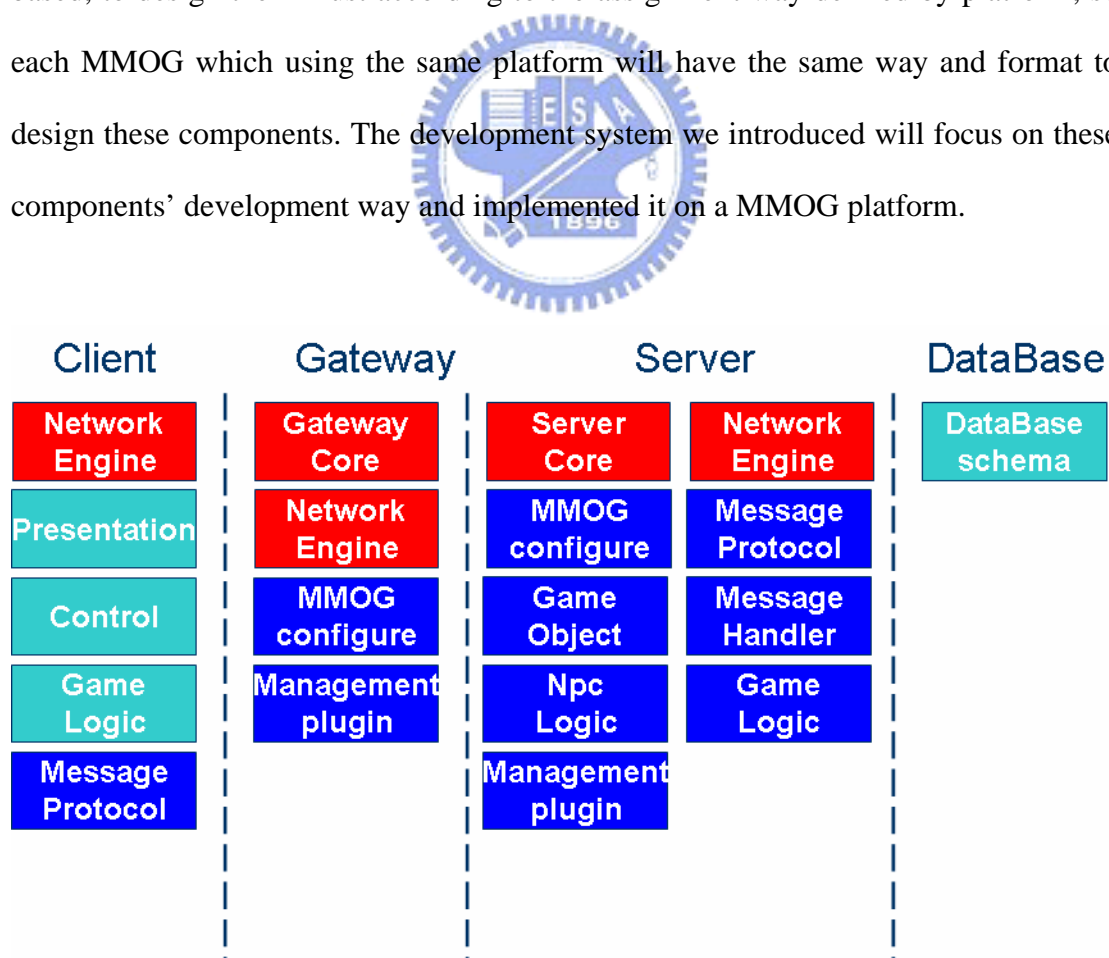
## 1.2 MMOG DEVELOPMENT

The total workload of a MMOG development is very huge and need various developers like network programmers, art designers, musicians, and game contents designers and system maintenance workers. It can be division into four parts there are client, gateway, server and database. Each part of the development must focus on different characteristics. The client's development focuses on the presentation, control interface, encryption mechanism and the game content. The gateway's and server's development focus on high performance NetEngine, message processor, load balance mechanism, and fault tolerance. And database focuses on data reliability and persistency storage.

The development way could be divided in two ways. First, all of the programs from client to server are developed by the game designer themselves. According to the method, it will cost lots of time for whole work, and the performance is not good. Another way is to use the MMOG platform to design a MMOG. We can use the platform's development system to develop. The development time will be reduced and we will get better server performance. According to economic benefits, using MMOG platform to develop will be a better choice. The development system's functionality and working flow will be different based on different platform. A useful MMOG development system should have several feature like easy to use, background is not necessary, have strong functions and with flexibility. Therefore, this paper introduces a development system framework for the MMOG platform; we will use XML to edit a MMOG content description document and also design a code generation engine, then we can generate the code by load the document into the engine. Therefore, the whole MMOG development works will become more simply, fast and elasticity.[3]

### 1.2.1 DEVELOPER'S VIEW FOR MMOG

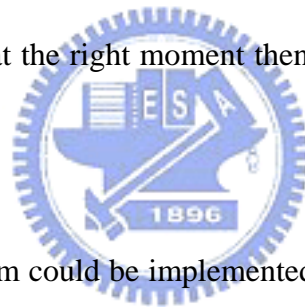
*Figure 1-2* is the developer's view for MMOG development, all components in this figure must be designed when develops the MMOG. The red components are provided by the MMOG platform, and the developer will not necessary to pay attention to their operation mechanism, so they can be used directly. Include the client core, gateway core, server core and network engine components and they are the most important components for the MMOG to affect the whole performance. The light blue components are game based components, according to different kinds of MMOG they will have different way to be designed, including the client side presentation, control, game logic and database components. And the deep blue components are platform based, to design them must according to the assignment way defined by platform, so each MMOG which using the same platform will have the same way and format to design these components. The development system we introduced will focus on these components' development way and implemented it on a MMOG platform.



*Figure 1-2 Developer's view for MMOG Development*

## 1.2.2 MMOG SERVER MANAGEMENT

After the MMOG environment has been built up, the server management will be an important work. The whole MMOG is a distributed environment, it is composed by game servers and gateways. So the MMOG's management will have some distributed environment problems. For example, as the player's count grows up continuously and inner mistakes are produced, the server performance will drop down or even crash. Under this situation, we must record each server's state continuously, and make some system maintenance mechanism according to different server's state. So in order to handle the above-mentioned problems, the server management system was presented. The server management system is a central management architecture and it provides the functions of server lookup and modulate, we can use them to log the servers' states and modulate at the right moment then we can keep the servers on the best state.



The server management system could be implemented by many ways. In the past, the management system was implemented according to different customer's definition, and we don't have a standard management way. This paper presents a JMX (Java Management extensions) based management system. JMX provides standard management architecture for distributed systems. We use a two layer JMX invoke mechanism to design a central management system framework, and also implement it on a MMOG environment. [4]

## 1.3 OVERVIEW

In chapter 2, the development and management system are presented. Based on the systems we presented, we implement them on the DOIT MMOG platform, and we will describe the design details for the system's work flow and architecture in chapter

3. And in chapter 4, we will make some concludes and discuss the future works of this system at the chapter 5.



## **CHAPTER 2    FRAMEWORK DESIGN**

We will introduce the framework design of the MMOG development and management system under the MMOG platform in this chapter. At the first section, we will discuss the general MMOG development flow and then we will present a development system to improvement it by using XML to describe the game contents and auto generates the code. The next section, we will present a JMX based management system, and discus the advantages based on the system architecture and the work flow.

### **2.1 MMOG DEVELOPMENT SYSTEM**

In the general MMOG development flow, the first step is to define the game contents like game roles, environment, events, game objects, NPCs and game playing scenario. After decide the game contents, the next step is to develop the server side and client side code. In order to develop these code by using the platform's API, the programmers must precede the developer's teaching course to learning how to development. That is developers must know the programming language and the functionality of the APIs which the platform assignment and provide, and then they will start to develop the code according to the definition of the game contents. And the game content designers must define a content script document thyself which tells the programmers what should they do, and the developers must understand that what the document mean is.

