A Framework for Developing Network based Games using Unity and Ice

Master Thesis

Yongjian Wang

Madrid, June 2017
This thesis is submitted to the ETSI Informáticos at Universidad Politécnica de Madrid in partial fulfilment of the requirements for the degree of Master of Science in Software Engineering.
Master Thesis
Master Universitario en Ingeniería del Software – European Master in Software Engineering

Thesis Title: A Framework for Developing Network based Games Using Unity and Ice

Thesis no: EMSE-2017-02
June 2017

Author: Yongjian Wang
Student
Universidad Politécnica de Madrid

Supervisor: Co-supervisor:
Jaime Ramírez Rodríguez Diego Riofrío Luzcando
Professor in Computer Science PhD. in Computer Science
Universidad Politécnica de Madrid Universidad Politécnica de Madrid

DLSIIS
ETSI Informáticos
Universidad Politécnica de Madrid

Human-Computer Interaction and
Advanced Interactive Systems
Group – Decoroso Crespo Lab
ETSI Informáticos
Universidad Politécnica de Madrid

ETSI Informáticos
Universidad Politécnica de Madrid
Campus de Montegancedo, s/n
28660 Boadilla del Monte (Madrid)
Spain
Abstract

Nowadays, Unity 3D is the dominant game engine in the market, because it supports the development of cross-platform games with a high productivity. In addition, Ice is a RPC framework with multiple programming languages mappings and a good support for load balancing and security.

Although Unity framework provides networking services for developing network games, these services are too low level for facilitating the development of multiuser online games. In this multiuser games, typically users control their avatars and visit virtual worlds in which they can interact with 3D objects, chat with other users or manage their own inventories of collected objects.

This project presents an online environment framework that is based on Unity and Ice RPC framework. With this framework, developers can build their own multiuser online games or simulation environments more easily than using directly Unity framework.

The proposed framework provides three kind of general components: client, Scene server and Online server. Clients and Scene servers implement the game logic relying on Unity framework. Basically, clients are responsible of player interaction and rendering whereas Scene servers keep the consistency of the different client views working with shared (network) game objects. On the other hand, Online server is responsible of the game access, the player account management and the persistency of the game state across different scenes and game sessions. In addition, the Online server will support the division of a virtual world into several scenes provided by different Scene servers. In this context, Ice is employed as a middleware between the Online server and the other two kind of components.

1 https://github.com/zeroc-ice
# Table of Contents

1. Introduction ................................................................................................................. 1

2. Background .................................................................................................................. 3
   2.1 Game Industry Overview ....................................................................................... 3
   2.2 General Network Based Games Overview .......................................................... 4
   2.3 Game Engines Overview ....................................................................................... 4
      2.3.1 Game Engine History ....................................................................................... 4
      2.3.2 Current Game Engines .................................................................................... 5
   2.4 The Unity Game Engine ......................................................................................... 5
      2.4.1 Unity Game Engine Market ............................................................................ 5
      2.4.2 Unity Training ................................................................................................. 6
      2.4.3 Unity Community ............................................................................................ 7
      2.4.4 Unity Asset Store ............................................................................................ 7
      2.4.5 Unity Utilities ................................................................................................. 8
      2.4.6 Unity License ................................................................................................. 8
   2.5 Support for networking in Game Engines ............................................................... 9
      2.5.1 Architectures of Network Game Engines ....................................................... 9
      2.5.2 Unity Networking ............................................................................................ 10
      2.5.3 AssetBundles ................................................................................................. 13
   2.6 Ice Overview .......................................................................................................... 13

3. Requirements Specification ......................................................................................... 17
   3.1 Scope of the Project ............................................................................................... 17
   3.2 Product Perspective ............................................................................................... 17
   3.3 Hardware and OS Platform Requirements ............................................................ 18
      3.3.1 Servers Hardware and OS Platform Requirements ......................................... 18
      3.3.2 Clients Hardware and OS Platform Requirements ......................................... 18
   3.4 Functional Requirements ....................................................................................... 18
   3.5 Non-Functional Requirements ............................................................................. 26

4. Development of the Solution ..................................................................................... 27
   4.1 Architecture of the Project ..................................................................................... 27
   4.2 Online and DataBase Server Design ...................................................................... 28
      4.2.1 DataBase Schema and Persistence Design .................................................... 28
      4.2.2 Online Services .............................................................................................. 30
      4.2.3 Slice Definition of Online Services ................................................................ 33
   4.3 Scene Server Design .............................................................................................. 34
      4.3.1 Unity Scene Server .......................................................................................... 34
      4.3.2 AssetBundle Server Scene .............................................................................. 36
      4.3.3 Utility Tools .................................................................................................... 37
   4.4 Game Client Design ............................................................................................... 38
      4.4.1 Unity Game Client .......................................................................................... 38
      4.4.2 Client Operations related to Online Server ..................................................... 39
4.4.3 Client Game Operations ................................................................................. 40
4.5 Game Client and Scene Server Interaction ..................................................... 41

5. Testing.................................................................................................................. 45
5.1 Runtime Environment Preparation .................................................................... 45
5.2 Demo Project...................................................................................................... 45

6. Developer Guide .................................................................................................. 51
6.1 Development Environment Preparation .......................................................... 51
   6.1.1 Online Server Environment Preparation .................................................. 51
   6.1.2 Client and Scene Server Environment Preparation ...................................... 52
6.2 Communication Data Layers ............................................................................ 52
6.3 Begin to Implement New Online Functions ..................................................... 54
6.4 Begin to Implement New In-Game Logic .......................................................... 57

7. Conclusions and Future Work ............................................................................ 59
7.1 Conclusion ......................................................................................................... 59
7.2 Future Work ...................................................................................................... 59

8. References .......................................................................................................... 61

9. Appendix ............................................................................................................. 62
9.1 Asset Store Resource Used in the Project ......................................................... 62
9.2 Tools, Frameworks, Libraries Used in the Project ............................................ 62
1. Introduction

Nowadays, we use smart TVs, computers, smartphones, tablets, smartwatches, VR devices, and these devices use different software platforms, Windows, Linux, macOS, iOS, tvOS, Android, Windows Phone, and so on. It’s almost impossible to develop a product with each different version for each platform. So, it is very important that developers use cross platform frameworks, libraries, build tools, technologies. That is one of the main reasons why Unity 3D has become in the dominant game engine in the market [1].

Unity has a very good performance on resources constrained devices comparing to other game engines such as Unreal Engine 4 or CryEngine. Unity is free or nearly free for personal developers or small teams, even for big companies, the license fee is quite fair. Unity itself is not an open source project. However, the source code of its main components like UI and Networking is available in Bitbucket².

Although Unity framework provides networking services for developing network games, these services are too low level for facilitating the development of multiuser online games. In this multiuser games, typically users control their avatars and visit virtual worlds in which they can interact with 3D objects, chat with other users or manage their own inventories of collected objects.

Ice RPC framework is also an easy to use framework, which provides eight programming languages mapping. Moreover, Ice is an open source project and its source code is available in Github.

This project presents an online environment framework that is based on Unity and Ice³ RPC framework. With this framework, developers can build their own multiuser online games or simulation environments more easily than using directly Unity framework.

The proposed framework provides three kind of general components: client, Scene server and Online server. Clients and Scene servers implement the game logic relying on Unity framework. Basically, clients are responsible of player interaction and rendering whereas Scene servers keep the consistency of the different client views working with shared (network) game objects. On the other hand, Online server is responsible of the game access, the player account management and the persistency of the game state across different scenes and game sessions. In addition, the Online server will support the division of a virtual world into several scenes provided by different Scene servers. In this context, Ice is employed as a middleware between the Online server and the other two kind of components.

