

國立交通大學

資訊科學與工程研究所

碩士論文

一個可支援手持裝置之巨量多人連線遊戲
的客戶端框架



A Client Framework for Massively Multiplayer Online
Games on Mobile Devices

研究生：彭品勻

指導教授：袁賢銘 教授

中華民國九十六年七月

一個可支援手持裝置之巨量多人連線遊戲的客戶端框架
A Client Framework for Massively Multiplayer Online Games
on Mobile Devices

研究生：彭品勻

Student：Pin-Yun Peng

指導教授：袁賢銘

Advisor：Shyan-Ming Yuan

國立交通大學
資訊科學與工程研究所
碩士論文



Submitted to Institute of Computer Science and Engineering

College of Computer Science

National Chiao Tung University

in partial Fulfillment of the Requirements

for the Degree of

Master

in

Computer Science

July 2007

Hsinchu, Taiwan, Republic of China

中華民國九十六年七月

一個可支援手持裝置之巨量多人連線遊戲 的客戶端框架

研究生：彭品勻

指導教授：袁賢銘

國立交通大學資訊科學與工程研究所

摘要

Massively Multiplayer Online Game (MMOG) 是目前非常受歡迎的一種遊戲類型，其特色在於可支援上千人同時在一個持續性的遊戲世界中進行遊戲。而隨著手持裝置遊戲市場的成長，以及無線技術的進步，手持裝置上的 MMOG 不但可行也開始受到重視。開發一款巨量多人連線遊戲有非常多的議題需要考量，尤其手持裝置遊戲必須在數百種裝置上運作，使得佈署與散佈 MMOG 更加困難。此外，手持裝置的應用程式容易受到周遭環境，像是電信服務或是網路連線問題的干擾。這些都使得開發手持裝置多人連線遊戲倍受挑戰。在本篇論文中，我們將提出一個考量 MMOG 與手持裝置遊戲議題的 MMOG 客戶端框架。嘗試提供一個具有彈性且容易佈署的平台，並使傳統 MMOG 與手持裝置 MMOG 的結合成為可能。

A Client Framework for Massively Multiplayer Online Games on Mobile Devices

Student : Pin-Yin Peng

Advisor : Shyan-Ming Yuan

Department of Computer Science

National Chiao Tung University

Abstract

Massively Multiplayer Online Game (MMOG) is the recently popular game genre that is capable of supporting more than thousands of players in a persistence game world concurrently. With the growing of mobile game market size and improvement of wireless technology, a MMOG on mobile device is possible and being concerned. There are many issues that may be encountered when developing a MMOG. Since the mobile game must be created to run on hundreds of device, to deploy and distribute the MMOG on mobile device is harder. Also, the mobile application is easily interrupted by the surrounding environment, such as telecom service or network connectivity. Those make the mobile online game development more challenging. In this paper, we will propose a client framework for MMOG on mobile device that consider both MMOG and mobile games issues. We will try to provide a flexible and easy-to-deploy platform and also make the combination of traditional MMOG and mobile MMOG possible.

Acknowledgement

首先要誠摯地感謝指導教授袁賢銘博士，從大四專題開始，以自由開明的風氣帶領我們，並給予最大的研究創意空間，使我在這些年中獲益匪淺。其次要感謝所有口試委員，在百忙之中給予我許多寶貴的建議，使得本篇論文能夠更完整而嚴謹。

在交大分散式系統實驗室裡，不管是學術上的討論，或是生活點滴的關心，感謝所有的學長姐、同學、學弟妹，因為你們讓兩年的研究生生活變得多彩多姿。特別感謝蕭存喻、葉秉哲、吳瑞祥、鄭明俊、邱繼弘學長們，在研究過程中給予許多指導與建議，並感謝曾共同參與相關研究的陸振恩、蘇科旭、葉倫武、范志欽學長們。此外，也感謝宋牧奇、脫志曜、熊家媛、蔡宗穎、尤喜夫同學，在這同窗兩年的期間，互相激勵與幫助。

最後，感謝父母從小的養育與教誨，給予我良好的學習環境，讓我的求學生活無後顧之憂，並以此文獻給我摯愛的雙親。

Table of Contents

Acknowledgement.....	III
Table of Contents	IV
List of Figures.....	VII
List of Tables.....	VIII
Chapter 1 Introduction.....	1
1.1 Preface.....	1
1.2 Motivation.....	1
1.3 Research Objectives.....	2
1.4 Outline of the thesis	4
Chapter 2 Background	5
2.1 Communication model of online games	5
2.1.1 Client-Server Architecture	6
2.1.2 Peer-to-Peer Architecture.....	6
2.1.3 Gateway-based Architecture	6
2.2 Technology Platform of Mobile Devices	7
2.2.1 Java	7
2.2.2 BREW	7
2.2.3 Symbian	8

2.2.4 Windows Mobile.....	8
2.3 DOIT middleware platform	9
2.4 Existing Online Game Platforms on Mobile Devices.....	10
2.4.1 BigWorld Technology	10
2.4.2 Ex Machina – Julius solution.....	11
2.4.3 GASP	11
2.5 Summary	12
Chapter 3 System Architecture.....	13
3.1 Three-Tier Architecture Overview	13
3.2 Mobile Online Game Client Architecture	14
3.2.1 Network Layer	15
3.2.2 GameCenter Layer	16
3.2.3 Application Layer.....	17
3.3 Game Protocol Development.....	17
Chapter 4 Implementation Details	19
4.1 Network Component.....	19
4.2 Game Application Layer	21
4.2.1 MIDlet.....	21
4.2.2 Game application interface	22

4.3 GameCenter Layer	23
4.4 Message Class and Config File Generator.....	24
4.5 Development Flow	27
Chapter 5 Application Demonstration.....	29
5.1 Mobile RPG engine.....	29
5.2 Game rule and protocol.....	30
5.3 Combine with PC version	31
5.4 Offline Game Playing	31
5.5 Implementation and result screenshot.....	32
Chapter 6 Conclusions and Future Works	36
6.1 Conclusions.....	36
6.2 Future Works.....	37
Chapter 7 Reference	39
Appendix.....	42
A.1 XML Schema Definition of Message Protocol.....	42
A.2 Development Example.....	43
A.2.1. Game Application	43
A.2.2. Game Center	44
A.2.3. Implement the message handler.....	45

List of Figures

Figure 2-1 Communication model of online service	5
Figure 2-2 DOIT platform architecture and component	9
Figure 3-1 Client-Gateway-Server Architecture	13
Figure 3-2 Mobile Online Game Client Architecture	15
Figure 3-3 Detail layout and processing of GameCenter Layer	16
Figure 4-1 Message format and process	19
Figure 4-2 Class diagram of network component.....	20
Figure 4-3 Life cycle of a MIDlet.....	21
Figure 4-4 Game application interface and Gamelet class	22
Figure 4-5 Class diagram of the GameCenter layer.....	23
Figure 4-6 Development flow	27
Figure 5-1 Screenshot of the mobieRPG demo game.....	29
Figure 5-2 Login form and game world.....	33
Figure 5-3 Player in the paused state	33
Figure 5-4 Demo game on both mobile and PC clients.....	34

List of Tables

Table 4-1 The definition of message and related components.....	24
Table 5-1 Game protocol of the demo game.....	30
Table 5-2 Message types and contents.....	32



Chapter 1 Introduction

1.1 Preface

Massively Multiplayer Online Game (MMOG) is the recently popular game genre that is capable of supporting more than thousands of players in a persistence game world concurrently. With the growing of mobile game market size and improvement of wireless technology, a MMOG on mobile device is possible and being concerned. There are many issues that may be encountered when developing a MMOG. Since the mobile game must be created to run on hundreds of device, to deploy and distribute MMOG on mobile device makes the game development more challenged. Therefore, a middleware solution for MMOG on mobile device gains more and more respect.



1.2 Motivation

With the capability growth of mobile device and the innovation of the wireless communication technique, wireless entertainment is respected recently. By the investigation in IGDA (International Game Developers Association) [1] , *2005 Mobile Game White Paper* [2] , the recent market trend shows that: the availability and quality of games are enhanced, network capability is improved and more traditional game publishers jump in. The game-play habits of consumers also change and reveal that they play for more than just killing down time.

Furthermore, the evolution of wireless technique leads the high speed, high quality and low latency network connectivity on mobile handsets. The connectivity

will support new distribution models and realize new game play types. It provides not only the online game service but enable the true real-time games between handsets and even as part of existing large network with PC and consoles.

All of the above inspires the design of mobile online games and game development platforms. Even though there are many existing MMOG and mobile game middleware solutions, most of them are closed and commercial products and focus only on MMO (Massively Multiple Online) or Mobile topic. As a MMO platform, it confronts more challenges in server-side management and system architecture, such as scalability, flexibility, load-balancing and performance. When the game world grows up, the system should support the increasing players and contents and then provide a load-balancing and high-performance service. Furthermore, the whole system should offer a flexible game protocol and content design to satisfy many different game types.

On the other hand, to design a mobile game platform, there are usually more client-side handset issues to be considered. The mobile client application is easy to be interrupted by surrounding environment. For example, when user is out of service range or receiving a coming call, the client application may suspend and even lose application resource. Therefore, we propose a client framework for MMOG on mobile device that consider both MMOG and mobile games issues. We hope to provide a flexible and easy-to-deploy platform and also make the combination of traditional MMOG and mobile MMOG possible.

1.3 Research Objectives

The objectives of our framework can be summarized as the following four parts:

- **Easy to Adapt and Extend**

The game contents and types of a MMOG can be various. An inflexible MMOG middleware will restrain the originality of game design. The communication protocol of middleware online service must be straightforward and easy to adapt and extend. The Game objects that present in the game world should also be as flexible as possible. We can provide some typical game object models but should not assume the forms of them. An ideal game middleware must provide the maximum resilience for game design.