Under this development flow, we find some parts that can be replaced by other way and will increase the development speed and reduce the development complexity. First, the game content description document format should be provided by the platform that uses a popular language to define. Therefore, the game content designers



could write the description document by using the language and the programmers could understand the document's content more easy and it also reduce the communication time between content designers and programmers which wasted on talk about the content's definition.

The second, we find that the programmer will write a lot of the duplicate code during the development process. Because the development of the MMOG component's program will follow an architecture which assigned by the platform and these programs may have the some methods or attributes are the same. Under this condition, the same MMOG component's programs will have a part of codes are similarity and the differentiations between them are the method operation logic and other attributes which are defined by themselves. Therefore, we draw out the similarity part of the programs and use the XML description document to describe the different parts like methods and attributes. Then we design a code generation engine according to the document's format and the regular of the programs. The code generation engine will generate the code according to the distribution of the document. Therefore, we can load the document into the code generation engine, and then the programs will be generated automatically. At last, the programmers must insert the code into the programs which can't be generated by the engine, and then the development work will be finished. According to the method, the whole development flow will be simplified and the development speed will be more fest.

Therefore, the design of the MMOG development system can be partition into some parts. First, we must analyze all of the programs which must be development at the game server. After we get the relationship and programming rule between these programs, we define the description data structure and also choice a description

language to describe these programs. At last, we design a code generation engine according to the relationship between the programs and the description language.

### 2.1.1 DESCRIBE THE MMOG CONTENT BY XML

A MMOG content description language should have some features like be suitable for describe and process the data, easy to be edited, support multi language code and with the verify mechanism. Therefore, we choose the XML (eXtensible Markup Language) to describe the MMOG contents, because XML is specially design for data description and exchange, and it is self-describing (the markup describes the structure and type names of the data, although not the semantics) and the XML file is a text file which could be edit by any text edit software, and it also support multi language code, and by using a XML schema file we could verify the XML document whether it is valid or not.



The MMOG contents description could be partition into some parts like messages; NPCs, MAPs, Game objects, Game logic, server configure files and others. Each kind of the data will have their own data structure, and all of these data structure can be representing by the data structure tree. Therefore, we defined the data tags at first, and then build the data structure tree according to the relationship between these data. Then the content designer can write the description document follow this data structure. In order to verify the document if it is valid, we will design a XML schema file to implement this work.

*Figure 2-1* is a XML file which describes the MMOG messages includes the version, package name, class path, and each message's contents. Each message represents a kind of player's action; according to different player actions we need a kind of

message to handle this action. Therefore, we define some messages in this XML document, and use a MMOG message xml schema document to verify it in the *Figure 2-2*. According to the method, the content designer could define data follow a standard structure, and will be more convenient to search or edit the document.

```

<?xml version="1.0" encoding="utf-8" ?>
- <MMOG_Message xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="mmog_message.xsd">
  <Version>1.0</Version>
  <PackageName>mmog.message</PackageName>
  <Classpath>C://sdk_test/</Classpath>
- <Messages>
- <Message>
  <MessageName>LoginMessage</MessageName>
  <MessageType>0x01</MessageType>
- <Params>
- <Param>
  <ParamName>id</ParamName>
  <ParamType>long</ParamType>
</Param>
- <Param>
  <ParamName>pass</ParamName>
  <ParamType>String</ParamType>
</Param>
</Params>
</Message>
- <Message>
  <MessageName>MoveMessage</MessageName>
  <MessageType>0x02</MessageType>
- <Params>
- <Param>
  <ParamName>direction</ParamName>
  <ParamType>byte</ParamType>
</Param>
- <Param>
  <ParamName>accept</ParamName>
  <ParamType>boolean</ParamType>
</Param>
</Params>
</Message>
</Messages>
</MMOG_Message>

```

*Figure 2-1 MMOG message script document*

```

<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:element name="MMOG_Message" type="mmog_message_type"/>

  <xsd:complexType name="mmog_message_type">
    <xsd:element name="Version" type="xs:string"/>
    <xsd:element name="PackageName" type="xs:string"/>
    <xsd:element name="Classpath" type="xs:string"/>
    <xsd:element name="Messages" type="messages_type">
    </xsd:complexType>

  <xsd:complexType name="messages_type">
    <xsd:element name="Message" type="message_type">
    </xsd:complexType>

  <xsd:complexType name="message_type">
    <xsd:element name="MessageName" type="xs:string"/>
    <xsd:element name="MessageType" type="xs:string"/>
    <xsd:element name="Params" type="Params_type">
    </xsd:complexType>

  <xsd:complexType name="Params_type">
    <xsd:element name="Param" type="Param_type">
    </xsd:complexType>

  <xsd:complexType name="Param_type">
    <xsd:element name="ParamName" type="xs:string"/>
    <xsd:element name="ParamType" type="xs:string"/>
    </xsd:complexType>
</xsd:schema>

```

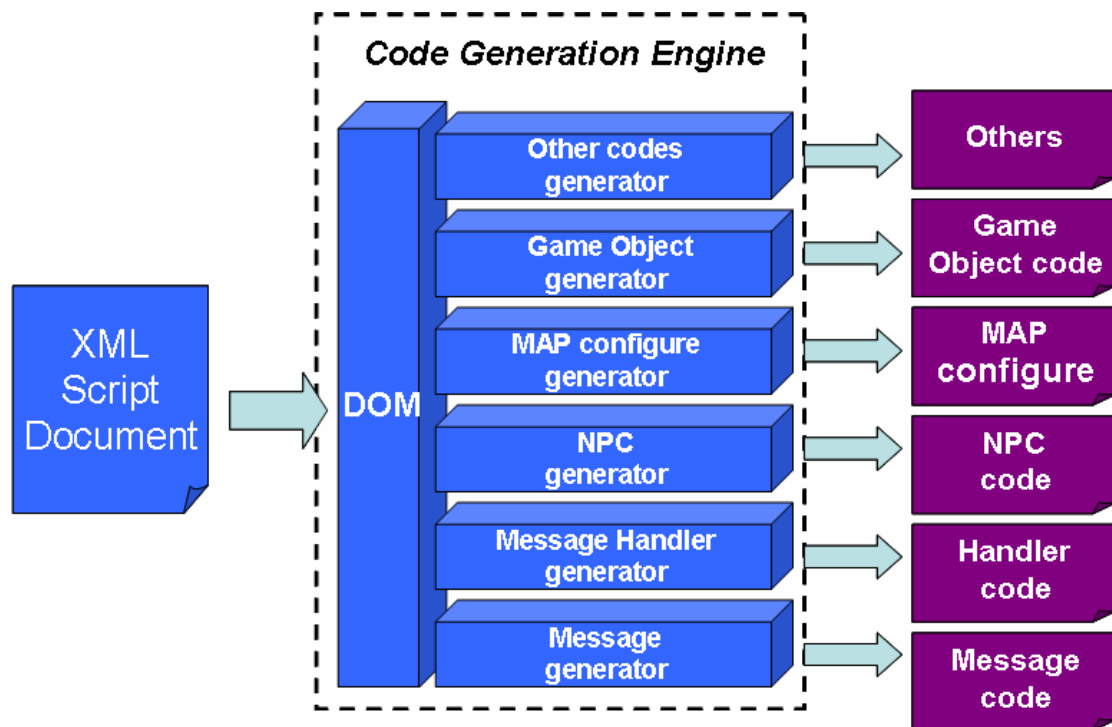
*Figure 2-2 MMOG message xml schema document*



## 2.1.2 CODE GENERATION

After defined the game contents document, the programmers must development the programs according to the definition of contents. The programmers must development all components' programs of the MMOG, the whole work is very complexity and complicated. In order to reduce the programmers' load, we load the document into the code generation engine and make a part of the programs be generated automatically. The programs we generated still need to be edit by programmer, because the code generation engine only generate the code which can be described, and others like method operation logic which could not be described in the description document must be write into the program by them self. The code generation engine use a XML parser to parser the XML description document, and send the data which were got in the parser process to the corresponding generator, then the generator will be

responsible to generate the codes.