The structure of the remainder of the documents is as follows. In chapter 2, I will present the background of the work including an introduction to Game Engines and Unity Engine, the main architectures of Network Games and an overview on Ice RPC framework. In chapter 3, I will present the project requirements specification including Hardware and OS Platform Requirements, Functional Requirements and Non-Functional Requirements. In chapter 4, I will explain the

² https://bitbucket.org/Unity-Technologies
³ https://github.com/zeroc-ice
development of the project including the architecture of the solution and the design of its main components, Online and Database Server, Scene Server and Game Client. In chapter 5, I will outline the testing of the project including the runtime environment preparation and the Demo Project implemented for testing the framework. In chapter 6, I will present a developer guide for future developers who want to use the framework to implement a network game. In chapter 7, I will end with some conclusions and the future work.
2. Background

2.1 Game Industry Overview

According to SuperData Research “Year in Review 2016”[1], the global revenue from game industry of 2016 is 91 billion US dollars. And the number of Mobile game is 40.6 billion US dollars, PC game is 35.8 billion US dollars. The share of Mobile game grows quickly and surpasses PC game.

Figure 1.1 shows mobile game revenue prediction from 2016 to 2019. The growth rate will be at least 10 percent per year.

![Mobile game revenue prediction](image)

Figure 1.1 Mobile game revenue prediction, region distribution and top5 mobile games [1]

There are many respected and attractive game companies such as Electronic Arts, Nintendo, Sony, Capcom, Activision Blizzard, Ubisoft, SuperCell, etc. Game industry has produced a lot of revenue, and provided numerous jobs.

Due to the many available game engines in the market, it was never easier than now to develop a game. Despite the fact that the big companies have been very successful, in these days there have arisen many video games of small developer teams. SuperCell was a very small team of 12 people when they put their first incredible game COC into the market. Rovio developed Angry Bird when they were a small start-up. Temple Run is a mobile game which earned billions of dollars, being just a product of a husband and a wife. Moreover, there was a miracle for a single independent developer who developed Flappy bird and earned 60 thousand dollars per day. More and more independent game developers, college students, freelance developers are developing games.
2.2 General Network Based Games Overview

As can be seen in Figure 1.1, the top 5 mobile games are Monster Strike, Clash of Clans, Clash Royale, Game of War: Fire Age and Mobile Strike. All of them are network based mobile games.

The first video computer games can trace back to 1950s [2], at that time, one or two players sitting together in front of a computer play a game. Later on, thanks to local area networks, multiple players can play a game in the same local area network. After the creation of Internet in 1983, that was the first time that multiple players could play together without location limit [3].

Comparing to non-networked games, networked games have the appealing of that multiple humans can compete with each other or cooperate with each other [3]. Massive players, friends and strangers from different places, even different countries can play the same game together. In addition, as all the user data is stored in servers, game players can access their data from different devices.

Due to the many benefits of being network based, game companies almost always make their games be network based. And almost all the game engines provide network facilities to develop network based games.

2.3 Game Engines Overview

2.3.1 Game Engine History

Before game engines, each game was written independently, therefore all the basic features need to be rewritten every time. The famous game engines such as id software’s DOOM an Quake, Epic Game’s Unreal and Valve’s Source engine have changed game development. Game engines have become fully featured reusable software development kits, which can be licensed and used to build almost any imaginable game [4].

Game companies develop their own game engines; they use their own game engines to develop their own games. Later on, as some of these game engines turn out to be so successful, other game companies also want to use them and this is a way to allow the owner game companies to make money from other game companies. In this way, some game companies give up their in-house game engines and turn to use those success game engines.

As time passes, some very famous and success game engines evolve several generations. Some examples of these game engines are Unreal Engine 4\(^4\), Unity 5\(^5\), CryEngine 3\(^6\), etc.

All these game engines can be reused by many games. Now more and more game companies tend to use these success game engines instead of developing their own game engines from scratch.

---

\(^4\) https://www.unrealengine.com
\(^5\) https://unity3d.com
\(^6\) https://www.cryengine.com
2.3.2 Current Game Engines

Nowadays, the core typical functionalities provided by game engines are a rendering engine, a physical engine, sound system, animation system, networking, etc. In addition, one of reasons why game engines have become more popular among game developers is because of their support for high level programming languages such as C#, Java, Python, etc.

Game engines are not only for games development but also have been applied to other fields such as visualization, training and military simulation.

Among hundreds game engines out there, game engines such as Unreal Engine, CryEngine, Source Engine and Unity are currently very popular. Their target platforms are Windows and game consoles such as Xbox, PlayStation and Nintendo. But now after the boom of mobile devices, Unity became the dominant global game development platform. Unity is easy to use and covers almost all platforms: Mobile, Desktop, AR, VR, Console, Web, Smart TVs.

The trends of game engines are to support different platforms, especially to improve mobile platform performance, and to support the latest numerous VR devices and continuous make the best use of the latest hardware to improve graphic performance. In this sense, every year at the Game Developers Conference (GDC), new technologies and games are introduced. So, in the last GDCs, virtual reality and augmented reality have become a trend. VR devices such as Oculus Rift, Samsung Gear VR, HTC Vive, PlayStation VR, have been introduced into the game market. In addition, all the most popular game engines are supporting these VR devices.

All the game engines are trying to introduce better performance and make the best use of new hardware to compete each other. And because of Unity’s big success, game engines are opening their license policy to be more beneficial to their users. Unreal Engine 4 made itself open source to its users, so that any user can access its source code in Github and use for free if their revenue is below 3000 US dollars.

2.4 The Unity Game Engine

2.4.1 Unity Game Engine Market

The Unity game engine is, by far, the dominant global game development platform.

7 https://cinema-suite.com/5-best-uses-for-unity/
8 https://www.oculus.com/
9 http://www.samsung.com/global/galaxy/gear-vr/
10 https://www.vive.com/eu/
12 https://github.com/EpicGames
13 https://thenextweb.com/gaming/2016/03/24/engine-dominating-gaming-industry-right-now/#.tnw_DWTDMpKQ
2.4.2 Unity Training

Unity programmers can use C#, UnityScript or Boo as programming languages. On the official website, a lot of free training resources are available. For beginners, tutorials are the best start place. These tutorials are categorized by topics; it is easy to find what material the developer need in every moment. The Unity Documentation is the best place to search the description about Unity APIs.
2.4.3 Unity Community

Unity has a great community to help developers to understand and use Unity efficiently.

Unity has discussion forums about different modules of Unity, and questions and answers forum to let users find the answers from both other developers and the official unity developers. This forum is also used to announce new beta and experimental versions.

2.4.4 Unity Asset Store

Unity asset store provides 15000+ free and paid-for 3D models, editor extensions, scripts, shaders, materials, audio files, animations and more to help developers to power up their Unity projects. It is the best place to find some basic assets to start a prototype or demo.

Unity editor has some included standard assets, which developers can use to create prototypes or demos, and even a whole project. Thanks to Unity standard assets and lots of free and paid-for assets in Unity asset store, a small team formed only by programmers with limited budget can start and finish a project without 3D designers.

---

2.4.5 Unity Utilities

Figure 2.5 Unity utilities\textsuperscript{15}

Unity is not just a game engine to develop games, it also integrates many useful unities and services to help game developers to analysis their games, to improve their developing process, to provide fundamental infrastructure services to easily create multiplayer games that allow them to monetize their games.