- **Easy to Develop and Deploy**

Time to market is very important for the game provider. The development of game contents need to be easy and fast. The platform should provide simple and flexible API for game developer. Moreover, most MMOG vendors update the contents frequently to provide attractive game. The game deployment mechanism should be simple to reduce the cost of content extension.

- **Dealing with the Mobile-specific issue**

The situation of a mobile game client is more complex than typical PC or console games. Although the quality and speed of wireless communication has been improved, the surrounding environment and many telecom issues will still make the network connectivity being poor. The application state on mobile handset is also hard to control. The limited hardware resource and high-priority telecom service will let the client application be interrupted easily. Thus, the framework of MMO mobile game client should take above problems into account and provide easy-to-use interface to handle those status.

- **Combination of mobile and traditional MMOG**

The spirit and game-play on PC MMOG and mobile MMOG can be the same. Although the PC or console provides more resource and capability to present such kind of game, the innovation of mobile device makes the combination possible. Even if the presentation may be different, by using the same communication protocol, the mobile device should be able to function the same game logic and play in the same virtual game world.

1.4 Outline of the thesis

In Chapter 2, we discuss the background of MMOG and mobile application development and introduce some existing platforms. In Chapter 3, we propose the architecture of our system, and discuss the purpose and responsibility of each component. For deep understanding, we describe the implementation details and the design method in Chapter 4. To demonstrate the usefulness of the platform we propose, the contents of Chapter 5 presents a hybrid MMO game built on our system. Finally, in Chapter 6, we give the conclusion and future works for our system.

Chapter 2 Background

2.1 Communication model of online games

To design an online service, communication model is an important part. Here is the discussion of three common types, namely, Client-Server, Peer-to-Peer and Gateway-based architecture. For MMOG platform, we may focus on the scalability to support massively clients. Moreover, since the client platform may be varied, the capability and related issue of client must be considered.

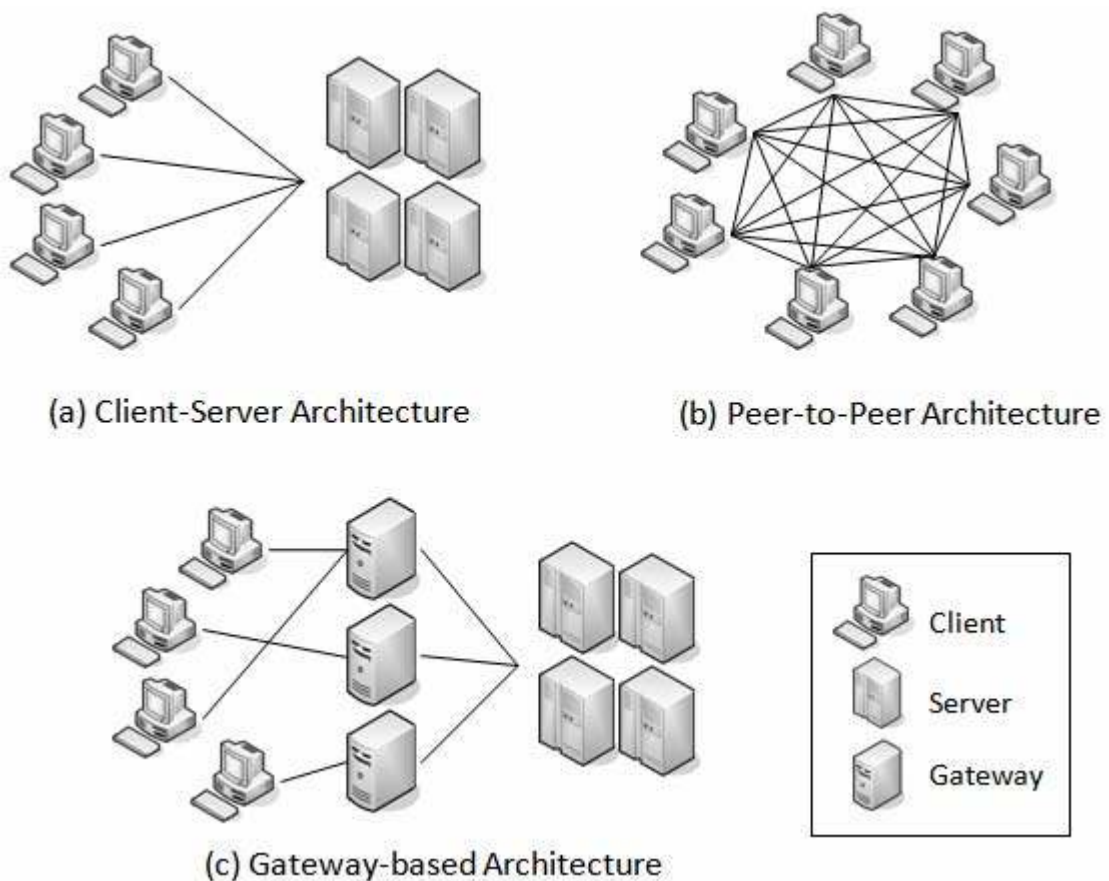


Figure 2-1 Communication model of online service

2.1.1 Client-Server Architecture

Client-Server model is the traditional architecture for online service and can also be applied to MMOG platform. Large-scale MMO Games, such as Ultima Online [3] , Lineage [4] or World of Warcraft [5] , often use server cluster (see Figure 2-1(a)) to handle massive players and provide a centralized and reliable service. But when the virtual game world extends or players increase, it may suffer from the scalability issue and bandwidth limitation at the server entrance.

2.1.2 Peer-to-Peer Architecture

P2P technique has grown and evolved maturely in recent year. Although P2P model (shown in Figure 2-1(b)) is suitable for multi-user and content share service, there are some problems for MMOG. The major one is the consistency problem. Communication cost for maintaining consistency makes this architecture hard to extend when user and virtual world grow up. The cheating and management problems also cause the P2P model uncommon in commercial product. Moreover, for mobile peer, it is more difficult to build up a large scale multiplayer online game.

2.1.3 Gateway-based Architecture

In the Gateway-based architecture, as Figure 2-1 (c) shows, clients treat gateways as servers and the gateways dispatch data from clients to the corresponding backend servers. The benefit of this model is that the gateways share the bandwidth at server entrance and may reduce the latency from client to server by deploy those gateways at main Internet Service Providers (ISPs). Although the gateway will bring

overhand to routing message and make the load-balancing more difficult, it can serve as a security defender of server and is easy to scale when service loading rises.

2.2 Technology Platform of Mobile Devices

Technology platforms are the hardware system and software component that form the mobile device application environment. Differing from traditional console or PC games, a mobile game may need to be implemented in various languages to run on hundreds of mobile handset. Within this section, we introduce four major platforms for mobile game client.

2.2.1 Java

Java is a programming language created and maintained by Sun, Inc. The runtime environment on mobile devices is served by Java™ Platform, Micro Edition [6] (Java ME). At present, Java is the most widespread platform technology apparently in terms of numbers of device shipped. To provide a complete application environment on mobile phones, a common adopted case is to combine the Connected Limited Device Configuration [7] (CLDC) with Mobile Information Device Profile [8] (MIDP).

2.2.2 BREW

Binary Runtime Environment for Wireless [9] , BREW for short, is an application development platform created by Qualcomm for mobile phones. It addresses specific market needs while allowing customers to create their own solution under BREW technology. Especially for mobile gaming, it offers solution that

includes hosted management service, network carrier billing system and many game related features.

2.2.3 Symbian

The Symbian OS [10] is an operating system for mobile devices and is well known in the form of the Series 60 handsets. There are multiple platforms based upon Symbian OS that provide the SDK for application development. Applications developed on it are in general portable between different Symbian handset. For game development, Symbian OS is great and provides rich graphical feature to perform.

2.2.4 Windows Mobile

Windows Mobile [11] is a compact operating system for mobile devices, such as Pocket PC, Smart Phone and Portable Media Center, with a set of applications based on the Microsoft Win32 API. To design the application built on Windows Mobile is a little similar to the development of desktop Windows application. For the latest Windows Mobile 6 version, it offers many features with security, scalability and manageability that make the mobile workforce productive.