*Figure 2-3 MMOG Code generation flow*

*Figure 2-3* show the MMOG Code generation flow. Each generator handle a kind of code, it will store this kind of program's architecture and the part of the same code and will have some method to generate the part of the different code according to received data.

## 2.2 MMOG MANAGEMENT SYSTEM

The MMOG management system usually resides on different machines. That means a MMOG management system is a distributed application. The design of a MMOG management system mainly involves dealing with the distributing, controlling. Before the JMX, there does not existence any standard management ways based on java about system setup, management, and lookup and shut down works. Therefore,

different software or applications will have different management way. The software's management way or function are designed according to the customers' necessary. We design the MMOG management system based on the JMX, and we build up a central management server which can be connect by any JMX support remote connection management ways, and we can add or delete the game servers dynamically.[5][6]

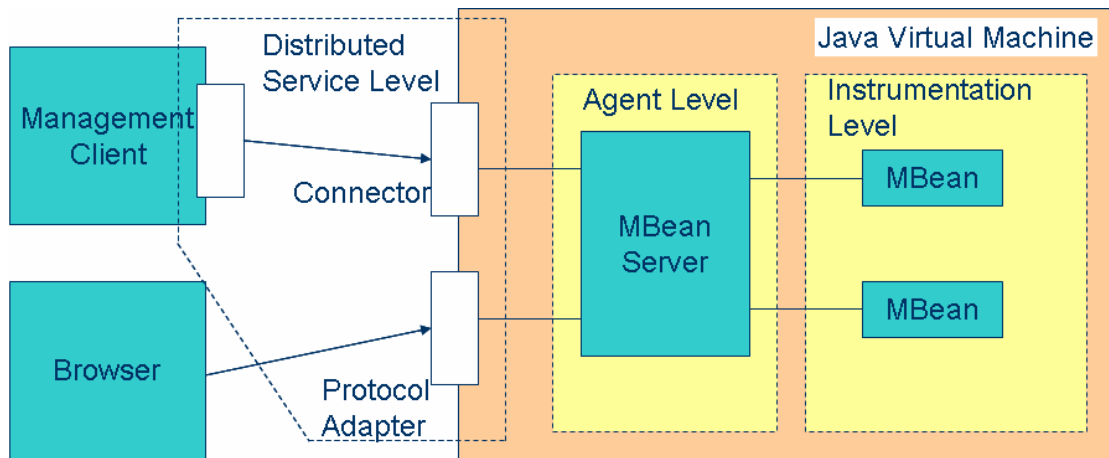
### 2.2.1 JMX OVERVIEW

The Java Management Extensions (JMX) technology is an open technology for management and monitoring that can be deployed wherever management and monitoring are needed. By design, this standard is suitable for adapting legacy systems, implementing new management and monitoring solutions and plugging into those of the future.



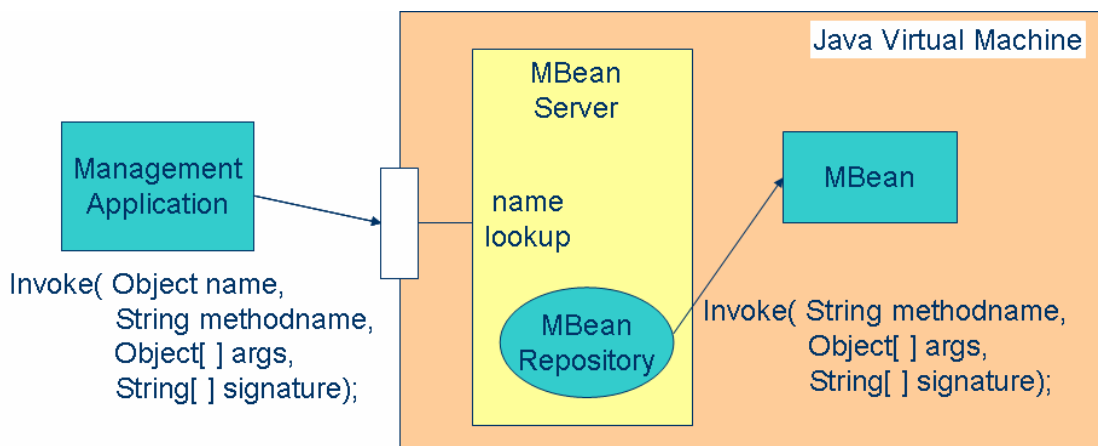
*Figure 2-4* is the JMX management architecture; it is implemented by three levels architecture. First, in order to manage the applications, we will create an MBean which be the most basic management component of JMX at the instrumentation level, it provide the functions to management the application. And then we register the MBean to the MBean server and it will be add to the MBean server's repository at the agent level. The MBean server can management the applications by invoke the MBeans, and it provide a unify management interface which can be invoke by remote managers. At the distributed service level, the MBean server doesn't provide the remote service connection functions itself, but we can achieve it by create a remote connector or connection protocol MBeans at the server. And then the managers which using different JVM can remote setup or management the components on the MBean

server through the connectors.



**Figure 2-4 JMX Management Architecture**

After build the MBean server and register some MBeans on it, then we can start management the applications. The MBean server invoke mechanism is show in the **Figure 2-5**, at first, the management application connect to the MBean server by JMX connector or other connection protocol, and it can lookup all of the MBeans which were registered at the server. Then it can send a invoke request to the server. The request includes some values like MBean name, method name, calling attributes and signatures.

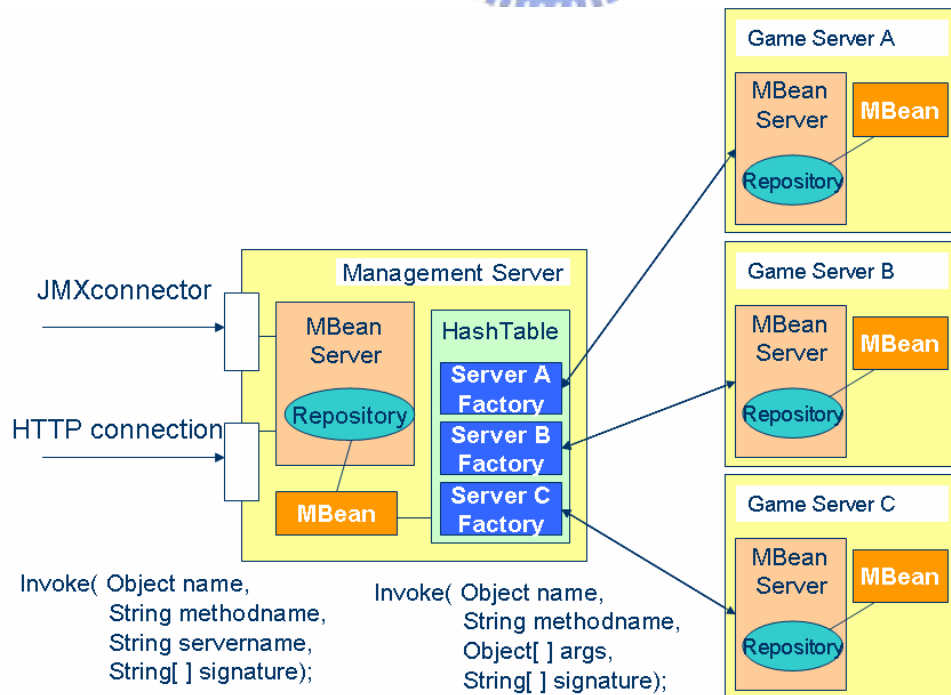


**Figure 2-5 MBean Server Invoke Mechanism**

After server receive the request it will find out the MBean from the MBean repository according to the assigned MBean name, and then send a invoke request to the MBean. The MBean will execute the assignment method and send the return value to the MBean server. And server will return this value to the management application which sent this request. Therefore, programmers can use this mechanism to implement a management application.