2.4.6 Unity License

Figure 2.6 Unity Licenses\textsuperscript{16}

Unity provides 4 licenses for different type of developers. For small teams or individual developers, personal license is free. For anyone who just begins to use Unity, personal version is the best

\textsuperscript{15} https://unity3d.com/services
\textsuperscript{16} https://store.unity.com/
choice. Unity’s license policy is very successful. That is one of the reason why it is dominating the game engine market. For this, other game engines as Unreal had to change their own old license policy.

2.5 Support for networking in Game Engines

2.5.1 Architectures of Network Game Engines

Network game engines has two architectures, host mode architecture and client-server architecture [7]. As it is shown in figure 2.7.

![Figure 2.7 Host mode architecture [8]](image)

![Figure 2.8 Client-server architecture [8]](image)

Usually a mature game engine would support both of these two architectures. The host mode architecture saves a lot of server hardware resources, because each player’s computer act as a client and a server at the same time. However, it also introduces a poor user experience if the player’s computer is not stable and there exists the risk of that some player’s computer may cheat to the host. The client-server architecture, on the other hand, provides a better user experience and security policy because servers are under control, in exchange of increasing developer’s expense.

When Unity supports massive multiplayer online games, it is almost impossible to run it at host mode, because it is common that normal player’s computers are not power enough to act as servers.
2.5.2 Unity Networking

Unity can be used to develop 2D/3D stand-alone games or network multiuser games. In both cases, the virtual world is divided into scenes by developers and depending on the game logic and the player interaction, these scenes are activated or deactivated.

In network games, Unity game engine encapsulates interaction operations on network based game objects.

Depending on who is the interaction active party, Unity provides different methods [9]: Commands and ClientRpc.

**Commands:** Called from the client and run in the server.
For security reasons, each player can only control his/her own objects. Therefore, if a client needs to change the state of a shared object (network object), the client player will have to use a Command to ask scene server to do it.

To use Commands, we need to add the [Command] custom attribute, and add the “Cmd” prefix to the method name.

```csharp
[Command]
public void CmdSaveState()
{
    GameServer gameServer = DataMaster.GameServer;
    AvatarState avatarState = new AvatarState();
    avatarState.position = DataUtils.ToRmiPosition(transform.position);
    avatarState.rotation = DataUtils.ToRmiRotation(transform.rotation);
    string servIPPort = gameServer.networkAddress;
    servIPPort += ";" + gameServer.networkPort;
    avatarState.serverEntry = new ServerEntry(gameServer.SceneLabelName, servIPPort, NetworkManager.networkSceneName);
    GamePlayer gamePlayer = GetComponent<GamePlayer>();
    gameServer.avatarManager.UpdateAvatar(gamePlayer.userName, avatarState);
    Debug.Log("CmdSaveState from: " + gamePlayer.userName);
    Debug.Log(string.Format("transform.position:({0}, {1}, {2})",
        transform.position.x,
        transform.position.y,
        transform.position.z));
}
```

Figure 2.11 Unity Network Commands

Figure 2.11 shows the client wants to save the current state, the client calls CmdSaveState and the scene server will invoke the method and save the client’s state. The scene server saves the client’s state is because that the client is not reliable, only scene server has the authority.

When using this kind of interaction, it is important to be aware of the resulting network traffic and avoid to send commands from the client every frame.

**ClientRpc Calls:** Called from the server and run in clients

To use ClientRpc we need to add the [ClientRpc] custom attribute to it, and add the “Rpc” prefix to the method name. ClientRpc calls are sent from objects on the server to objects on clients. They can be sent from any server object with a NetworkIdentity that has been spawned.
Figure 2.12 shows an example to use ClientRpc, the scene server calls RpcJump2AnotherServerNew, so the corresponding client will invoke this method and jump to a new scene server.

**Arguments to Remote Actions:** Arguments to remote actions cannot be subcomponents of game objects, such as script instances or Transforms.

For remote actions depending on who is the active party we can use either Command or ClientRpc. However, for network objects attributes like player’s HP, level, items, etc. if we need to synchronize them in all the clients, we can use SyncVars, SyncList to synchronize attributes.
Unity provides High Level API (HLAPI)\(^{17}\) to build multiplayer Unity games. It is built on the top of Low Level API of transport communication layer. HLAPI aims to make easier the development of multiplayer Unity games. It provides useful services for multiplayer games as follows [10].

- *Message handlers*
- *General purpose high performance serialization*
- *Distributed object management*
- *State synchronization*
- *Network classes: Server, Client, Connection, etc.*

**Note:** All game objects that are intended to exist on all the clients will appear only when the server calls NetworkServer.SpawnObjects method. All objects with component NetworkIdentity in a scene are disabled by default.

### 2.5.3 AssetBundles

For online games, packages could have a considerable size, which could cost a lot of time to download before user could play the game. However, a lot of the game content is not a prerequisite for beginning to play. For this, Unity provides AssetBundles to solve this problem.

In the first release, we may provide a small package that contains only the essential content to begin to play and later the additional content will be downloaded when the game requires it. In Unity, this additional content can be packed into an AssetBundle. AssetBundles can contain asset files such as models, materials, textures and scenes, but it cannot contain scripts. AssetBundles must be built for each target platform.

### 2.6 Ice Overview

Ice is a comprehensive RPC framework with support for C++, C#, Java, JavaScript, Python, etc. Ice provides Interface Description Language (IDL) slice to define method signatures exposed by the servants and the data structures to share between clients and servers. Ice also provides several components to provide different features: IceNode is used to implement needed services; IceGrid is used to manager IceNodes; and Glacier2 is used to do firewall traversal and session manager [11].

---

\(^{17}\) https://docs.unity3d.com/Manual/UNetUsingHLAPI.html
Ice provides tools like slice2cpp, slice2java, slice2cs, slice2py, slice2php to generate specific mappings.

Ice provides IceGrid to manager Ice Servers, and because it allows multiple replicas of the same server. In this way, you can achieve load-balance across these replicas. IceGrid can manage several Ice Servers. Ice Client doesn’t know which specific Ice Server will fulfil each request at the beginning. So, IceGrid tells Ice Client to contact one specific Ice Server to finish the request. All these middleware operations are transparent to Ice clients; Ice framework helps to do all these seamless.
To provide a safer environment, Ice provides Glacier2, which works as a server Firewall. All the backend services are hidden from the public. Ice Client sends all the requests to Glacier2, and Glacier2 will distribute all of these requests to backend servers and sends back the results. Glacier2 also has the responsibility to manage sessions. Some polices can be configured in Glacier2 like restricting specific users, IPs, etc.

To balance the load or to distribute different services Ice allows for deploying multiple instances of Glacier2.

C# is the primary programing language for Unity and Ice RPC framework has several programming languages mappings including C#. But when we first introduced Ice code into Unity, there was a very big problem. The reason of this problem was that the last stable version of Unity is using Mono 2.0 to support C# and the lowest version of .Net supported by Ice is 4.0.
On 2 Feb, 2017, Unity released a very experimental version on forum Betas & Experimental Features > Experimental Scripting Previews, which begins to support .Net 4.6. Hence, in this project we adopted this version of Unity as workaround.

See the forum link below to find the information about the Unity supported .Net version https://forum.unity3d.com/threads/upgraded-mono-net-in-editor-and-some-players-on-5-6-0b5.454387/

The download link http://beta.unity3d.com/download/860e381a5579/public_download.html

To this day, it looks like that the new beta version https://unity3d.com/unity/beta/ has already support for .Net 4.6, but we could not check it properly. Make sure in Unity Editor -> Player Settings -> Configuration, the Api Compatibility Level is at least .Net 4.0 as below.

![Configuration](image)

Figure 2.18 Unity .Net version
3. Requirements Specification

This software requirement specification has been documented using the most suitable sections for this project existing in the IEEE 830 template [15].