2.3 DOIT middleware platform

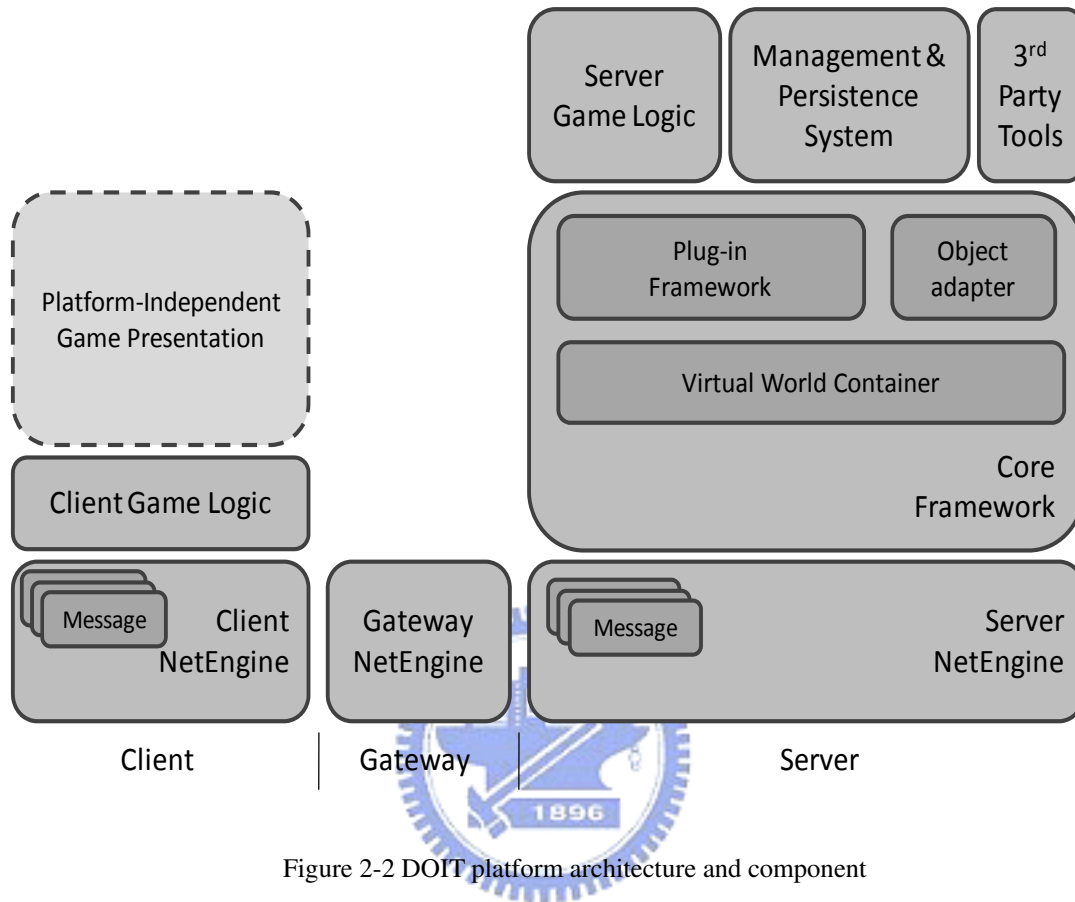


Figure 2-2 DOIT platform architecture and component

DOIT (Distributed-Organized Information Terra) [12] [13] is a MMOG middleware platform which has evolved for several years in our lab (DCSLab of CS NCTU). It presents a client-gateway-server architecture and has following features: scalability, flexibility, high-performance and ease-to-use API. For game development customization and enhancement, it provides the user defined message-based protocol and plug-in framework [14]. The persistence of virtual game world is processed by an ORM (Object Relational Mapping) database system [15]. Furthermore, it offers a management tool to monitor the game world [16].

Up to present, it has been implemented in Java SE solution and got good performance under thousands of player concurrently. But it puts more emphasis on the

whole MMOG middleware platform design and does not provide the detail design of client framework or any mobile device supporting now.

2.4 Existing Online Game Platforms on Mobile

Devices

Both of MMOG and mobile game market has continued to grow over the past several years. Although there are some existing MMOG middlewares and mobile game platforms, most of them are commercial products and developed respectively. For commercial reasons, it's hard to figure out the detail design by the public documents. In this section, we try to explore relevant features as many as possible.

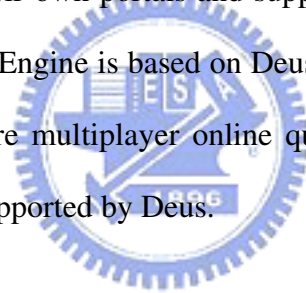
2.4.1 BigWorld Technology



The BigWorld Technology Suite [17] is a complete MMOG solution. It provides a scalable, reliable, customizable and high performance server infrastructure that can handle million of players. For managing the development and live support of MMOG, it offers a management tool to handle cluster setting, view log and check the server status. The client engine integrates the 3D engine, APIs and a set of tools to build up and manage a MMOG world. Even the tools for content creation, map editing, and particle effect generation, the BigWorld Technology suite is covered as well. In addition, the BigWorld Mobile is a layer for mobile device to integrate with the full BigWorld online world. You can establish an individual mobile MMOG or build a mobile version game to play a minor part of existing online game.

2.4.2 Ex Machina – Julius solution

Ex Machina [18] provides a framework to create, deploy and manage multiplayer online games on mobile, web, broad- and narrowcast. Julius solution combines three platforms, Deus, IP United and Quiz Engine, to offer an end-to-end solution for commercial online games. It highlights several features to build up a multiplayer online game with back-end management system for payment, business model and in-game advertising. Each of the three platforms has its own duty in Julius. Deus is a platform and SDK which support massively multiplayer and location based gaming. IP United is a community gaming service that allows game providers to offer numbers of online games as part of their own portals and support the cross-game user service and player competition. Quiz Engine is based on Deus, integrates with IP United, and can create, edit, and configure multiplayer online quizzes, tests, puzzles and much more games for any device supported by Deus.



2.4.3 GASP

GASP (GAMing Service Platform) [19] [20] is an open source platform, under L-GPL license, for mobile multiplayer online game. GASP is implemented in pure Java and conforms the Open Mobile Alliance (OMA) Game Service specification. It uses the object-oriented communication to communicate between client and server over cellular phone network using HTTP. GASP server runs as a Apache Tomcat servlet and uses MySQL database for information storage. Client side interface is available for J2ME MIDP and Doja profiles. The mobile client sends messages over HTTP and the message parameters are formed and formatted by the MooDS (Mobile optimized objects Description and Serialization) protocol. Game developer can

specify the structure of message using dedicated syntax and generate the message encoding/decoding classes.

2.5 Summary

By the discussion in sections 2.1, the client-server or gateway-based communication model is more suitable for MMOG. Especially for commercial MMPG products, the P2P model may bring out more cheating and business problems. For technology platforms of mobile devices, although the future of Symbian is worthy of expectation, the Java technique seems to be essential for widely game deployment at present. Furthermore, most existing middlewares do not offer to develop a MMOG that support to run on both PC and mobile device. The platforms that aim at the mobile device are usually poor of server scalability.



Chapter 3 System Architecture

In this chapter, we present the system overview of our framework that includes the three-tier architecture of the whole online service and the mobile client framework with several layers. Additionally, in the last subsection, we describe the development of game protocol in our system.

3.1 Three-Tier Architecture Overview

Base on the analysis of the communication model in section 2.1, we believe that the gateway-based model is better for multiplayer online game platform, especially for supporting the mobile device. To realize the objective of integrating with existing MMOG platform, we choose the DOIT platform as the fundamental three-tier framework. Since it is not a commercial product and provides abstract interfaces and flexible message protocol, the integration of different client platforms is feasible.

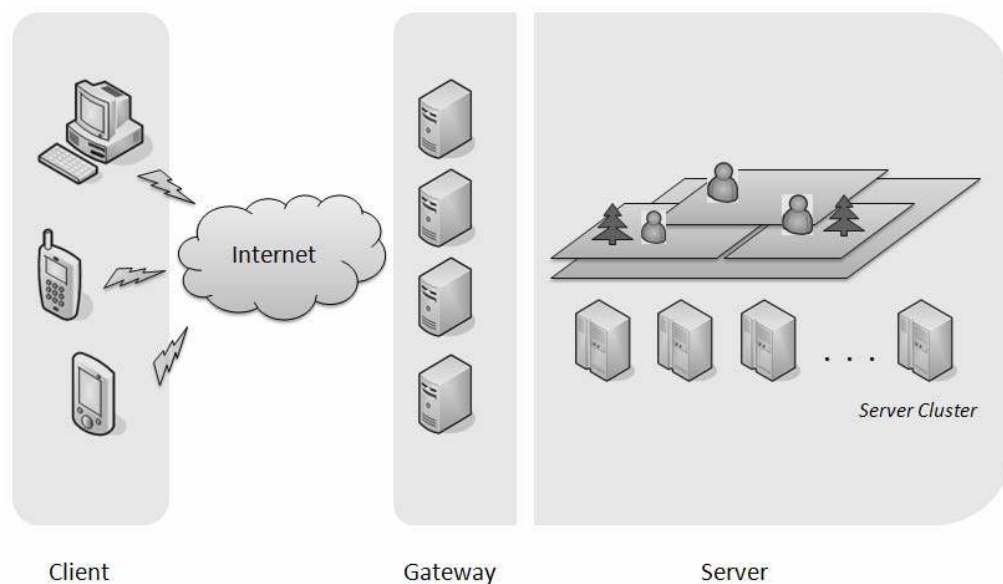
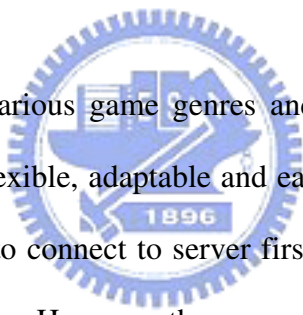


Figure 3-1 Client-Gateway-Server Architecture

Figure 3-1 shows the three-tier architecture. In server-side, the server clusters form the virtual world and handle the game logic and game world persistence. The virtual game world is divided into multiple regions and each server may contain zero to many regions. The game objects within one region share the same game context and can migrate between regions. The gateway in the middle layer is responsible for routing message from client to server. For mobile online games, the gateway can be designed to handle special issues related to mobile clients, such as user authentication, session pending and so on. Moreover, client-side users may use different platforms but the same protocol to communicate to the server.

3.2 Mobile Online Game Client Architecture



In order to satisfy the various game genres and requirement, an online game client framework should be flexible, adaptable and easy to customize. In general, the MMO PC game clients need to connect to server first and keep the connection alive while a user is playing the game. However, the connectivity of mobile phone clients is not reliable and some mobile users may wish to spare the network resource indeed. For example, a white paper [21] of Java BluePrints points out some aspects of motivation for designing disconnected operation in wireless client and also describes guidelines to design it. In chief, it mentions two reasons for offline operation, that is, the poor network quality and typical use cases for mobile wireless devices.

The mobile client application may interrupt by telecom service or surrounding environment, such as incoming calls, short message or low-battery. The status changes may cause the resource of the application be released. Therefore, the management of network connectivity and application status must also be taken into consideration.