### 2.2.2 MMOG MANAGEMENT SYSTEM DESIGN

The MMOG environment is combined by many servers. And each server maintains different components of the MMOG, and they will have some services different from others. Therefore, each server will need their own management application, and we need a central management console which provide a unify management interface. Therefore in order to build up a central management environment, we introduced a MMOG management system architecture show in the *Figure 2-6*.



*Figure 2-6 MMOG Management System Architecture*



It is three-layer JMX architecture; it includes the central management server, game servers and remote management application. The system manager can connect to the central management server by using the browser or others management application, and then invoke the methods of the MBean which register at the central management server. The central management server can invoke the methods of the MBean which register at the game server by using the game server factory object reference. According to each game server, the central management server will have each game server's server factory object reference. The game server factory is a management agent for corresponding game server, and it is responsible for the connection to the game server and invokes the Mbean's method on the game server.

Under this architecture, each game server will build up an MBean server, and design a Mbean according to the resource which want to be management. The MBean define the method like GetValue, SetValue or InvokeMethod, and implemented by using the game server's management API. After build up the MBean server, and register the Mbean in it, the game server will connect to the central management server and make a registry procedure which using the JMX invoke mechanism.

Because the game servers may be add or shut down dynamically during the running time. We must control all of the game servers in the MMOG environment. Therefore, we must design a dynamic server management mechanism, it allows each game server can be join to the management system actively or passive. We will create a registry MBean at the central management server which responsible for game servers registry procedure. After receive the registry request from the game server, management server will create a game server factory and add it to the hash table. The game server factory store the game server's name, IP address and port. And it haves some

functions like connect to game server or invoke a game server's MBean's method. The management server have a hash table which store all server factory's reference and it will have a servers management MBean which responsible for find out the server factory form the hash table according to the manager's assignment and execute the server factory's method to achieve a remote management function invoke mechanism.

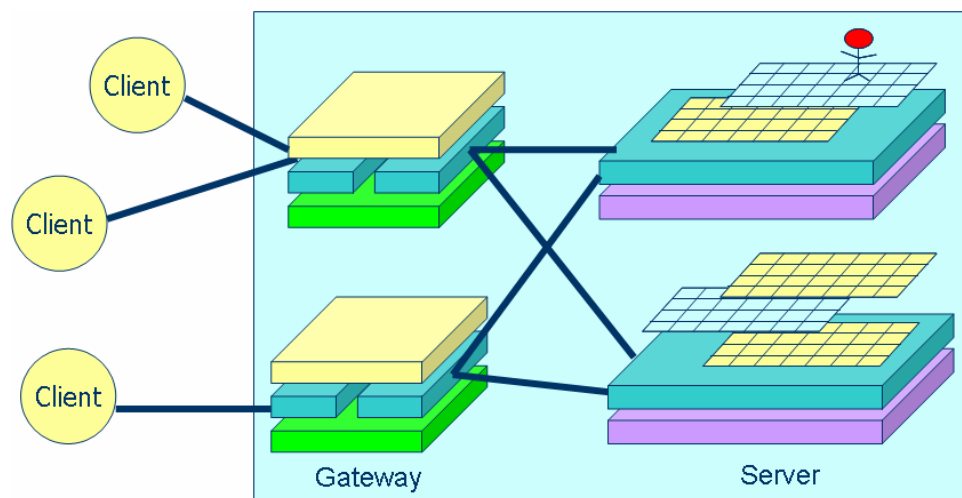
Therefore, the whole management work will become simply and with more elasticity. The manager just only send a request which includes a server name, method name and other attributes to the central management server, then the server will accomplish this request and send the return value to the manager. And when a new game server is add to the MMOG environment, it can be management immediatly by using the registry procedure. And we can invoke a method to refresh the hash table at the management server, because the game servers maybe crash at the runtime so we need to handle this event and remove the server factory from the hash table.

## CHAPTER 3 SYSTEM IMPLEMENTATION

In this chapter, we choose the DOIT MMOG platform to implement the management and development system we presented on it. At first we will introduce the DOIT MMOG platform, and according to the design of the platform we let a part of the server code be generating by our development system, and create an MBean by using the DOIT platform server API. Then we implemented a central management system which provides JMX connector and HTTP connection.

### 3.1 DOIT PLATFORM OVERVIEW

DOIT is an acronym that stands for Distributed Organized Information Terra. It was designed by the National Chiao Tung University Department of Computer and Information Science Distributed Computing System Laboratory. It is a server side platform technology, and it has some futures like Scalability, Availability, Flexibility and Simplicity. It can support ten thousand of people to interaction in the virtual world in the same time. The DOIT platform system is three-tier architecture includes the servers, gateways and clients show in the *Figure 3-1*.



*Figure 3-1 DOIT Platform Architecture*

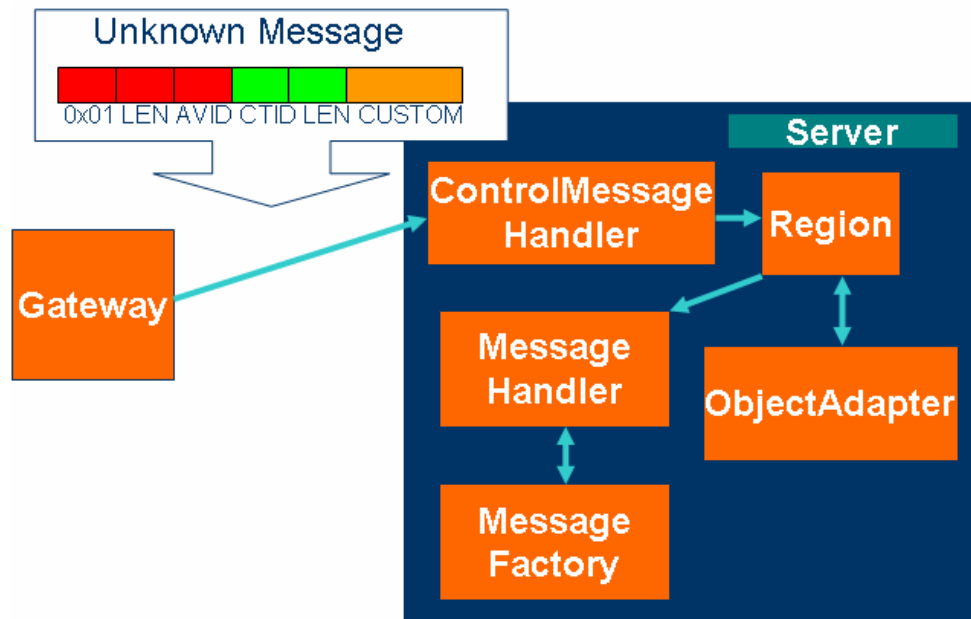
Each client connects to the game server through the gateway, and their action request will be process at different server according to their virtual position. The full virtual world will be divided into several regions which each server responsible for a part of the virtual world, and will be modulated dynamically according to the degree of server load.

### 3.2 DEVELOPMENT SYSTEM IMPLEMENTATION

The DOIT platform provides the high performance game servers and gateways but doesn't include the clients. Because the client side development will be different according to different kind of the MMOG. The platform keeps the elasticity of the client side design, it allows the client side development according to game developer design themselves. The client side design can be implement by JAVA or C++ or other program language and can use the high performance network engine which provided by platform. The only thing which the programmer should be considered in the client side development process is the message protocol definition; it must be implemented follow the server side definition.