3.1 Scope of the Project

The goal of the project is to build a network framework to help other developers who want to build their own online game using Unity.

The project will provide the fundamental functions that one online game needs; a demo to demonstrate the framework; and some guidelines to show how to use the framework.

3.2 Product Perspective

The whole project consists of a Server side (see figure 3.1) and a Client side (see figure 3.2). The Server side consists of Online Server, Scene Server and AssetBundle HTTP Server. Client side is a game Client.

For Server Side, AssetBundle HTTP Server is a simple HTTP Server, nginx or apache or any other HTTP server can be chosen.
The Online Server handles all the operations related to online services session management, data persistence, online users interactive, etc. The Online Server will be written in Java based on Ice.

The Scene Server is responsible for all the game scene related to the movement in the scene, interact with the scene environment, etc. A Scene Server will be a Unity dedicated server.

The Client is the product to the end users. After clients connect to Online Server, clients can chat with each other, gets the scene server list, restore its last state in a previous game session, etc. A Client is a Unity Client.

3.3 Hardware and OS Platform Requirements

3.3.1 Servers Hardware and OS Platform Requirements

The hardware needs required for running the server side of the project is any common PC, and if the project is used to develop any specific games, the hardware requirements will depend on each game’s needs. This project’s server side includes online server and Unity Scene Server. Both of the two server parts are cross platform. Online server is written in Java, therefore it can run on Windows, Linux and macOS. Due to Unity features, the Unity Scene Server can run on Windows, Linux, macOS and even other console platforms, mobile platforms, but for economic and performance reasons, only desktop and server OS platforms are recommended.

3.3.2 Clients Hardware and OS Platform Requirements

The hardware needs for clients are different depending on the game type, the optimization and the user experience you want to achieve.

The client is assumed to be built on Unity, which supports Desktop platforms, console platforms, mobile platforms, WebGL, etc.

3.4 Functional Requirements

<table>
<thead>
<tr>
<th>Code</th>
<th>FR1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Client-Server Architecture</td>
</tr>
<tr>
<td>Description</td>
<td>The system has a separate server system</td>
</tr>
<tr>
<td>Preconditions</td>
<td></td>
</tr>
<tr>
<td>Inputs</td>
<td></td>
</tr>
<tr>
<td>Outputs</td>
<td></td>
</tr>
<tr>
<td>Note</td>
<td>The server consists of online server and Scene Server</td>
</tr>
</tbody>
</table>

Table 3.1 FR1
<table>
<thead>
<tr>
<th>Code</th>
<th>FR2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>4 avatars to choose</td>
</tr>
<tr>
<td>Description</td>
<td>In login panel, it has 4 avatars to choose</td>
</tr>
<tr>
<td>Preconditions</td>
<td></td>
</tr>
<tr>
<td>Inputs</td>
<td></td>
</tr>
<tr>
<td>Outputs</td>
<td></td>
</tr>
<tr>
<td>Note</td>
<td>All avatars are downloaded from unity asset store free assets</td>
</tr>
</tbody>
</table>

Table 3.2 FR2

<table>
<thead>
<tr>
<th>Code</th>
<th>FR3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Online server</td>
</tr>
<tr>
<td>Description</td>
<td>Clients log in through online server system</td>
</tr>
<tr>
<td>Preconditions</td>
<td></td>
</tr>
<tr>
<td>Inputs</td>
<td>User id, password</td>
</tr>
<tr>
<td>Outputs</td>
<td>OK and user is logged in into the online server system</td>
</tr>
<tr>
<td>Note</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.3 FR3

<table>
<thead>
<tr>
<th>Code</th>
<th>FR4</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Scene Server list</td>
</tr>
<tr>
<td>Description</td>
<td>Clients get Scene Server list after login</td>
</tr>
<tr>
<td>Preconditions</td>
<td>Clients Logged into the online server</td>
</tr>
<tr>
<td>Inputs</td>
<td></td>
</tr>
<tr>
<td>Outputs</td>
<td>Scene Server list</td>
</tr>
<tr>
<td>Note</td>
<td>Scene Servers need to be started before clients</td>
</tr>
</tbody>
</table>


Table 3.4 FR4

<table>
<thead>
<tr>
<th>Code</th>
<th>FR5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Scene Server state</td>
</tr>
<tr>
<td>Description</td>
<td>The Scene Server list will show each server current state: online or offline. If the scene supported by the Scene Server is an AssetBundle scene and it is not cached in the client yet, downloadable icon will show on top of the entry and cover the Scene Server state.</td>
</tr>
</tbody>
</table>

Table 3.5 FR5

<table>
<thead>
<tr>
<th>Code</th>
<th>FR6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Chat panel</td>
</tr>
<tr>
<td>Description</td>
<td>Clients can access chat panel after the login by pressing “m” key</td>
</tr>
<tr>
<td>Preconditions</td>
<td></td>
</tr>
<tr>
<td>Inputs</td>
<td></td>
</tr>
<tr>
<td>Outputs</td>
<td>Show or hide chat panel</td>
</tr>
<tr>
<td>Note</td>
<td>Chat service is implemented by online server, so after connecting to online server, chat system will be available. Messages coming from the same user will be shown in one specific colour.</td>
</tr>
</tbody>
</table>

Table 3.6 FR6

<table>
<thead>
<tr>
<th>Code</th>
<th>FR7</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Command “/list” in chat panel</td>
</tr>
<tr>
<td>Description</td>
<td>Chat panel supports “/list” command to show current online users</td>
</tr>
<tr>
<td>Preconditions</td>
<td></td>
</tr>
<tr>
<td>Inputs</td>
<td></td>
</tr>
<tr>
<td>Outputs</td>
<td>Online user names.</td>
</tr>
<tr>
<td>Note</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.7 FR7

<table>
<thead>
<tr>
<th>Code</th>
<th>FR8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Chat message sending</td>
</tr>
<tr>
<td>Description</td>
<td>Chat panel supports “@username message” command to send a message to an specific online user</td>
</tr>
<tr>
<td>Preconditions</td>
<td></td>
</tr>
<tr>
<td>Inputs</td>
<td>message</td>
</tr>
<tr>
<td>Outputs</td>
<td>The message is shown on the target user’s panel.</td>
</tr>
<tr>
<td>Note</td>
<td>All the messages will be private therefore they will be received just by the target user.</td>
</tr>
</tbody>
</table>

Table 3.8 FR8

<table>
<thead>
<tr>
<th>Code</th>
<th>FR9</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Download AssetBundle scene</td>
</tr>
<tr>
<td>Description</td>
<td>In Scene Server list panel, if on asset bundle scene entry, downloadable icon is clicked, the downloading process will begin with progress indicator.</td>
</tr>
<tr>
<td>Preconditions</td>
<td></td>
</tr>
<tr>
<td>Inputs</td>
<td>Selected asset bundle scene</td>
</tr>
<tr>
<td>Outputs</td>
<td>AssetBundle scene becomes accessible</td>
</tr>
<tr>
<td>Note</td>
<td>AssetBundle scene will only be downloaded once and it will be cached till the game session expires.</td>
</tr>
</tbody>
</table>

Table 3.9 FR9

<table>
<thead>
<tr>
<th>Code</th>
<th>FR10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Enter Scene Server</td>
</tr>
<tr>
<td>Description</td>
<td>In Scene Server list panel, if online Scene Server entry is clicked, client will enter the Scene Server</td>
</tr>
<tr>
<td>-------------</td>
<td>-----------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Preconditions</td>
<td></td>
</tr>
<tr>
<td>Inputs</td>
<td>Selected Scene Server</td>
</tr>
<tr>
<td>Outputs</td>
<td>Client is inside the scene supported by the Scene Server.</td>
</tr>
<tr>
<td>Note</td>
<td>Only Scene Servers with status “online” are accessible.</td>
</tr>
</tbody>
</table>