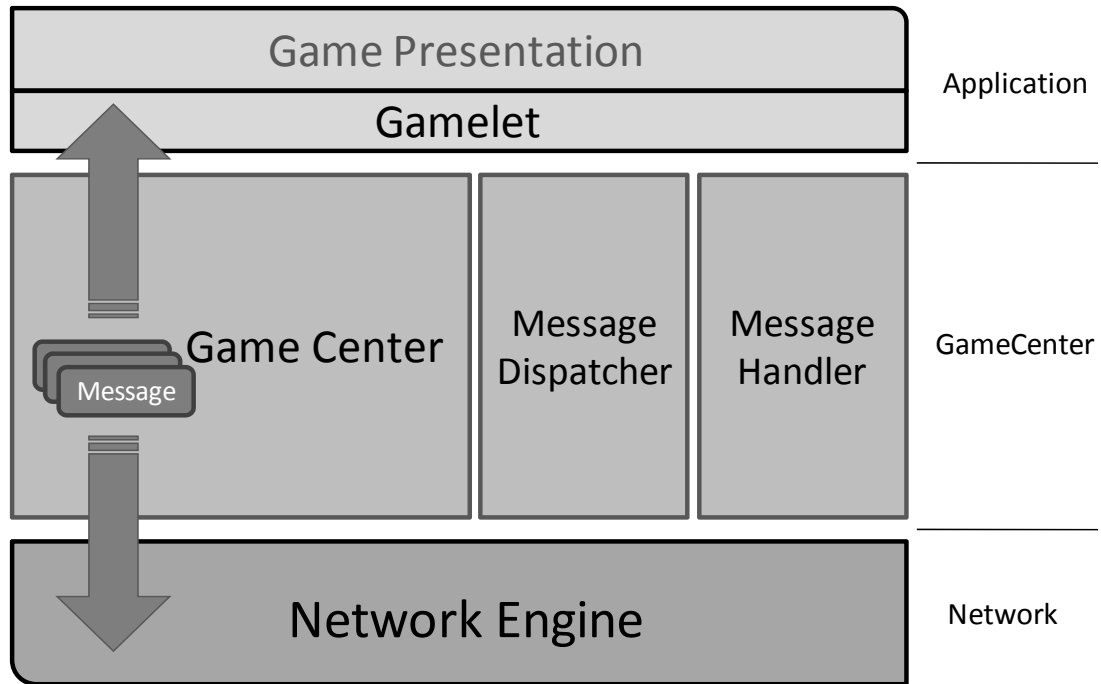


Figure 3-2 Mobile Online Game Client Architecture

To deal with the problems mentioned above, the architecture we design comprises three layers (Shown in Figure 3-2). Each layer contains several components and has particular purpose. The data received from server pass through each layer and is processed as sever update or client-side game logic. In the following subsections, we discuss the functionality and key point of each layer.

3.2.1 Network Layer

The network layer processes all network connection tasks and communicates with server. The communication protocol we use is message-oriented that packing any data transfer between client and server as formatted messages. The network layer abstracts the implementation as interface and can be implemented with different network protocols. For example, we can provide HTTP or TCP/UDP implementation to accommodate different network supports of mobile devices.

3.2.2 GameCenter Layer

The GameCenter in the middle layer collects information from application and network layers to analyze the status of game and process game logic. As Figure 3-2 shows, there are three main components: Game Center, Message Dispatcher and Message Handler. The Game Center receives messages from Application Layer or Network Layer and passes them to the proper Message Dispatcher according to current game status. While the dispatcher gets the messages, it dispatches them to corresponding Message Handler. The handler performs the game logic and then the game presentation updates at the application layer.

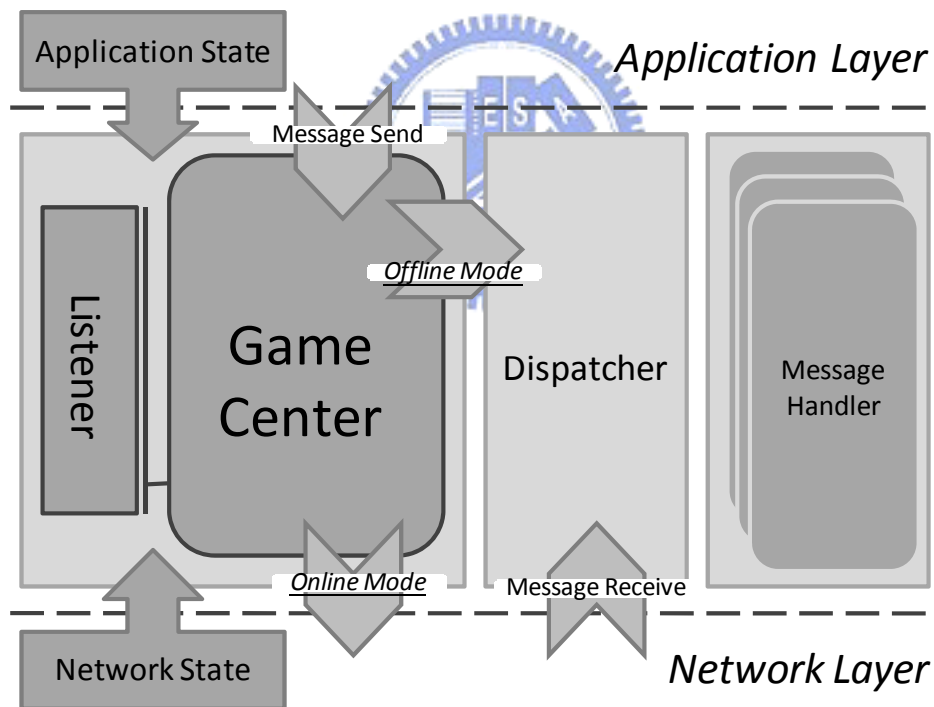


Figure 3-3 Detail layout and processing of GameCenter Layer

Due to the intermittent network connectivity, the mobile online game may need to operate in a disconnected mode. Figure 3-3 shows the detailed message processing of the GameCenter layer and the delivery path of messages. The game center and

listeners monitor the state change of application and network layers. Messages come from gateway (server) are received and packed by network layer and then dispatched. On the other hand, the messages create by the game application itself are sent to the game center layer and forward to the network layer or dispatch to the handler if in offline mode. Depending on the current situation, the game client can serve in different mode and support offline play.

3.2.3 Application Layer

The application layer performs the game presentation to player and exchanges information with lower layers. Since we choose the Java ME solution to develop the system, the base interface is named “Gamelet” as an extension of MIDlet. The Gamelet abstracts the implementation of application state change and offers an event-based interface. The application status can be monitored by defining the application events and register the listeners. It also provides callback functions to be notified when network status alters.

3.3 Game Protocol Development

As we use the message-oriented mechanism to communicate between clients and servers, the development of the game protocol can be simplified as the following steps:

1. Defining the game logic
2. Transforming the game logic to message protocol
3. Implementing the message handler

The game logic in Step 1 depends on the game genre or game contents. For typical role-play games, it may include movement, chat or any interaction between players and NPCs (Non-Player Character). To deploy those game actions, we need to transform them into message protocols. The transformation between logic and protocol is to parameterize the messages. Once the server-side and client-side receive the messages, they process those messages to perform the game logic and update.

The major workload of the development is to translate message protocol into the underlying programming language. To simplify the process, we also provide a method to generate the message class codes and deployment configuration files by defining the message content and related handler in dedicated syntax. The detail procedure will be described in the next chapter.



Chapter 4 Implementation Details

We describe the detail implementation of our framework in this chapter, especially on the three-layer client side framework. The syntax of message protocol definition will also be explained in the final section.

4.1 Network Component

The development of network layer can be separate into three parts, that is, NetEngine, Channel, and Communication Protocol. NetEngine component is a message-oriented lightweight service. It supports network establishment method: connect (for client) or listen (for server). When the connection is established, we can get the connection handle, Channel, and monitor the state by ChannelListener. The Channel is an abstraction of connection and executes the data sending/receiving. The communication protocol we use is a message-based protocol. All data transferred between client and server are encapsulated as a message and the protocol is defined as a list of message types.

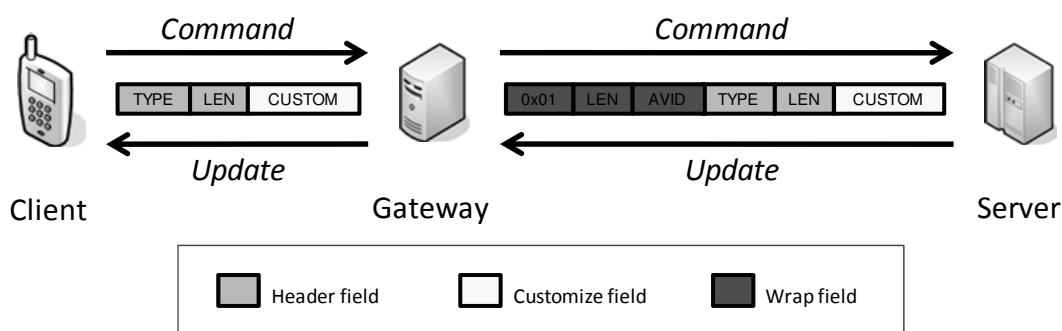


Figure 4-1 Message format and process

Figure 4-1 shows the message format and the delivery between client, gateway and

server. Each message contains the header to specify the message type and length and the payload of user define data. When a message is received by the gateway, it tags it with user id (AVID – Avatar ID) to let the game server knows whom the message belonged to. Then it routes the message to the correct server that holds the player data now. When the server sends the update message, the gateway unwraps the message as well.