At the server side, there are some components must be implemented according to the definition of the platform. There are message, message handler, message factory, NPC configure, and server configure, vwlogic list document and game objects. **Figure 3-2** shows the DOIT platform inner message process flow, it explains the relationship between the message, message handler and message factory. When an unknown message is sand to the server, it will be process at the ControlMessageHandler first. According to the AVID of the message, the message will be delivery to the relative region. AVID is the player's avatar id in the virtual world, and each AVID will be assign to one region according to the player's position in the virtual world. The region

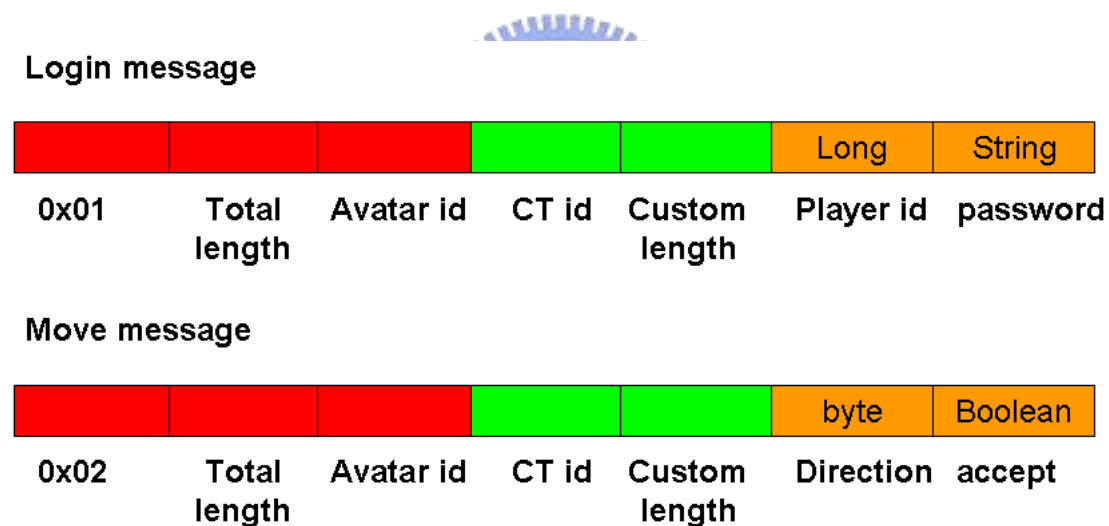
will find out the message handler at the Object Adapter according to the type value of the message, and then send the message to the handler. The handler will cast the unknown message to the original message type by using the message factory. And then execute the message operation, and create an update message return to the client.



*Figure 3-2 DOIT Platform Inner Message Process Flow*

The components which programmers must development are the messages, message handlers, and message factories in the inner message process flow and they are the generality part of the work load in the server side development. Therefore, the development system which we presented will focus on to these components' development. In order to descript these components, the first step is to define the XML schema document according to the format of the message, and find out the relationship between these components. In the DOIT platform, each message is defined as a byte buffer. In the *Figure 3-2*, there is an unknown type message be sent to the server. This message is composed by two components. First is the fixed component, they include a message type, total message length, avatar id, player id and

the length of custom component. Each message will have the same format at the first component and next component is the custom format. According to the different messages' function, the client will need to send some different attributes to the servers for the game logic operation. Therefore, the custom component is composed by some attributes. In the DOIT platform, we support some types of the attributes include int, string, Boolean, byte, short, long and float. For example, a player login message is sent when a new client join to the virtual world and it must include the player id and password attribute. So the custom component in the player login message is composed by a player id string and a password string, the full format is show in the *Figure 3-3*.



*Figure 3-3 Examples of the Message Format*

Because of the total length, avatar id, ct id, custom length attributes' value are assigned dynamically during the message sending process, so we will not necessary to describe them. We must describe the others, include the message type, and custom attributes and we also describe the message version number, package name and the class path. *Figure 2-1* is a MMOG message description document; it describes the

login and move message protocol. And we defined an xml schema document to verify it; the xml schema document is show in the *Figure 2-2*. After defined the MMOG script document, the next step is to design the code generation engine. The code generation engine will generate the code according to the attributes' value which we described in the script document. Therefore, we analyze the code which we want to generate to induce some regular and relationship between these codes.

```

package mmog.message;

import mmog.net;
import java.nio;

public class MoveMessage
    extends Message {
    private byte direction;
    private boolean accept;
    public MoveMessage() {
    }

    public void decode(ByteBuffer bb) {
        direction = bb.get();
        if (bb.getShort().intValue() == 0) {
            accept = "true";
        }
        else {
            accept = "flase";
        }
    }

    public void encode(ByteBuffer bb) {
        bb.put(direction);
        if (accept == "false") {
            bb.putShort(0);
        }
        else {
            bb.putShort(1);
        }
    }

    public int type() {
        return 0x02;
    }

    public byte getDirection() {
        return direction;
    }

    public void setDirection(byte direction) {
        this.direction = direction;
    }

    public boolean isAccept() {
        return accept;
    }

    public void setAccept(boolean accept) {
        this.accept = accept;
    }
}

```

```

package mmog.message;

import mmog.net;
import java.nio;

public class LoginMessage
    extends Message {
    private long id;
    private String pass;
    public LoginMessage() {
    }

    public void decode(ByteBuffer bb) {
        id = bb.getLong();
        Short s = bb.getShort();
        pass = bb.get(s.intValue());
    }

    public void encode(ByteBuffer bb) {
        bb.putLong(id);
        bb.putShort( (Short) pass.getBytes().length);
        bb.put(pass.getBytes());
    }

    public int type() {
        return 0x01;
    }

    public long getId() {
        return id;
    }

    public void setId(long id) {
        this.id = id;
    }

    public String getPass() {
        return pass;
    }

    public void setPass(String pass) {
        this.pass = pass;
    }
}

```

*Figure 3-4 The Code of MoveMessage and LoginMessage*

*Figure 3-4* shows the code of the move and login message. We block all the

components which are different from another in the programs. The first block is the red color block; it includes the message name and attributes' name and type and the message type value which all described in the MMOG script document. Therefore, this block code will be generated according to the MMOG script document directly without others logic determination. Next is the blue color block; because each message program will have encode and decode method, and the method logic operation will be different according to the attributes included by the message.

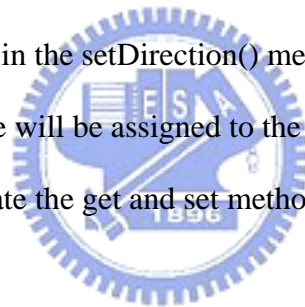
The decode method function is to put the attributes' value from the byte buffer, and the encode method is doing the inverse work. Therefore, we must insert the code into the method according to the type of the attributes. In the decode method, we must use the JAVA byte buffer API to get the values from the byte buffer, and the sequence which we get the attributes must follow the sequence of the attributes which we described in the MMOG script document. And the encode method is also. For example, we described the move message in the *Figure 2-1*, it defined the move message have two attributes; direction and accept, and they will have the same order in the byte buffer.

Therefore, in the decode method, we assign direction attribute value by using the get() method to get a byte value from the byte buffer and then we use some code to get the accept attribute value. Because the JAVA byte buffer API doesn't provide the method to get or put the Boolean and string value, we use a short value to replace the Boolean value, and we use a short value and a byte array to replace the string. In the decode method, we use getShort() method to get a short value from the byte buffer and to differentiate the value. If it is equal to zero then it represent the Boolean value false, if it is equal to one then it represent the Boolean value true. So, we can assign a Boolean



value to the accept attribute according to the short value. If the attribute type is the string, then we use a short value to record the length of the string and use the bytes to store the string. By the same way, in the encode method we also need to use some JAVA byte buffer API to put the value into the byte buffer according to the type and sequence of the attributes.

The last block is the purple color block, if composed by the get and set methods. Each attributes in the message will accompany a get and a set method. We must insert these methods according to the attributes' name and type. For example, in the move message, the attribute direction will have the `getDirection()` and the `setDirection()` methods. In the `getDirection()` method, we will return a byte value which is the value of the direction attribute. And in the `setDirection()` method, we will send a byte value into the method, and this value will be assigned to the direction. Others attribute will follow the same way to generate the get and set methods.