Table 3.10 FR10

<table>
<thead>
<tr>
<th>Code</th>
<th>FR11</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Controller avatar in Scene Server</td>
</tr>
<tr>
<td>Description</td>
<td>User can control an avatar.</td>
</tr>
<tr>
<td>Preconditions</td>
<td>After enter an online Scene Server,</td>
</tr>
<tr>
<td>Inputs</td>
<td>User events (keyboard, mouse, etc.)</td>
</tr>
<tr>
<td>Outputs</td>
<td>Scene refresh implemented in the scene</td>
</tr>
<tr>
<td>Note</td>
<td>Avatar control is implemented in the scene, so it is out of the scope of this project.</td>
</tr>
</tbody>
</table>

Table 3.11 FR11

<table>
<thead>
<tr>
<th>Code</th>
<th>FR12</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Inventory panel</td>
</tr>
<tr>
<td>Description</td>
<td>Clients can access inventory panel by pressing “i” key</td>
</tr>
<tr>
<td>Preconditions</td>
<td>After enter a Scene Server,</td>
</tr>
<tr>
<td>Inputs</td>
<td></td>
</tr>
<tr>
<td>Outputs</td>
<td>Show or hide inventory panel</td>
</tr>
<tr>
<td>Note</td>
<td>Item icons are shown in the inventory panel</td>
</tr>
</tbody>
</table>

Table 3.12 FR12

<table>
<thead>
<tr>
<th>Code</th>
<th>FR13</th>
</tr>
</thead>
</table>

22
<table>
<thead>
<tr>
<th>Name</th>
<th>Pick up item</th>
</tr>
</thead>
<tbody>
<tr>
<td>Description</td>
<td>Clients can pick up items such as an axe, a knife, a mace. These items will show up in client’s inventory panel.</td>
</tr>
<tr>
<td>Preconditions</td>
<td>After enter an online Scene Server</td>
</tr>
<tr>
<td>Inputs</td>
<td>Selected item</td>
</tr>
<tr>
<td>Outputs</td>
<td>New item into the inventory</td>
</tr>
<tr>
<td>Note</td>
<td>The items in the scene are selected by move the mouse curse and click left mouse key.</td>
</tr>
</tbody>
</table>

Table 3.13 FR13

<table>
<thead>
<tr>
<th>Code</th>
<th>FR14</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Drop items</td>
</tr>
<tr>
<td>Description</td>
<td>By right clicking the item icon, the item will be removed from the inventory and dropped on the nearby ground</td>
</tr>
<tr>
<td>Preconditions</td>
<td>With inventory panel open</td>
</tr>
<tr>
<td>Inputs</td>
<td>Selected item</td>
</tr>
<tr>
<td>Outputs</td>
<td>Item is removed from inventory</td>
</tr>
<tr>
<td>Note</td>
<td>The items picked up from a scene could be taken and dropped on another scene supported by another Scene Server.</td>
</tr>
</tbody>
</table>

Table 3.14 FR14

<table>
<thead>
<tr>
<th>Code</th>
<th>FR15</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Jump to another scene supported by another Scene Server</td>
</tr>
<tr>
<td>Description</td>
<td>When the avatar walks to the rotating cylinder with the scene label (developers can choose other object and make the label name match other scene server name), if the Scene Server that supports that scene is online otherwise if it is not online an error message “Server is not found” will show up, the avatar will leave the current scene and enter the new scene.</td>
</tr>
<tr>
<td>Preconditions</td>
<td></td>
</tr>
</tbody>
</table>
### Table 3.15 FR15

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Selected gate to a new scene</th>
</tr>
</thead>
<tbody>
<tr>
<td>Outputs</td>
<td>Avatar shows up in the new scene</td>
</tr>
<tr>
<td>Note</td>
<td>If the target Scene Server scene is an AssetBundle scene and hasn’t been cached yet, the client will begin downloading it and will jump to the Scene Server once downloading is finished. Jump to another scene supported by the same Scene Server is implemented in each game, therefore it is out of the scope of this project. When the user jumps to another scene, he/she carries his/her inventory to the new scene.</td>
</tr>
</tbody>
</table>

### Table 3.16 FR16

<table>
<thead>
<tr>
<th>Code</th>
<th>FR16</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Exit panel</td>
</tr>
<tr>
<td>Description</td>
<td>In the Scene Server, by pressing “esc” key, an exit panel will show up</td>
</tr>
<tr>
<td>Inputs</td>
<td></td>
</tr>
<tr>
<td>Outputs</td>
<td>The exit panel shows “Save and Exit”, “Exit without Save”, “Back to Login”</td>
</tr>
<tr>
<td>Note</td>
<td></td>
</tr>
</tbody>
</table>

### Table 3.17 FR17

<table>
<thead>
<tr>
<th>Code</th>
<th>FR17</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Save and Exit game</td>
</tr>
<tr>
<td>Description</td>
<td>If “Save and Exit” is clicked, current avatar’s state will be saved and the game will be over.</td>
</tr>
<tr>
<td>Preconditions</td>
<td>In exit panel</td>
</tr>
<tr>
<td>Inputs</td>
<td></td>
</tr>
<tr>
<td>Outputs</td>
<td>Avatar’s state is saved till next game session</td>
</tr>
<tr>
<td>Note</td>
<td>Avatar’s state may consist of its position and rotation in the scene, etc. However, the developer of the game will have freedom to decide what will be saved.</td>
</tr>
<tr>
<td>Code</td>
<td>FR18</td>
</tr>
<tr>
<td>-------</td>
<td>------</td>
</tr>
<tr>
<td>Name</td>
<td>Back to Login panel</td>
</tr>
<tr>
<td>Description</td>
<td>If “Back to Login” is clicked, the client will be disconnected from the current Scene Server and will be sent back to login panel.</td>
</tr>
<tr>
<td>Preconditions</td>
<td>In exit panel</td>
</tr>
<tr>
<td>Inputs</td>
<td></td>
</tr>
<tr>
<td>Outputs</td>
<td>Game session is over and login panel is shown</td>
</tr>
<tr>
<td>Note</td>
<td>This will not save avatar’s state, bring the client back to login panel</td>
</tr>
</tbody>
</table>

Table 3.18 FR18

<table>
<thead>
<tr>
<th>Code</th>
<th>FR19</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Exit without Save</td>
</tr>
<tr>
<td>Description</td>
<td>If “Exit without Save” is clicked, current avatar’s state will not be saved and the game will be over.</td>
</tr>
<tr>
<td>Preconditions</td>
<td>In exit panel</td>
</tr>
<tr>
<td>Inputs</td>
<td></td>
</tr>
<tr>
<td>Outputs</td>
<td></td>
</tr>
<tr>
<td>Note</td>
<td>This will not save avatar’s state, and will close the client.</td>
</tr>
</tbody>
</table>

Table 3.19 FR19

<table>
<thead>
<tr>
<th>Code</th>
<th>FR20</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Restore State</td>
</tr>
<tr>
<td>Description</td>
<td>If “RestoreState” is clicked, the client will connect to the last saved scene with the same avatar’s position and rotation as they were when the user left that scene.</td>
</tr>
<tr>
<td>Preconditions</td>
<td>Login to online server,</td>
</tr>
<tr>
<td>Inputs</td>
<td></td>
</tr>
<tr>
<td>Outputs</td>
<td>Avatar shows up in the saved scene.</td>
</tr>
<tr>
<td>Note</td>
<td>This will not save avatar’s state if exit.</td>
</tr>
</tbody>
</table>