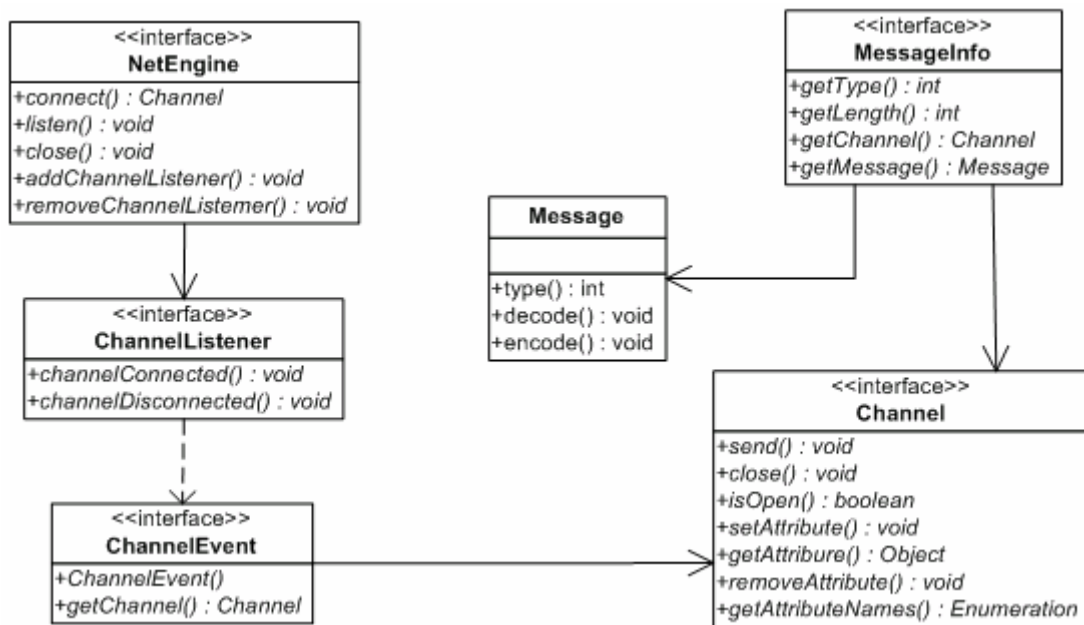


Figure 4-2 Class diagram of network component

The definition of the network components and their relationship are shown in Figure 4-2. When the connection is built by the NetEngine, it begins to send and receive data. We use the Observer design pattern to trace the connection status or any channel event. One can register a listener that implements the interface ChannelListener to monitor the Channel. While the data receive from server, it is identified by the message type and a message template is created. By calling the decode() function, we retrieve the message contents. Then the MessageInfo with message, type, length, channel and some other information is delivered to the

GameCenter Layer and dispatched to corresponding handler. To send the message is similar. The message created by the game application is finally encoded by the `encode()` function and sent to the gateway.

4.2 Game Application Layer

4.2.1 MIDlet

A MIDlet is a MIDP application in Java ME solution. All applications for MIDP are able to retrieve properties from the application descriptor and notify and request state changes.

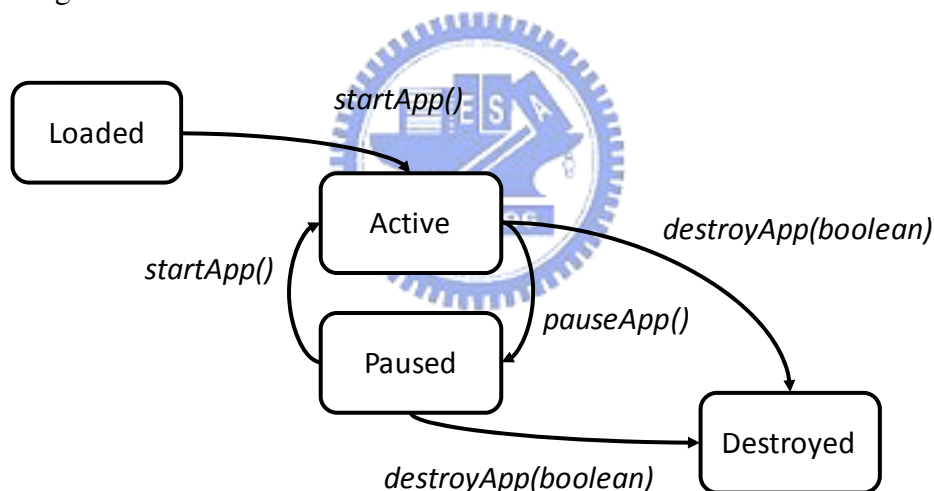


Figure 4-3 Life cycle of a MIDlet

Figure 4-3 shows the life cycle of a MIDlet and there are four states: loaded, active, paused, and destroyed. When the MIDlet state changes, the mobile device application manager will call the related function, `startApp()`, `pauseApp()`, `destroyApp()`, before it enters next state. For example, when the mobile phone user has a coming call, the application manager will call the `pauseApp()` and force the MIDlet to enter the paused state. When the call end, the manager call the `startApp()` and the MIDlet back to

active state. Depending on the practical implementation, those state transformations may cause the application resource to be released temporarily. Thus, the game application itself and the Game Center in the middle layer should be notified when those state change.

4.2.2 Game application interface

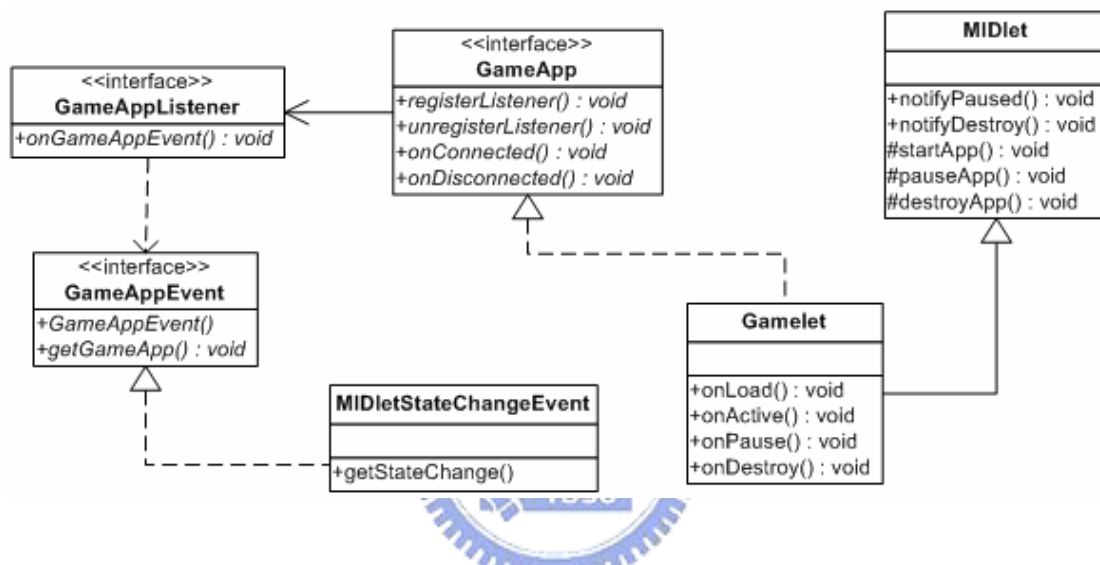


Figure 4-4 Game application interface and Gamelet class

To generalize a game application and the game events of it, we define three interfaces, GameApp, GameAppListener and GameAppEvent (shown in Figure 4-4). The GameApp provides abstract functions to be notified and respond when network connection status changes. We can specify the application event by implementing the GameAppEvent. To monitor the application events, one can implement the GameAppListener and registers it.

Therefore, to manage the MIDlet state, we create an abstract Gamelet class that extends the MIDP MIDlet class and implements the GameApp interface. The Gamelet wraps the state change procedure calls and provide other abstract functions. When the MIDlet state is forced to changed, the Gamelet generates the state change events and

executes the notification to report the status modification.

4.3 GameCenter Layer

The main duty of the GameCenter layer is to collect status information and process the game logic in correct mode. As we discuss in Chapter 3, there are three main components in the GameCenter layer. The implementation of them is described afterward.

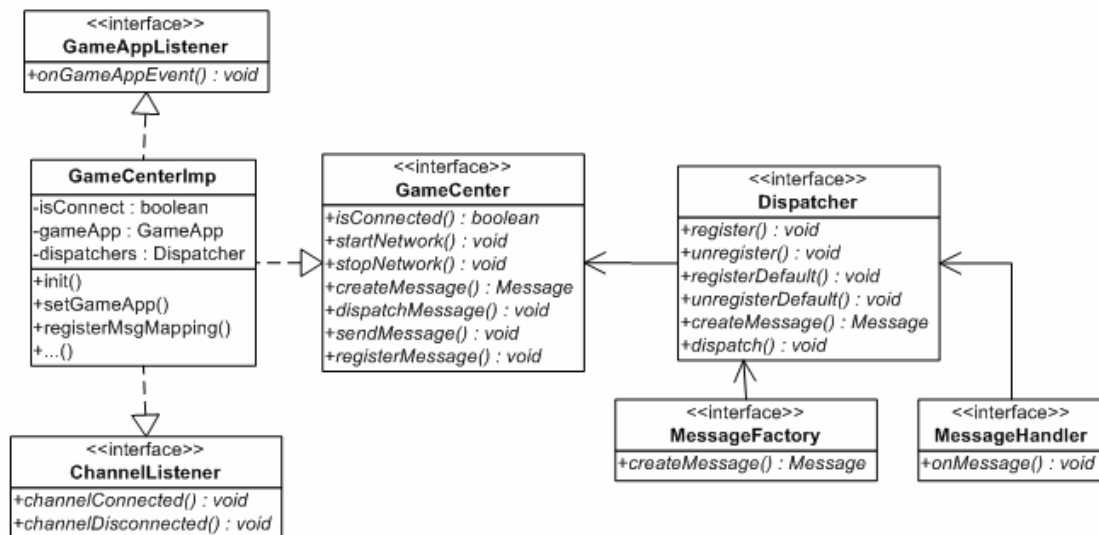


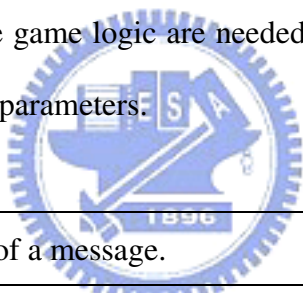
Figure 4-5 Class diagram of the GameCenter layer

As Figure 4-5 showed, the GameCenter works as the communication medium between Network and Application layer. Initially, the GameCenter register the message mapping, including the message type, message factory and message handler, to the Dispatcher. When a message is received, the network layer calls the createMessage() function to get the message template that created by the corresponding MessageFactory. When the message has formatted and packed, the dispatchMessage() function passes it to the Dispatcher. Then we dispatch the messages by calling the onMessage() method of the MessageHandler.