Besides the message code, we will also generate the message factory and message handler code. The code of the login message factory and handler is show in the **Figure 3-5**. They are also follow a fixed format and the only thing that different is the class name. We just only insert the message name into the program where the red block area. Therefore, we can generate these codes fast without do any logic determine. After generated the code, the last work is to generate the vwlogic properties file. The game server will load the message handlers dynamically when system startup according to the description of the vwlogic properties file. The vwlogic properties file will store the entire handler and factories name and class path, and the record format is show in the next page. Each line represents a message handler or factory and it distributes the class path and message type.

1=cis.game.common.message.LoginMessageFactory/0x01

2=cis.game.server.handler.LoginMessageHandler/0x01

1=cis.game.common.message.MoveMessageFactory/0x02

2=cis.game.server.handler.MoveMessageHandler/0x02

```
package mmog.message;

import mmog.net.*;
import mmog.doit.*;
import java.util.*;

public class LoginMessageHandler
    implements GameMessageHandler {
    private static Logger log = Logger.getLogger(LoginMessageHandler.class);
    private GameContext context;
    private Container container;
    public void init(GameContext context) {
        this.context = context;
        container = context.getContainer();
    }

    public void onMessage(MessageInfo msginfo) {
        /*---need something to work---*/
    }
}

package mmog.message;

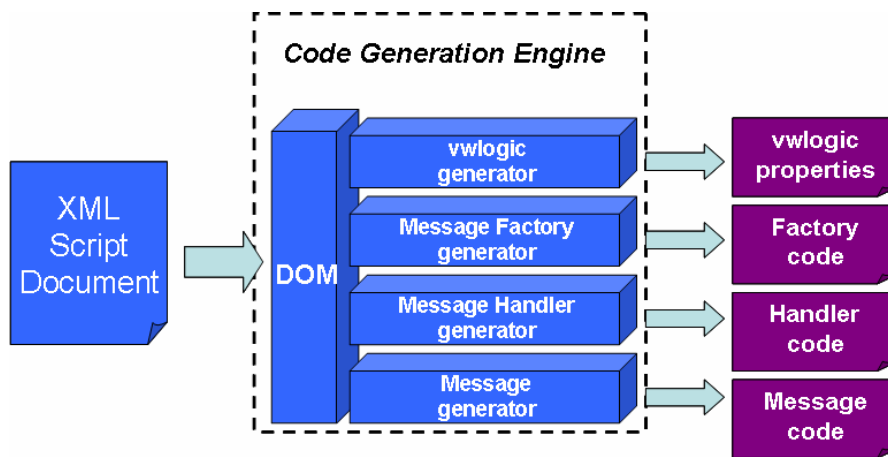
import mmog.net.MessageFactory;
import mmog.net.Message;

public class LoginMessageFactory
    implements MessageFactory {
    public LoginMessageFactory() {
    }

    public Message createMessage() {
        return new LoginMessage();
    }
}
```

*Figure 3-5 The Code of LoginMessageHandler and LoginMessageFactory*

After analyzed the code of the message, factory and handler, then we designed a code generation engine according to the results. The code generation engine must have some functions like load, parser and generate each kind of the code. **Figure 3-6** is the DOIT platform code generation engine architecture. At first, we load the MMOG script document from the assignment path, and then use a XML parser [DOM] to parser the MMOG script document. According to the content of the document, call each generator to generate the code.



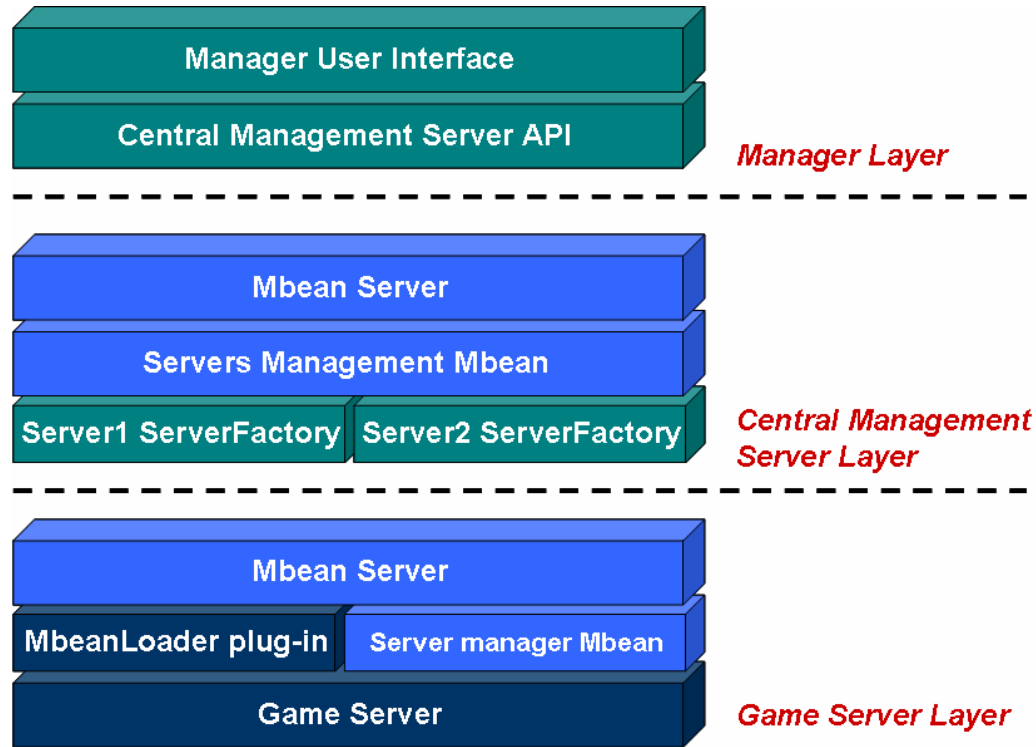
*Figure 3-6 DOIT Platform Code Generation Engine*

These codes are not accomplish, because there are still have some logic code which can't be generated and need to be inserted into the program. Like the message handler program, the onMessage method is responsible for process the message, and it will need to connect to the database or reference to the server state. Therefore, we didn't define the script way in advance; the programmer must insert the logic code into the method according to their necessary.

### 3.3 SERVER MANAGEMENT SYSTEM IMPLEMENTATION

The implementation of the management system will follow the JMX architecture. The system architecture is show in the *Figure 3-7*. It was divided into three layers, game server layer, central management server layer and manager layer. At the game server layer, we packaged the resource which we want to management into the Mbean, and register the local server information to the central management server. At the central management server layer, we use a central management mechanism to management all of the game servers, and provide a remote connection port which allow the remote control. At the manager layer, we defined some remote management API and we implemented a remote management application by using these API. The system

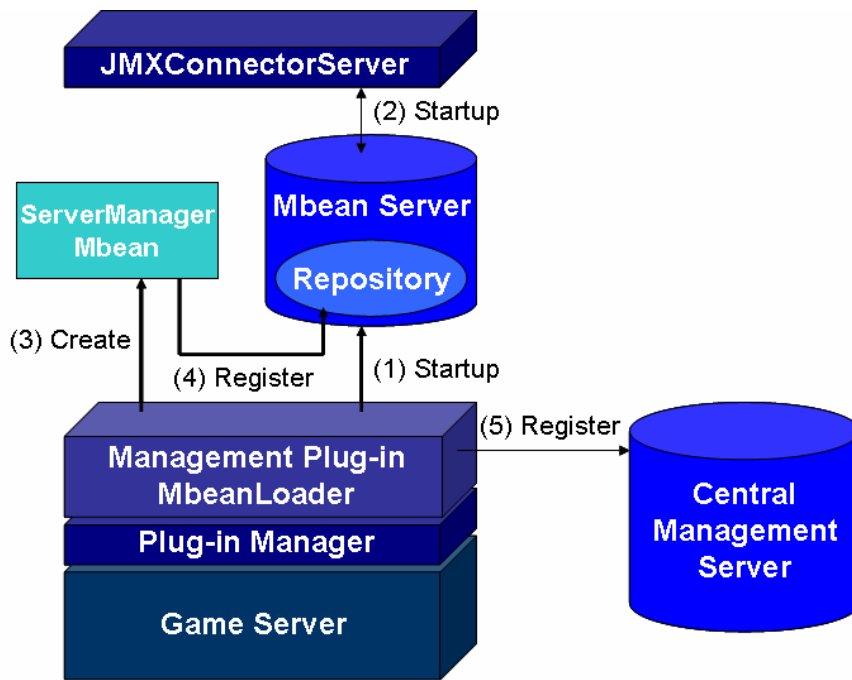
manager can use this application to management the MMOG game servers. The entire management system work flow and design detail will be discussed in the follow subsection.