Table 3.20 FR20
### 3.5 Non-Functional Requirements

<table>
<thead>
<tr>
<th>Code</th>
<th>NFR1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Deploy servers in different machines</td>
</tr>
<tr>
<td>Description</td>
<td>Online server and Scene Servers should be able to run on the same or on different machines</td>
</tr>
<tr>
<td>Inputs</td>
<td></td>
</tr>
<tr>
<td>Outputs</td>
<td></td>
</tr>
<tr>
<td>Note</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.21 NFR1

<table>
<thead>
<tr>
<th>Code</th>
<th>NFR2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Customize online server’s address</td>
</tr>
<tr>
<td>Description</td>
<td>Client should be able to change the online server’s destination address in login panel</td>
</tr>
<tr>
<td>Inputs</td>
<td>New address</td>
</tr>
<tr>
<td>Outputs</td>
<td></td>
</tr>
<tr>
<td>Note</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.22 NFR2

<table>
<thead>
<tr>
<th>Code</th>
<th>NFR3</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Adding a new Scene Server should not require any change in the online server’s code.</td>
</tr>
<tr>
<td>Description</td>
<td></td>
</tr>
<tr>
<td>Inputs</td>
<td></td>
</tr>
<tr>
<td>Outputs</td>
<td></td>
</tr>
<tr>
<td>Note</td>
<td></td>
</tr>
</tbody>
</table>

Table 3.23 NFR3
4. Development of the Solution

4.1 Architecture of the Project

The project consists of a client side and a server side. The server side contains online server, scene servers and AssetBundle HTTP server. The scene servers are powered by Unity that will communicate with the online server using Ice. The AssetBundle HTTP server is a standalone http server that is used to distribute AssetBundle Assets. A Scene server is responsible for in-game logic, that is, avatar control, item operations, scene objects’ behavior, etc. Online server handles the log in, game server’s management, account management, chatting management, inventory persistence, avatar state persistence, etc.

Client side is a game client, which is powered by Unity. A user will need to install the game client into his/her machine to join to the network game.

If a user wants to join to the network game, first the client needs to connect to the online server. The communication between the client and the online server is achieved through Ice. In Figure 4.1, a game client knows the glacier2router’s address. All clients’ requests are sent to glacier2router. Glacier2router will dispatch all received requests to the backend icegridnode. All the online server code resides in onlineserver.jar, which is loaded by icegridnode.

After logging into the online server, the game client can see scene server list. If the scene server is running a AssetBundle scene that the client hasn’t downloaded yet, the client can download the assets from AssetBundle HTTP Server by clicking the downloadable icon showed on the scene server list.

![Figure 4.1 Deployment Diagram](image-url)
4.2 Online and DataBase Server Design

4.2.1 DataBase Schema and Persistence Design

There are two tables related to user account, one is user_accounts_tbl to store account data such as nickname, email, creation time, level, etc. Another separate table, auth_tbl, is used to check user’s credentials. The benefit of this separation in two tables is to make it easier to adapt to multiple auth strategies, like Facebook, google, email, or customize auth strategy.

The avatars_tbl is used to store data related avatar. The column avatar_state is declared as binary type, and it stores serialized data of avatar’s state such as which the last visited scene was, and which its position and rotation were, etc.
The inventory_items_tbl is used to store items data. The column item_state is declared as binary type and it is used to store item state.

All the data tables above are mapped with Java Classes (see figure 4.5) by using Hibernate.
Hibernate is used to easily handle database information through objects. Each database table is mapped to an entity that is operated as an object in online server. For example, figure 4.6 shows how InventoryEntity class is mapped to table inventory_items_tbl. The operations related to the database are handled by using the Hibernate Query Language and Criteria Queries.

4.2.2 Online Services
Services package classes AccountServiceI, AvatarServiceI and NotificationRegisterServiceI implement from slice interfaces AccountService, AvatarService and NotificationRegisterService respectively. These classes offer services for game clients. All the functions are implemented in online server. The game client has the same interfaces to call remote methods of the online server objects.

Figure 4.8 Service Account, Avatar interface

AccountServiceI handles account related operations: CreateAccount, GetAccountByName. AvatarServiceI handles avatar related operations. FR20 “Restore State” (see table 3.20) uses getAvatar to get the data to restore avatar’s state. FR17 “Exit and Save” (see table 3.17) uses updateAvatar to save avatar’s state.

NotificationRegisterServiceI handles messages that are invoked by Online Server instead of Client like Chat process. First a Client invokes “SendChat” service, when Online Server receives this, Online Server will find this “SendChat” message’s receiver and invokes “OnSendChat” service.

Figure 4.9 scene server initial service interface
Scene Server related operations are in DsServiceI and ServerServiceI. As starting up, each Scene Server will register itself into online server providing its host address and its running server scene.

![ChatService, InventoryService interface](image)

Figure 4.10 ChatService, InventoryService interface

ChatService is responsible for replaying chat message. InventoryService is responsible for inventory operations. All services classes implement those corresponding slice interfaces, and located in service package.

![Scene Server related interface](image)

Figure 4.11 Scene Server related interface

UserAccountDB is responsible for UserAccount related database operations. AuthDB is for Auth related database operations, AvatarDB is for avatar related database operations, InvenotryDB is for inventory related database operations. All the database related operations are located in db package.
4.2.3 Slice Definition of Online Services

As describe in 2.5.4 Ice Overview, Ice client and Ice server communicate to each other by sharing the same data structure and interfaces definitions specified with Slice language (Specification Language for Ice).

```c
struct ServerEntry
{
    string serverName;
    string servAddr;
    string name;
};
sequence<ServerEntry> serverInfoSeqT;
struct Position
{
    float x;
    float y;
    float z;
};
struct Rotation
{
    float x;
    float y;
    float z;
    float w;
};
struct AvatarState
{
    Position position;
    Rotation rotation;
    ServerEntry serverEntry;
};
struct ServerInfo
{
    serverInfoSeqT serverInfoSeq;  // hasLastState = true;
};
```

Figure 4.12 Some data structures defined in TypeDef.ice

All the data structures that will be shared by several services are defined in TypeDef.ice file. All the interfaces are defined in separated ServiceNameService.ice files.

```c
#ifndef AVATAR_SERVICE_H
#define AVATAR_SERVICE_H

#include &lt;TypeDef.ice&gt;

module es
{
    module upm
    {
        module fjl
        {
            module rmi
            {
                interface AvatarService
                {
                    AvatarProfile getAvatar(string name);
                    void updateAvatar(AvatarProfile avatarProfile);
                }
            }
        }
    }
};
#endif
```

Figure 4.13 AvatarService.ice
For example, all the operations related to Avatar resident in AvaterService.ice (see figure 4.14) and the same applies for other services (see figure 4.15) such as ChatService, AccountService, etc.

![Figure 4.14 All slice files](image)

### 4.3 Scene Server Design

#### 4.3.1 Unity Scene Server

In this project Unity Scene Server and game client share most of the code and assets. However, Scene Server has the “Server” scene as the start scene, whereas game client has the “Client” scene as the start scene. As building the server in the Unity Editor, the Server scene must be the first scene.