Furthermore, when the game application create the message and calls the `sendMessage()`, the GameCenter forwards it to the network layer and sends to the server-side. If in disconnected mode, it will be dispatched to the offline handler. By separating the dispatcher and handlers into different set, we can process the game logic in multiple modes. By implementing the `ChannelListener` and `GameAppListener` interfaces, the game center can also monitor the network connectivity and application events.

4.4 Message Class and Config File Generator

To build a message-oriented online game, it needs several messages with different types that present the game logic are needed. Each message is packed, with unique type and zero-to-many parameters.



Message Name	Name of a message.
Message Type	A unique number to identify the message in practice. It presents as an unsigned integer with range 0~65535.
Message Parameter	Customized data of a message. The parameters can be zero to many with Java primitive type. The order of parameters is related to the encoding/decoding implementation.
Message Factory	Use to create the message. (Factory Design Pattern)
Message Handler	Use to process the game logic of a message.

Table 4-1 The definition of message and related components

To define a message protocol, there are several fields and components, as the Table 4-1 shown. In order to speed up the development process, we use XML

(Extensible Markup Language) [22] syntax to define the message format and generate the corresponding class code and configuration file. The class code includes message and factory class with proper parameter encoding/decoding operations and setters/getters functions. The configuration file is used to register the message handler to the message dispatcher. The XML Document Types Definition is as follows (for full XML Schema Definition, please refer to Appendix 1A.1):

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT MDOIT_Message ( Version, PackageName, Messages )>
<!ELEMENT Version (#PCDATA)>
<!ELEMENT PackageName (#PCDATA)>
<!ELEMENT Messages ( Message+ )>
<!ELEMENT Message ( MessageName, MessageType, Params, Handlers )>
<!ELEMENT MessageName (#PCDATA)>
<!ELEMENT MessageType (#PCDATA)>
<!ELEMENT Params ( Param+ )>
<!ELEMENT Param ( ParamName, ParamType )>
<!ELEMENT ParamName (#PCDATA)>
<!ELEMENT ParamType (#PCDATA)>
<!ELEMENT Handlers ( OnlineHandler, OfflineHandler )>
<!ELEMENT OnlineHandler (#PCDATA)>
<!ELEMENT OfflineHandler (#PCDATA)>
```

The PackageName specify the Java package of the Message and MessageFactory classes. Each message includes name, type, parameters and handlers. Since we support to operate in a disconnected mode, the “Handlers” includes two types of message handler, one for online mode and one for another. For example, to define a MoveMessage with playerId and coordination of posX and posY, the content of XML file can be:

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE MDOIT_Message SYSTEM "xml/mdoit.dtd">
<MDOIT_Message>
  <Version>1.0</Version>
  <PackageName>demo</PackageName>
  <Messages>
    <Message>
      <MessageName>MoveMessage</MessageName>
      <MessageType>0x01</MessageType>
      <Params>
        <Param>
          <ParamName>playerId</ParamName>
          <ParamType>int</ParamType>
        </Param>
        <Param>
          <ParamName>posX</ParamName>
          <ParamType>int</ParamType>
        </Param>
        <Param>
          <ParamName>posY</ParamName>
          <ParamType>int</ParamType>
        </Param>
      </Params>
      <Handlers>
        <OnlineHandler>handler.MoveMessageHandler</OnlineHandler>
        <OfflineHandler>handler.LocalMessageHandler</OfflineHandler>
      </Handlers>
    </Message>
    <!-- Other Messages -->
    <Message>
      .....
    </Message>
  </MDOIT_Message>

```



By parsing this, we generate the Java message class file with corresponding member fields and functions. Here is the code segment of encoding/decoding functions.

```

public void decode(MessageBuffer bb) {
  playerId = bb.getInt();
  posX = bb.getInt();
  posY = bb.getInt();
}

public void encode(MessageBuffer bb) {
  bb.putInt(playerID);
  bb.putInt(posX);
  bb.putInt(posY);
}
// setters and getters of each member field
// .....

```

With the encode() and decode() function, the NetEngine can pack or unpack the message by correct order and format. Additionally, we create a configuration file as follow.

```
#<type>, <factory>, <handler>, <isOnlineMode>
0x01, demo.MoveMessageFactory, handler.MoveMessageHandler, true
0x01, demo.MoveMessageFactory, handler.LocalMessageHandler, false
#other message mapping
.....
```

While the Game Center initialize, it read the configuration file and register the message mapping. According to this, messages received from network or processed in offline line mode will be dispatched to correct handler.

4.5 Development Flow

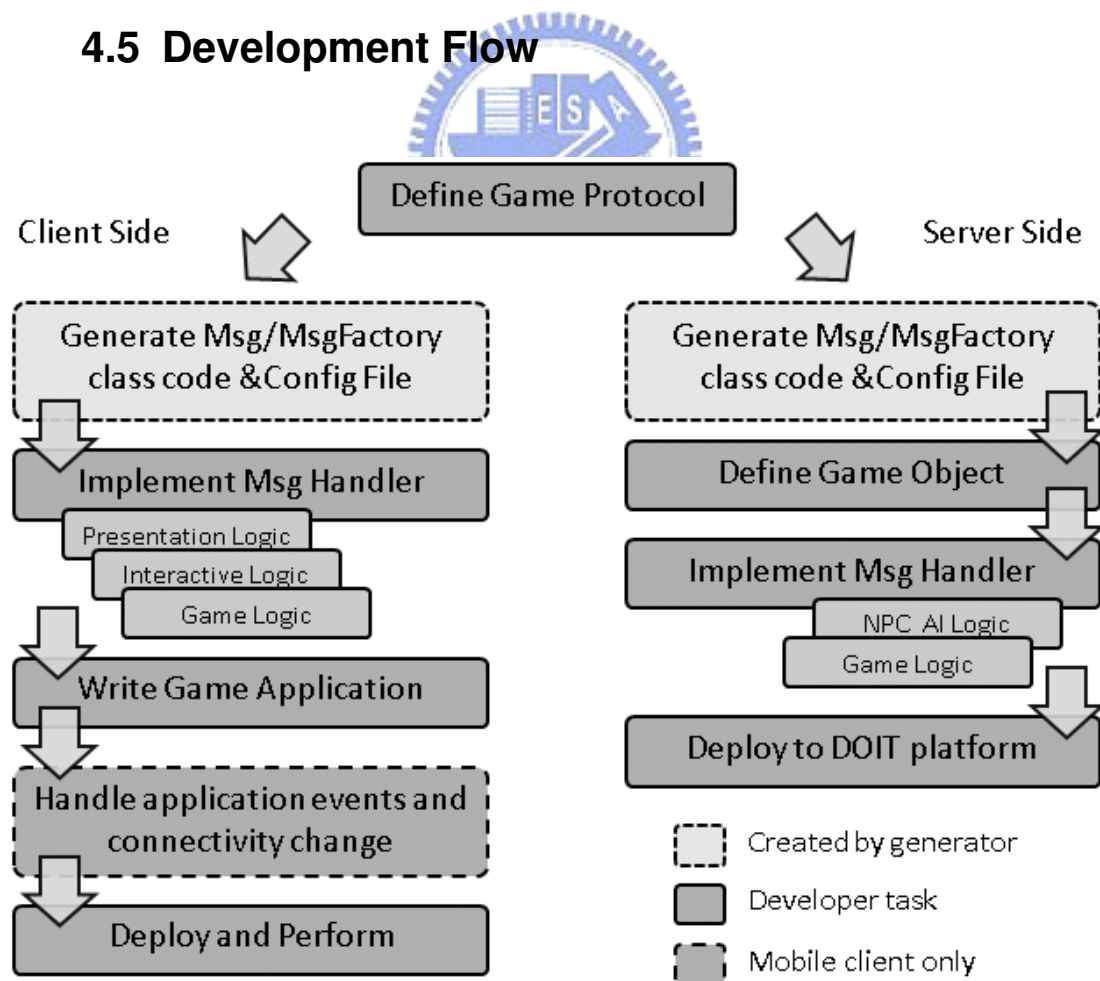


Figure 4-6 Development flow

The flow of developing a MMOG on our system is separated into client-side and server-side (Shown in Figure 4-6). First of all, we define the game protocol for the game content and create the message definition file. The message and message factory class codes and configuration file are generated according to the definition. For the server-side, we also need to define the game objects in the game world. By implementing both server-side and client-side message handlers, we can perform the full game logic. Then we write the client-side game application to perform the game presentation. For mobile version, we may need to handle the application events and network connectivity. Finally, all of the game protocols are deployed and form a MMO game world.



Chapter 5 Application Demonstration

To demonstrate the MMOG framework we propose, we implement a simple, 2-D tile-based online RPG game based on an open source mobile RPG engine. We implement the TCP/IP Network Engine to support the network communication at Java Me and MIDP application platform. Also, we implement the game center and design some game rules to show a simple online game with supporting of offline gaming. (For more implementation detail or development examples, please refer to Appendix 1A.2)

5.1 Mobile RPG engine



Figure 5-1 Screenshot of the mobierPG demo game

mobileRPG [23] is a 2D Role Playing Game (RPG) engine that targets at Java ME MIDP platform under GNU General Public License. It provides a set of dedicated syntax to create the game object, action script and game map for a RPG. Each game object is defined as an Entity and can set a list of ScriptAction to move automatically. The game world is two-dimension by using images to form a tile-based map. Figure

5-1 Screenshot of the mobieRPG demo gameshows the demo game built on mobileRPG with players and script Non-Player Characters (NPC) on the Town map.

5.2 Game rule and protocol

There are two kinds of game object in our online game. One is the Player that represents a user with playerId and 2-D coordination. Game players can move their character to four directions over the map. The other one is the NPC that moves randomly and automatically.