*Figure 3-7 DOIT Platform Management System Architecture*

### 3.3.1 GAME SERVER LAYER

At the game server layer, the architecture and work flow are show in the *Figure 3-8*. We designed a management plug-in program named MbeanLoader. This program is designed follow the DOIT platform plug-in module and will be executed when the server is startup. It will do some works; it will startup an Mbean server and also creates a Serveranager Mbean which implemented by DOIT platform server management API, then it will register this Mbean to the Mbean server and connect to the central management server and make the register mechanism. Therefore, there are some programs we need to development includes the Serveranager Mbean and MbeanLoader.



*Figure 3-8 Management Plug-in Work Flow*

In the DOIT platform, the server provides some **management API**. We use these API to implement the Serveranager Mbean. Therefore, we defined a ServerManagement MBean interface follow the JMX definition. The interface defined the server state get and set debug and remote register invoke methods. The Serveranager Mbean will implement this interface and will be registered to the Mbean server.

```


public interface ServerManagerMBean {
    public Integer getregioncount();
    public void setregionnumber(Integer regionnumber);
    public String getregionname();
    public void setdebug(Integer debug);
    public void Debug();
    public Integer getAVAccount();
    public Integer getNPCcount();
    public Integer getRegionID();
    public void resetRegion();
    public void register()
}
  
```

There is a part of the ServerManager code in the below. The getregioncount() method is to get the server region count. The method content is to use the MBeanLoader object reference to lookup the RegionManager object reference, and then use the RegionManager's getRegionCount() method to get the count. So, each of the method in the ServerManager will be implemented by the same way.

```
public class ServerManager implements ServerManagerMBean {
    public MBeanLoader mbeanloader;
    public Integer regioncount;

    public ServerManager() {
    }

    public Integer getregioncount() {
        RegionManager a = (RegionManager) mbeanloader.context.
            lookupComponent(
                RegionManager.COMPONENT_NAME);
        return new Integer(a.getRegionCount());
    }
}
```



After implemented the Serveranager Mbean, we must implement the MbeanLoader program. Because it is a plug-in for the DOIT platform, so we must development it according to the platform plug-in definition. The DOIT platform defined a plug-in interface; each plug-in program will implement this interface, and they will get the server component manager object reference. By this object reference, the plug-in program can get the server state or call some component method. Therefore, the entire MbeanLoader code can be divided into four parts. First is the platform defined part, this part is composed by a init(ServerContext context, Hashtable properties) method. This method will be invoked by the game server when this plug-in program was start. The game server will send the ServerContext object reference and a Hashtable

reference to the method, and then the MbeanLoader will get these object reference.

Here is the init() method content:

```
public void init(ServerContext context, Hashtable properties) throws MMOGComponentException {
    this.properties = properties;
    this.context=context;
}
```

The next part is to startup an Mbean server. During this process, we must assign a port number to the Mbean server, and also create a JMXConnectorServer for remote connection. This part of the code is show in the below.

```
LocateRegistry.createRegistry(port);
MBeanServer mbs = MBeanServerFactory.createMBeanServer();
JMXServiceURL url = new
JMXServiceURL("service:jmx:rmi:///jndi/rmi://localhost:"+port+"/server");
JMXConnectorServer cs =JMXConnectorServerFactory.newJMXConnectorServer(url, null, mbs);
cs.start();
```

The third part is to create a ServerManager Mbean, and register it to the Mbean server.

This part of the code is show in the below.

```
sm = new ServerManager();
sm.mbeanloader = this;
mbs.registerMBean( sm,new ObjectName( "MBeans:type=mmog_management.ServerManager" ) );
```

The last part is to connect to the central management server and make a register procedure. We will use a JMXConnector to connect to the central server, and then set the local server name, port and IP to it. At last, we will invoke the addserver method to create a server factory which represents this game server's remote control component. The code is show in the below.

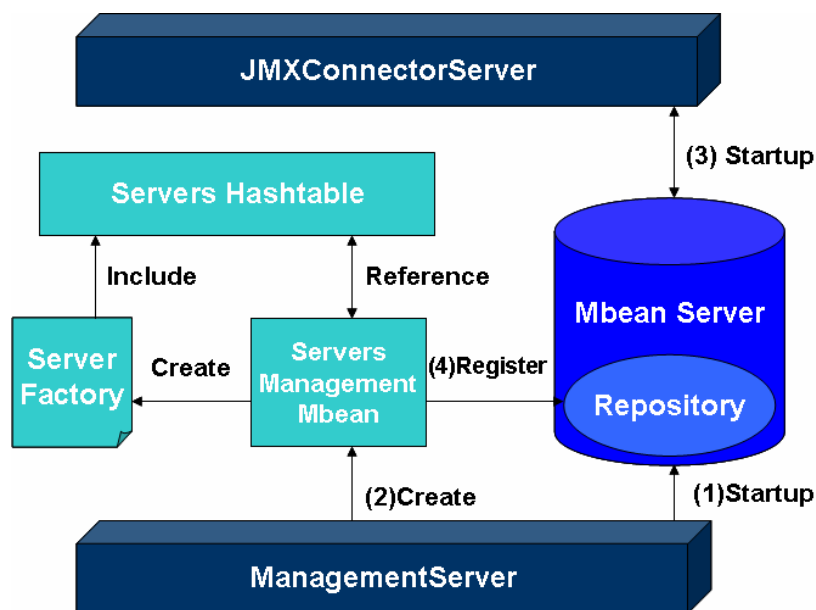
```

JMXServiceURL url = new JMXServiceURL("service:jmx:rmi:///jndi/rmi://" + managementserverip +
":" + managementserverport + "/server");
JMXConnector jmxc = JMXConnectorFactory.connect(url, null);
MBeanServerConnection mbsc = jmxc.getMBeanServerConnection();
ObjectName mbeanName = new
ObjectName("MBeans:type=mmog_management.ServersManagement");
mbsc.setAttribute(mbeanName, new Attribute("servername", servername));
mbsc.setAttribute(mbeanName,new Attribute("serverip", address.getHostAddress()));
mbsc.setAttribute(mbeanName, new Attribute("serverport", port));
mbsc.invoke(mbeanName, "addserver", null, null);

```

### 3.3.2 CENTRAL MANAGEMENT SERVER LAYER

At the central management server layer, the architecture and work flow are show in the *Figure 3-9*. At first, we design a program named ManagementServer which is responsible for the initialize of the central management server; it will startup an Mbean server and creates a ServersManagement Mbean and also registers it to the Mbean server. After the management server have start, it will waiting for the remote connection. It provides two kind of the service, one is for the game server and another is for the manager.



*Figure 3-9 Central Management Server Work Flow*



The game server can connect to the central management server to make a registry procedure which be introduced at the chapter 2.2.2. When the central management server receives this request, it will create a ServerFactory object according to the game server's register information. When the central management server receives the manager's request, it will find out the responsible server factory, and call this server factory to execute the manager's request. Therefore, at the central management server layer, the components which must be development are the ManagementServer, ServerFactory and ServersManagement Mbean.