![Figure 4.15 scene server as initial scene](image)

Scene Server communicates with Online Server using Ice, and communicates with Game Client using Unity Networking. Game Client communicates with Online Server using Ice.
For Client and Scene Server communication, all the connection procedures code is located in GameClient.cs and GameServer.cs.

```csharp
void ParseParameters()
{
    onlineIP = Util.GetArg("-OnlineIP");
    onlinePort = Util.GetArg("-OnlinePort", 0);
    sceneName = Util.GetArg("-SceneName");
    sceneLabelName = Util.GetArg("-SceneLabelName");
    servIP = Util.GetArg("-IP");
    servPort = Util.GetArg("-Port", 0);

    if (null == onlineIP || 0 == onlinePort
        || null == sceneName || null == sceneLabelName || null == servIP || 0 == servPort)
    {
        Debug.LogError("ParseParameters Failed. Check the parameters!!!");
    }
}
```

A unique machine can run several Scene Servers. To make them work correctly, each Scene Server needs to specify a different available port and other parameters to initialize Scene Server (see figure 4.18).

Those running Scene Servers are built in the same way with Unity Editor. To make them load different scenes, we use the following commands.
Scene Server first connects itself to Online Server to register itself and then waits for client’s connection. Game Client connects to Online Server first and call GetServerInfo method of Online Server. Next, Game Client connects to the selected Scene Server and login into it with name and avatar info. If Scene Server accepts client’s login, returns the scene name that the client has to load. After the Game Client has loaded the same game scene, Scene Server will add the player avatar to the scene of the client by calling method ServerAddPlayer of the Game Client.

Scene Server, Game Client and Online Server share the same Slice definitions for the Java classes defined in Online Server. For Scene Server and Game Client the corresponding C# code for these Slice definitions is generated by using slice2cs utility, which is provided by Ice. In this way, Online Server methods can be invoked from C# classes in Scene Server and Game Client.

4.3.2 AssetBundle Server Scene

To make AssetBundle scene work, firstly it is necessary to assign an AssetBundle asset label to the scene (see figure 4.20).
Secondly, it is necessary to build the AssetBundle scene. To make it easier, a C# script is introduced to handle this operation (see section 4.3.3 Utility Tools).

After AssetBundle scene is built, for the server side, an AssetBundle HTTP server is set up for client access including the files of the AssetBundle scene (see figure 4.25).

In the client side, AssetBundle scenes have to be downloaded before the client can use them. The details of this process are described in section 4.4 Game Client Design.

4.3.3 Utility Tools
To make the build of AssetBundle scenes easier, a C# script “AssetBundlesBuilder.cs” is used to build AssetBundle scene.
To add the custom build option in the build menu of the Unity Editor, only a few lines of code are enough (see figure 4.24).

```csharp
[MenuItem("Build/AssetBundlesBuilderOSX AssetBundles(OSX)", false, 121)]
static void BuildAllAssetBundles()
{
    BuildPipeline.BuildAssetBundles("AssetBundles/OSX", BuildAssetBundleOptions.None, BuildTarget.StandaloneOSXUniversal);
}
```

Figure 4.24 AssetBundle builder snippet

4.4 Game Client Design

4.4.1 Unity Game Client

Game Client starts with the scene “Client”. The code for establishing the connection between the Game Client and Online Server resides in GameClient object. As building a client, the Client scene must be the first scene.
Figure 4.25 Client initial scene

GameClient object is the entry point for connecting to Online Server and Scene Server.

4.4.2 Client Operations related to Online Server

Figure 4.26 client online related managers

All the online operations reside in Online Package. Each manager will call the methods of the corresponding service that is declared in Slice definitions and implemented in the Online Server. AvatarManager is for avatar related operations like persistent avatar’s state. ChatManager is for chat operations like send chat message. ClientNotification is for messages that client will act as a passive receiver, and online server will act as an active sender like a client receives chat message from online server. DsManager is only for scene server, because a scene server also communicates with online server using Ice, all online operations that a scene server needs will be here like register itself to online server. InventoryManager is for inventory related operations like get inventory. OnlineManager is for connection management to online server. ServerListMgr is for scene server list operations.
Figure 4.27 depicts a collaboration diagram that describes how Game Client interacts with Online Server. The other online operations related to the management of the Inventory or the Avatar are developed likewise.

4.4.3 Client Game Operations

![Unity Project assets layout](image)

Figure 4.28 Unity Project assets layout
All the game objects are used as prefabs to gain the benefit that we can reuse them and the changes will apply to all the objects that made from the same prefab.

![UI related classes](image)

All the UI related code resides in UI folder and UI related code combines with Unity prefabs to work correctly. ChatUI works for gathering chat message from chat panel, and chat commands parse and process. DragPanel can make the inventory panel draggable. EscPanelAction has the methods to work for each button in EsaPanel. InventoryUI is for update inventory panel. ItemSlot is the background of the item icon in inventory panel. Login is for login process. ServEntryButton is for each scene server entry that in the server list panel. UIController is for the control of chat panel state.

### 4.5 Game Client and Scene Server Interaction

In Unity network system, each network object has its own NetworkIdentity component (see figure 4.30) and each game client has a special game object – player object that has local authority (see figure 4.30) for controlling movement.

![Network Identity (Script)](image)

Unity synchronizes network objects positions by using the NetworkTransform component (see figure 4.31)
For all those network objects, the scene server spawns them (see figure 4.32) to make them available to all clients.

```
class MySpawner : NetworkBehaviour
{
    public GameObject treePrefab;

    public void Spawn()
    {
        GameObject tree = (GameObject)Instantiate(treePrefab, transform.position, transform.rotation);
        tree.GetComponent<Tree>().numLeaves = Random.Range(10, 200);
        NetworkServer.Spawn(tree);
    }
}
```

To let Unity know all the spawnable objects, all the prefabs must be registered (see figure 4.33) into a NetworkManager class or any of its subclasses. In this project, these subclasses are, for client, the GameClient class, and for scene server, the GameServer class.
As described in chapter 2.5.2 Unity Network, Unity Network has low level network message system and High Level API: Commands - which are called from the client and run on the server; and ClientRpc calls - which are called on the server and run on clients. For GameClient and Scene Server interactions using Commands and ClientRpc calls, arguments will automatically be passed (see figure 4.34).

```csharp
class Player : NetworkBehaviour
{
    public GameObject bulletPrefab;

    [Command]
    void CmdDoFire(float lifeTime)
    {
        GameObject bullet = (GameObject)Instantiate(
            bulletPrefab,
            transform.position + transform.right,
            Quaternion.identity);

        var bullet2D = bullet.GetComponent<Rigidbody2D>();
        bullet2D.velocity = transform.right * bulletSpeed;
        Destroy(bullet, lifeTime);

        NetworkServer.Spawn(bullet);
    }

    void Update()
    {
        if (!isLocalPlayer)
            return;

        if (Input.GetKeyDown(KeyCode.Space))
        {
            CmdDoFire(3.0f);
        }
    }
}
```

Since Unity 5.2, not only local player object can have client authority, but also other network object could have client authority. All those network objects that have client authority can send
Commands. For pickup item operation, the player call command “CmdPickItem” (see figure 4.35), then the code will be executed on the scene server within “CmdPickItem”, the scene server call ClientRpc “item.RpcIconize” (see figure 4.36) to let all clients iconize the item.

Figure 4.35 CmdPickItem

Figure 4.36 RpcIconize

For those attributes of network objects that need to be synchronized use “SyncVar” annotation (see figure 4.37) and Unity will automatically do the synchronization for all the clients.

Figure 4.37 SyncVar [17]
5. Testing

5.1 Runtime Environment Preparation

To test the demo project, the server side should be ready at first. The first step is to import the SQL schema to MariaDB. The AssetBundle scene HTTP server is a normal web server. In this project, nginx is used to host the resources.

The next step is to start Online Server. Online Server consists of IceGrid and glacier2router, to make it easy to start these. The script files are in tools folder, bash files for macOS and batch files for Windows. Start IceGrid first using start.icegridnode.sh in macOS or start.icegridnode.bat in Windows followed by start.icegridadmin.sh or start.icegridadmin.bat.