Message Name	Message Definition and Description
LoginMessage	Player login message, include username and password
LoginResultMessage	Server sends back the login result
LogoutMessage	Player is offline
PlayerMoveMessage	Player is moving
PlayerUpdateMessage	Player login, logout or move
NPCBornMessage	NPC is born and starts to walk around
NPCUpdateMessage	NPC is moving or updates
AttackNPCMessage	Player attacks NPC
NPCDieMessage	NPC is knocked out by a player
AFKMessage	Mobile player changes to pause mode
AFKBackMessage	Mobile player is back to active mode

Table 5-1 Game protocol of the demo game

The message definition of the demo game is shown in Table 5-1. When the players send the username/password and logins successfully, they can move over the

map and receive the surrounding information. One can see himself with other players and NPCs that move and interact on the game map. The NPCs will move to random direction and send the NPCUpdateMessage to the objects within AOI (Area of Interesting). The player can attack the NPC next to him and then the NPC will die and reborn somewhere. Since the MIDP application may change to different states, if the client application is interrupted, it sends an AFKMessage to report the temporary leave. When it backs to active mode, an AFKBackMessage is sent to notify other players.

5.3 Combine with PC version

The demo game build on the mobileRPG engine is a mobile online RPG over TCP/IP. Since we use message-oriented protocol to communicate, we can easily develop a PC version with the same game protocol and communicate to the same game server. To build the PC version, DOIT platform has provided the TCP NetEngine using Java SE technology. We just need to implement the client-side messages and handlers for PC version. Although the mobile and PC versions using the same protocol, the game presentation can be different and even provide distinct features. For example, as we have mentioned, the mobile client may support the offline service.

5.4 Offline Game Playing

In general, we assume that the game client must always keep the connection alive to access the MMOG service. Thus, we can offer the full functionality for the player over Internet. But the mobile user may disconnect intermittently, we can

provide a subset of game feature to support the offline gaming. In our demo game, when the game clients are forced to disconnect or stop the network manually, they changes to the offline mode. The online players and NPCs on the game map will disappear and then the scripted NPCs replace them. Under the disconnected mode, player can still move and interact with NPCs that controlled by the game center and offline message handlers. When the network connectivity is reset, the game client will regain the full online game service.

5.5 Implementation and result screenshot

Message Name	Type	Message Contents
LoginMessage	0x01	Username, password
LoginResultMessage	0x02	playerId, posX, posY
PlayerMoveMessage	0x03	posX, posY
PlayerUpdateMessage	0x04	updateCode, playerId, posX, posY
AFKMessage	0x05	playerId
AFKBackMessage	0x06	playerId
NPCBornMessage	0x0A	npcId
NPCUpdateMessage	0x0C	npcId, posX, posY
NPCDieMessage	0x0E	npcId
AttackNPCMessage	0x0F	npcId
LogoutMessage	0xFF	

Table 5-2 Message types and contents

The message types and contents are shown in Table 5-2. We have deployed the game on some mobile phones, such that Nokia 6111, Sony Ericsson Z800i and

Motorola E1070, and through 3G or GPRS to communicate with server. Following, we show some screenshots of the demo game emulated on Sun Java™ Wireless Toolkit 2.5 for CLDC. When the MIDlet is loaded, it shows a form to fill in the login information and connect to server:

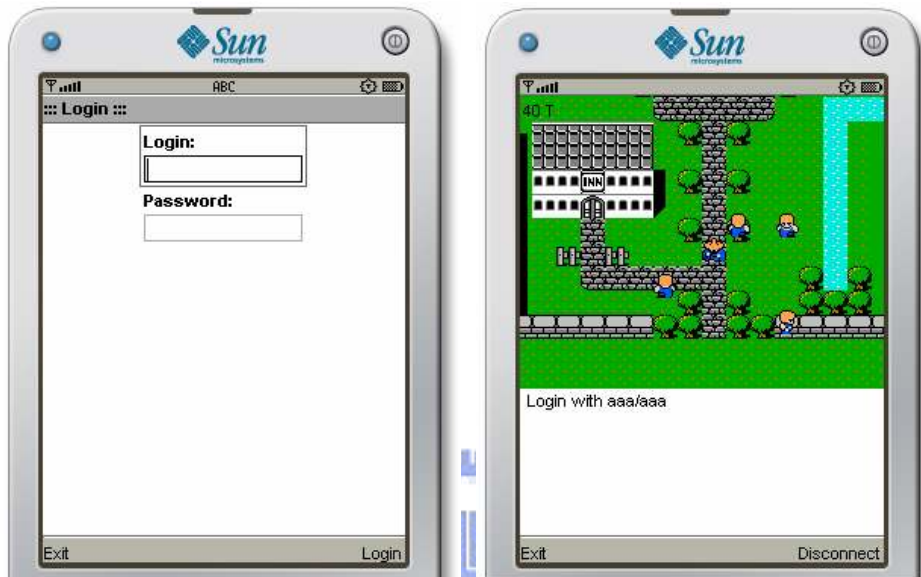


Figure 5-2 Login form and game world



Figure 5-3 Player in the paused state

The white space in the bottom is the message box and the character in the center is the player himself with other NPCs around. When other players login, we can get the player update message and locate the position of them. Within the AOI (area of interesting), they broadcast their information and update. When some players change to paused state, others will receive an AFKMessage and then display the corresponding character with gray scale (as shown in Figure 5-3).

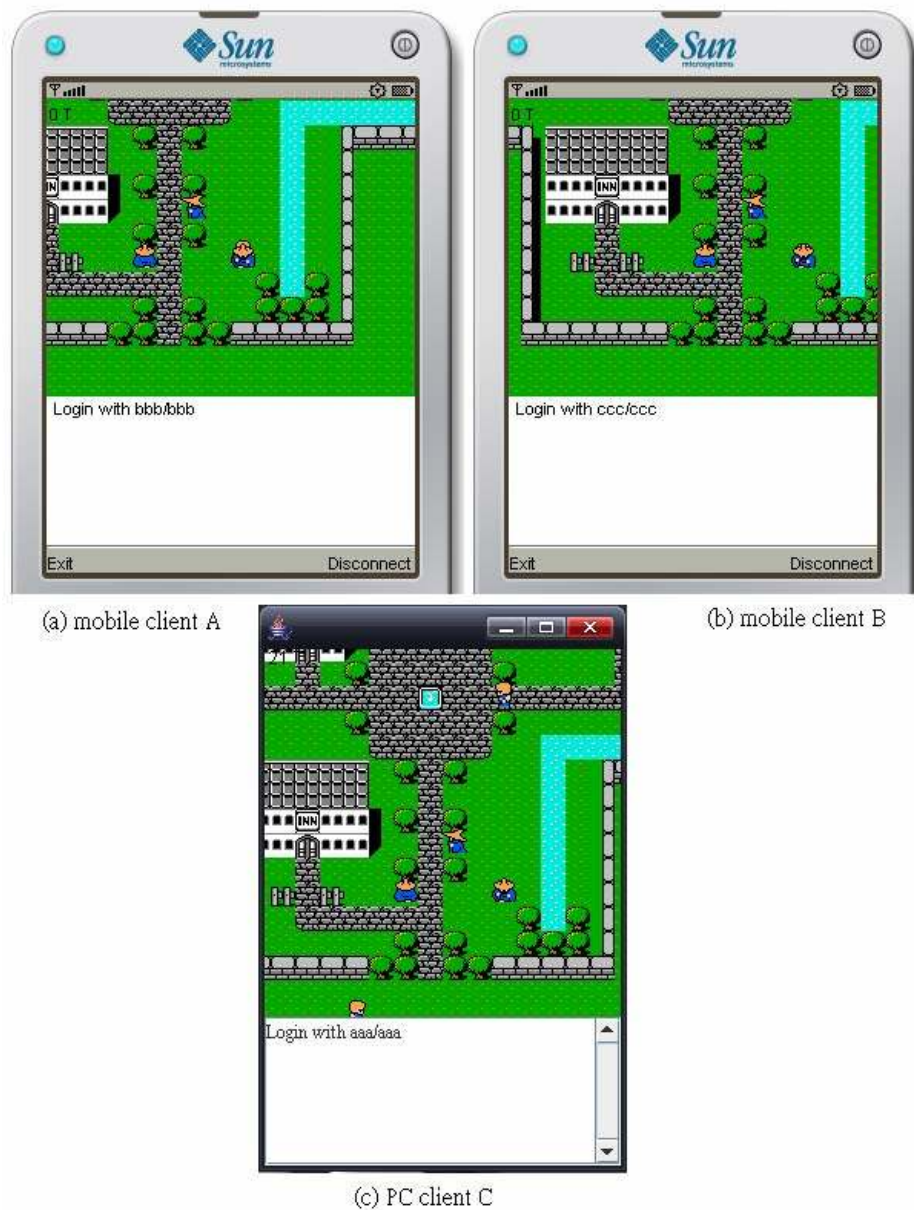


Figure 5-4 Demo game on both mobile and PC clients

For PC version, we implement it by using Java SE and Swing/AWT technology to present the game world. Figure 5-4 shows the demo game running on different platforms. They share the same game world and can locate and identify each other. We assume that the PC client should be always online and the offline gaming is only supported in the mobile version. Moreover, instead of offering the same functions on both sides, we can let the mobile client to support a special part or a subset of full game functionality. The combination of the mobile and PC client provides a novel way to design an online game.



Chapter 6

Conclusions and Future Works

6.1 Conclusions

It is not an easy task to develop a middleware platform for MMOGs on mobile devices. We have figured out the problems that may encounter when designing a MMOG on mobile handsets. In the following parts, we discuss and analyze our system by the objectives we mention in section 1.3

- **Easy to Adapt and Extend**

To offer a flexible game development platform, we use message-based communication mechanism and reserve as few platform-specific data as possible. Also, we provide the most resilience for defining the game objects, application events and network protocol. Game developers can easily map the game logic to message-oriented protocol and implement the game objects, events and low-level networking to form the virtual game world.