The ManagementServer's function is similar to the ServerManager which at the game server layer. Therefore, we omit the description of its design detail. We will discuss focus on other part. The ServersManagement Mbean will provide the function for manager to get each server's state, and the registry function for game server. The ServersManagementMbean interface is show in the below. It is dividing into two kinds. First is the get state method like the regioncount() method in the code. This kind of method will be implemented by using the ServerFactory object reference.

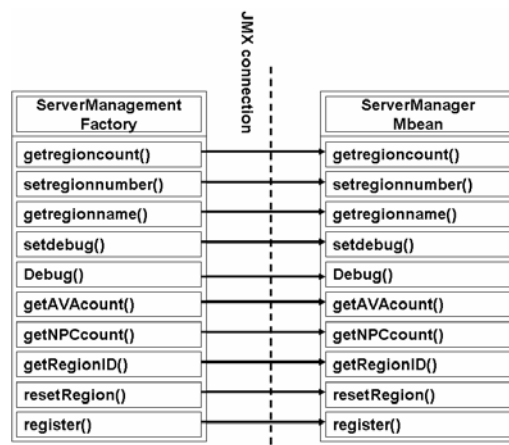
According to different server name, we can get the different ServerFactory object reference, and each ServerFactory object will provide the methods which connect to the game server and to invoke each method in the ServerManagerMBean. And the second kind of the method is the registry method like the addserver() method in the code. The complete registry procedure is that, the game server will connect to the central management server, and then registry his information by invoke the setservername(), setserverip() and setserverport() methods in the ServersManagement Mbean. And then invoke the addserver( ) to create a ServerFactory object, and put this object reference to the Hash table.

```

public interface ServersManagementMBean {
    public void setservername(String servername);
    public void setserverip(String serverip);
    public void setserverport(String serverport);
    public void setdebug(Integer debug);
    public void addserver();
    public ServerManagementFactory getservermanagementfactory();
    public Integer regioncount();
    public String regionname();
    public void Debug();
    public void setregionnumber(Integer regionnumber);
    public Integer getservercount();
    public ManagementData getmdata();
    public Regiondata getregiondata();
}

```

After the implementation of the ServersManagement Mbean, another component we must development is the ServerManagementFactory class. *Figure 3-10* shows the overview of the ServerManagementFactory class. It has the one to one relationship to the ServerMnanger Mbean. According to each method of the ServerMnanger Mbean, the ServerManagementFactory will also have the same name method. Because the ServerManagementFactory plays a intermediary role, it is responsible for the connection to the game server and invoke the Mbean;s method. Therefore, it will have the same methods as the ServerMnanger Mbean.

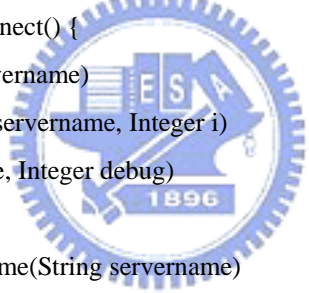


*Figure 3-10 Overview of the ServerManagementFactory Class*

### 3.3.3 MANAGER LAYER

At manager layer, the managers can do management work by the management application. We defined a set of the servers' management API to implement the management application. The servers' management API are show in the below. We design a ServersManagementFactory class; it provides some management methods which can be used by the management application. We can construct a ServersManagementFactory object by a central management server's IP address and the server port number. And then we can use this object's method to connect to the central management server, and also get some game server's state.

```
public class ServersManagementFactory
{
public ServersManagementFactory(String host, String port)
public MBeanServerConnection connect() {
public int getregioncount(String servername)
public String getregionname(String servername, Integer i)
public void debug(String servername, Integer debug)
public String[] getservers()
public Vector getregionsbyServerName(String servername)
public Regiondata getregiondata(String servername, Integer regionnumber)
}
```



## CHAPTER 4 CONCLUSIONS

After we implemented the development and management system under the DOIT MMOG platform, we have got some results in the below.

(1) To use the XML to be the MMOG content description language will have some advantage based on the XML's characteristic. First, the game content designer will have a simply way to learn the description syntax, and they can edit a MMOG description document by any text edit software. Second, because the XML uses the Unicode encoding, so the content designer can use any language code to edit the document. Third, a lot of the data in the MMOG are formed by structured data, so they are suitable be described by XML. After edited the description document, we can use a XML schema file to verify if it is valid, and we can also use the existing XML parser to analyze the document content and then process these data.

(2) By load the MMOG content description document into the code generation engine, we can generate the server side message processing component code. After generating the code, the programmers just only write the game operation logic code into the message handler; then the MMOG server side development will be finished. According to the method, it will reduce the work load of the programmers and also accelerate the development speed. In addition, because we use the XML to describe the message protocol, we can modify the message protocol by change the message attributes' order in the description document, and also generate the code. Therefore, we can get the new version of message code, and we can implement the MMOG encryption mechanism by modify the message protocol continuously.

(3) In the development of the MMOG management system, we use the JMX to solve the network communication problems and use the Mbean server to implement a central management server. It will reduce the work load of the MMOG management system development. And due to all of the resource which we want to management will be packaged into the Mbean in our management system. If there have some new resource also must be management, then we just only add some methods into the Mbean and make an Mbean re-registry action, and then we can keep on the management work and also add these new management resource. And the manager can connect to the management server by http connection or other JMX support connection protocols. According to the method, it will make the management work with more elasticity and more convenient.



## CHAPTER 5 FUTURE WORKS

In our MMOG development system, the development flow is that. At first, we must write a MMOG content description document follow a XML schema which defined by the MMOG platform. And then load this document into the code generation engine; it will generate the code according to the document content. Under this flow, each MMOG platform will define a XML schema, and the game content designer will describe the game content according it. Therefore, a MMOG description document will be accepted by one unique MMOG platform.

In order to let a MMOG to run on different MMOG platform, we hope to define a standard MMOG description language based on XML in the future. It will be defined as a most basic set of the data tags which can be used in any kinds of the MMOG, and it also provides a custom tag mechanism. The user can define tags himself by the customization mechanism for different components between different kinds of MMOG. To use this language, the game content designer can write a MMOG description document follow a standard XML schema and this document can be loaded by many different code generation engine and also generate the code. According to the method, game content designer just only write a MMOG description document, and then we can generate the game code in different MMOG platform. Therefore, we can create the same MMOG on different MMOG platform.

In addition, the DOIT MMOG platform just defined the server side development way, therefore the code generation engine which we designed for it will only responsible for generate the server side code. Because there are exist some components are similarly between the game server and the client, and they can be described in the document. Therefore, we hope that there are a part of the client side code can be

generated by the code generation engine too. So we must design different programming language based code generation engine for different client side implementation way.

At the MMOG management system, we hope to add some system inspection mechanism into the central management server to handle the distributed system management problems. They can be the load balance mechanism, fault detection or others. It will increase the management system's functionality and the reliability.



## BIBLIOGRAPHY

- [1] Zona Inc., <http://www.zona.net>
- [2] BigWorld Technology, <http://www.bigworldgames.com/>
- [3] Extensible Markup Language (XML), <http://www.w3.org/XML/>
- [4] Java Management Extensions (JMX),  
<http://java.sun.com/products/JavaManagement/>
- [5] Wang Shaofeng, Sun Jianguang. "A Framework Design of Workflow Management System with Java RMI", ACM SIGPLAN Notices, Sep.2001, Vol.36, No.9, p86-93
- [6] Wang Shaofeng. "The Role of Java RMI in Designing Workflow Management System", ACM SIGSOFT Software Engineering Notes, Mar. 2001, Vol.26, No.2, pp49-52
- [7] Mulligan, J. & Patrovsky, B. (2003), Developing Online Games: An Insider's Guide, New Riders.
- [8] Lager SDK, <http://www.lager.com.tw/sdk/index.html>