- **Easy to Develop and Deploy**

Development speed and time to market is quite important for game provider. We provide a dedicated XML syntax to define the message protocol and generate the underlying programming codes and configuration files. This makes the development of game protocol faster and the deployment of the changeful MMOG contents easier. Moreover, it also helps to alter the message contents effortlessly to prevent game protocol cheating.

- **Dealing with the Mobile-specific issue**

The mobile clients have more uncertain issues than PC clients. We use the observer design pattern to monitor the application status and network connectivity. Game developer can define their own events and listeners to trace and process. To handle the intermittent connectivity on mobile devices, we provide a mechanism to support disconnected game playing and can easily switch to connected mode when network quality restored.

- **Combination with mobile and traditional MMOG**

By deploying the MMOGs on DOIT middleware, we can provide a game world for diverse client platforms with the same communication protocol. In our demo game, we have show an actual example of this kind of MMOG, but also keep the flexibility to offer different game play under some situation. Although most of existing MMOGs are centered on one kind of end device, the combination of different client devices can bring out another opportunity of game playing.

6.2 Future Works

In current work, we focus on the client-side framework for mobile device that offers the basic interfaces to design a MMOG. However, there are still plenty of problems should be taken into consideration.

In server-side, we can provide a mobile-specific proxy (gateway) to process the connection from mobile clients. The proxy can wrap the message to distinguish the data come from or sent to the mobile clients and provide different game service. Since the mobile clients may be paused or lose connectivity intermittently, the proxy can help to keep the connection information to hand over and even caches it to reduce the

back-end data access. Furthermore, we can even provide different mobile-specific proxy to support different mobile client platform and help to handle client application status change. For example, when MIDlet clients change to paused-mode, the proxy can buffer the message to prevent message lose.

For the client-side, we just design the framework with three layers and provide interfaces to be implemented. But there are still some common issues should be encountered. For example, since the client application may be interrupted regularly, to avoid the resource missing, we should provide a serializable object interface to store and restore the mobile resource we hold.



Chapter 7 Reference

- [1] International Game Developers Association (IDGA)
<http://www.igda.org/>
- [2] IDGA, July 2005, IDGA 2005 Mobile Games White Paper
http://www.igda.org/online/IGDA_Mobile_Whitepaper_2005.pdf
- [3] Ultima Online - <http://www.uoherald.com/>
- [4] Lineage - <http://www.lineage.com/>
- [5] World of Warcraft - <http://www.worldofwarcraft.com/>
- [6] Java™ Platform, Micro Edition (Java ME)
<http://java.sun.com/javame/>
- [7] Connected Limited Device Configuration 1.1
<http://jcp.org/aboutJava/communityprocess/final/jsr139/>
- [8] Mobile Information Device Profile 2.0
<http://jcp.org/aboutJava/communityprocess/final/jsr118/>
- [9] BREW - <http://brew.qualcomm.com/brew/en/>
- [10] Symbian - <http://www.symbian.com/>
- [11] Windows Mobile - <http://www.microsoft.com/windowsmobile/>
- [12] Chen-en Lu, Tsun-Yu Hsiao, Shyan-Ming Yuan. Design issues of a Flexible, Scalable, and Easy-to-use MMOH Middleware. In Proceeding of Symposium on Digital Life and Internet Technologies 2004.

- [13] Tsun-Yu Hsiao, Shyan-Ming Yuan, "Practical Middleware for Massively Multiple Online Games", IEEE Internet Computing (SCI), Volume 9, Issue 5, Sep/Oct 2005, pp.47-54
- [14] Chen-en Lu, Shyan-Ming Yuan. "Design Issues of a Flexible, Scalable, Easy-to-use MMOG Middleware", 國立交通大學，資訊科學系碩士論文，民國 93 年 6 月
- [15] Lun-Wu Yeh, Shyan-Ming Yuan. "A Research of Persistence Component on MMOG Middleware", 國立交通大學，資訊科學系碩士論文，民國 94 年 6 月
- [16] Ko-Hsu Su, Shyan-Ming Yuan. "A Framework of MMOG Development and Management System", 國立交通大學，資訊科學系碩士論文，民國 93 年 6 月
- [17] BigWorld Technology - <http://www.bigworldtech.com/>
- [18] Ex Machina - <http://www.exmachina.nl/>
- [19] GASP - <http://gasp.objectweb.org/>
- [20] R. Pellerin, F. Delpiano, F. Duclos, E. Gressier-Soudan et M. Simatic. GASP: an open source gaming service middleware dedicated to multiplayer games for J2ME based mobile phones. In 7th International Conference on Computer Games, November 2005
- [21] Sun Microsystems, Inc., June 2003, Supporting Disconnected Operation in Wireless Enterprise Applications
http://java.sun.com/blueprints/guidelines/designing_wireless_enterprise_applications/index.html

[22] Extensible Markup Language - <http://www.w3.org/XML/>

[23] Mobile RPG Engine - <http://sourceforge.net/projects/mobilerpg/>



A.1 XML Schema Definition of Message Protocol

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="MDOIT_Message" type="mmog_message_type">
  </xs:element>
  <xs:complexType name="mmog_message_type">
    <xs:sequence>
      <xs:element name="Version" type="xs:string">
      </xs:element>
      <xs:element name="PackageName" type="xs:string">
      </xs:element>
      <xs:element name="Messages" type="messages_type">
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="messages_type">
    <xs:sequence>
      <xs:element name="Message" type="message_type"
        minOccurs="0" maxOccurs="unbounded">
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="message_type">
    <xs:sequence>
      <xs:element name="MessageName" type="xs:string">
      </xs:element>
      <xs:element name="MessageType" type="xs:string">
      </xs:element>
      <xs:element name="Params" type="params_type">
      </xs:element>
      <xs:element name="Handlers" type="handlers_type">
      </xs:element>
    </xs:sequence>
  </xs:complexType>
  <xs:complexType name="params_type">
    <xs:sequence>
      <xs:element name="Param" type="param_type"
        minOccurs="0" maxOccurs="unbounded">
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:schema>
```

```

<xs:complexType name="handlers_type">
  <xs:sequence>
    <xs:element name="OnlineHandler" type="xs:string">
    </xs:element>
    <xs:element name="OfflineHandler" type="xs:string">
    </xs:element>
  </xs:sequence>
</xs:complexType>
</xs:schema>

```

A.2 Development Example

In this section, we describe the implementation detail of the demo game as an example of developing a game on our framework.

A.2.1. Game Application

The mobileRPG is a game engine for Java ME MIDlet games. Every MIDlet application should extend the MIDlet class to be managed by the application management software (AMS). In the original mobileRPG source code packages, it is the Main class that extends this class. In the section 4.2 we describe the application interface of our framework. By extending the Gamelet class, the mobileRPG application can easily be supported by the framework.

```

import mmog.game.Gamelet;
.....

public class Main extends Gamelet implements CommandListener,
Runnable
{
  .....
}

```

To monitor the MIDlet life cycle, we can register the GameAppListener and handle when the MIDletStateChangeEvent occurs.

A.2.2. Game Center

The game center works as a communication medium of network and application layers. One can design his own game center by implementing the game center interface. In our demo game, we implement a game center that supports different gaming modes and monitors the network status. It implements the GameCenter and ChannelListener interfaces and initialize the network engine. The game application can start or stop network connection and send messages through the public methods it provides.

```
// some example codes in the Main class

// initial game center and register GameApp
this.gameCenter = new GameCenterImp();           // construct
this.gameCenter.setGameApp(this);               // register GameApp

// send logout message
LogoutMessage msg = new LogoutMessage();        // create message
this.gameCenter.sendMessage(msg);               // send
```

When the network status changed, the game center will call the onConnected or onDisconnected functions of the GameApp.

```
// in the GameCenterImp
// callback function of the ChannelListener
public void channelDisconnected(ChannelEvent e) {
    this.setConnected(false);
    System.out.println("Network connection fail");
    this.netEngine.close();
    this.game.onDisconnected();           // notify the GameApp
}

// in the GameApp (Main)
public void onDisconnected() {
    // handle the disconnected situation
    .....
    this.alertScreen(map, "Warning!! Network is disconnected!");
    .....
}
```

A.2.3. Implement the message handler

In the section 4.4 we propose a method to define the game protocol and generate the message class codes automatically. The message and message factory class only solve the problem of sending and receiving data through the network engine. To process the game logic, we must implement the corresponding message handlers. The message handler interface is as below:

```
public interface MessageHandler {  
  
    /**  
     * Initiation.  
     * By overriding this method, we can obtain the game center.  
     * @param gameCenter The GameCenter  
     */  
    public void init(GameCenter gameCenter);  
    /**  
     * Receive a message.  
     * @param msginfo The message info  
     */  
    public void onMessage(MessageInfo msginfo);  
}
```

The `init()` function is called when the handler is first loaded and registered by the GameCenter. While the game center receives the messages and dispatches to the corresponding handler, the `onMessage` function will be fired. If the handler just processes the simple game logic, it may only need the information contains in the received message. But if it needs to present some graphical interaction, it can get the GameApp through the GameCenter that obtained in the `init` function and operate more complex functions. Following is an example of LoginHandler that handle the login result from the server.

```
public class LoginHandler implements MessageHandler {

    private GameCenter gameCenter;
    private Main main;

    public void init(GameCenter gameCenter) {
        // obtain the game center and get the GameApp (Main)
        this.gameCenter = gameCenter;
        this.main = (Main) gameCenter.getGameApp();
    }

    public void onMessage(MessageInfo msginfo) {
        System.out.println("Receive LoginResultmessage.....");
        // get the LoginResultMessage
        LoginResultMessage msg =
            (LoginResultMessage) msginfo.getMessage();
        main.setPlayerData(
            msg.getPlayerid(), msg.getX(), msg.getY());
        main.startGame();
    }
}
```