Then start glacier2router using start.gate.sh or start.gate.bat.

After the Online Server is ready, Scene Servers can be launched. In Windows platform, the batch files could help us to do that. But in macOS platform the command needs to be copied from bash files to console to run the command because the “open” command cannot correctly start the Scene Server when it is in a bash file.

After the server side (Online Server, AssetBundle scene HTTP server and Scene Servers) is ready, clients can be launched.

5.2 Demo Project

To test the framework, we decided to develop a demo project, which is a simple game that allowed us to test the main operations of the framework in an interactive way. The demo client is tested completely on Windows 10 and macOS 10.12 and partially on a Nexus 5 mobile.

All the assets of the demo project are downloaded from Unity Asset Store. The details of the assets and links are in appendix chapter. The demo project consists of 3 game scenes (City, GrassHouse and RoomInside) with 4 avatars. As a sample of the different tests we carried out with the demo project, we will explain the following process.

client log in -> see server list -> download AssetBundle scene -> choose one online Scene Server to connect -> move around in the game scene -> chat with another online user -> show inventory -> pick up items -> jump to another Scene Server -> drop item from the inventory -> save and exit game -> log in -> restore last state.

Next, we will show a sequence of screenshots to illustrate the process outlined above.
The screenshot in figure 5.1 is the login panel. To test the demo project, the default online server address is 127.0.0.1, but it can be changed by typing another address. After entering username and password, you can login in by clicking login button.

Figure 5.2 Login and Avatar Chosen Panel
The login panel also shows an avatar choose area, a different avatar can be chosen by clicking the arrows buttons (see figure 5.2). In the demo project, there are 4 avatars available.

![Figure 5.3 Server List Scene](image)

After signing in, the server list panel is shown (see figure 5.3). Each server entry will appear with the state indicator to show online or offline. The right corner “CleanCache” is to clean the cached AssetBundle scene. The “RestoreButton” is to restore the avatar to the last saved state, which means to set the avatar in the last visited scene with its last position and rotation before leaving the game.

![Figure 5.4 Server List Scene with Chat Panel](image)
The chat system is supported by online server, so after signing in, the chat panel is available on the left side of the screen (see figure 5.4). To write somebody a message uses this format: @somebody message. To know who is online, use the command:/list.

Figure 5.5 Server List Panel with AssetBundle Scene Not Cached

When the client finds that there is AssetBundle game scene that has not been downloaded yet in the client, a downloadable icon will be shown on the entry of this scene (see figure 5.5). By clicking the downloadable icon, the AssetBundle scene will be downloaded from AssetBundle scene http server and the progress of the download process will be shown in the server entry.

Figure 5.6 Server List Panel with AssetBundle Scene Downloading
In the game scene, the inventory panel will show up by pressing “i”. If we pick up the item in the game scene, the item will be moved to the avatar’s inventory. Also by clicking the item in the inventory panel, the item will be moved back to the game scene. Additionally, when the avatar jumps to other scene, the avatar will carry the items in its inventory with it to this game scene.

Rotating cylinders with labels (see figure 5.8) represent gates to other scenes of the game (supported by other Scene Servers). When the avatar touches a cylinder, the avatar will leave the current scene and will teleport to another scene (supported by other Scene Server).
When teleporting from the city scene to another scene, the avatar will show up near the city cylinder (see figure 5.9).

Although this example does not show more than one avatar in the virtual world, multiuser operation has been tested also with the demo project.
6. Developer Guide

6.1 Development Environment Preparation

6.1.1 Online Server Environment Preparation

The project is using IntelliJ IDEA as online server project IDE, and Gradle as build system. All dependencies in build.gradle will be handled by Gradle automatically. It is necessary to make sure during the Gradle build process that the Internet is accessible.

```
dependencies {
    compile 'com.zeroc:ice:3.6.3'
    compile 'com.zeroc:glacier2:3.6.3'
    compile group: 'org.apache.logging.log4j', name: 'log4j-api', version: '2.7'
    compile group: 'org.apache.logging.log4j', name: 'log4j-core', version: '2.7'
    compile group: 'mysql', name: 'mysql-connector-java', version: '5.1.6'
    compile 'org.hibernate:hibernate-core:3.6.7.Final'
    compile "org.slf4j:slf4j-simple:1.7.9"
    compile "javassist:javassist:3.12.1.GA"
    compile group: 'commons-codec', name: 'commons-codec', version: '1.5'
}
```

Figure 6.1 dependencies in build.gradle

Ice 3.6.3 is needed, follow [https://zeroc.com/distributions/ice](https://zeroc.com/distributions/ice) instructions to install Ice. After the installation, make sure that glaricer2router, icegridnode, slice2cs, slice2java are available.

![Figure 6.2 online server config files](image)

Check the configuration files in config folder, make sure the port number 4060 is available through firewall if the online server is going to accept connections from other computers. This port number is changeable in gate.cfg.
6.1.2 Client and Scene Server Environment Preparation

The project relays on both Unity and Ice, it’s very important that Unity supports that same or higher .Net version than Ice.

Ice 3.6.3 C# is based on .Net 4.0 and on 2 Feb, 2017 Unity released the version that supports .Net 4.6.

See the forum link below to find the information about the Unity supported .Net version
https://forum.unity3d.com/threads/upgraded-mono-net-in-editor-and-some-players-on-5-6-0b5.454387/

The download link http://beta.unity3d.com/download/860e381a5579/public_download.html

Currently, it looks like the new beta version also works with .Net 4.6. However, the code of this project has not checked with this new version.

6.2 Communication Data Layers

The project involves Online Server and Scene Server. When the client communicates with Online Server, the data flow will go through the layers depicted in figure 6.3.

![Figure 6.3 client and Online Server data flow](image)
On the other hand, when the client communicates with Scene Server, depending on which phase of the game is running, the data flow will be different.

Before the network client object is spawned, only low level message exchange is available.

```csharp
void RegisterClientHandlers()
{
    client.RegisterHandler(CustomMsgType.SC_Login, SC_Login);
    client.RegisterHandler(CustomMsgType.SC_PlayerReady, SC_PlayerReady);
}
```

Figure 6.4 register message handle methods

To be able to use Unity Network Message, the message handle methods need to be registered with message types (see figure 6.4).

```csharp
// called when connected to a server
public override void OnClientConnect(NetworkConnection conn)
{
    CSLoginMessage csLoginMessage = new CSLoginMessage();
    csLoginMessage.userName = userName;
    csLoginMessage.avatarId = avatarPanel.AvatarCursor;
    csLoginMessage.cookie = cookie;
    if (restoreFlag == true)
    {
        csLoginMessage.restoreFlag = true;
        restoreFlag = false;
    }
    client.Send(CustomMsgType.CS_Login, csLoginMessage);
    Debug.Log(string.Format("OnClientConnect conn.ToString():{0}",
```

Figure 6.5 Send Unity Network message snippet

To send a message to the connected server, fill in the fields of the defined message structure and send the message with the corresponding message type (see figure 6.5).

However, after the client is spawned by Scene Server, for communication between client’s player and scene server it is more convenient to use Commands and ClientRpc. See chapter 2.5.2 Unity Network.
Some operations that involve both Scene Server and online server will have a mix layers.

6.3 Begin to Implement New Online Functions

Client and Online server communicate with each other through Ice. The first step is to write new Slice definitions.

```java
interface AvatarService {
    AvatarProfile getAvatar(String name);
    void updateAvatar(AvatarProfile avatarProfile);
}
```

Figure 6.7 AvatarService Slice

The Figure 6.7 is an example of how to write Slice definition. The Slice has some built-in types like int, string, byte, sequence …, other compounded types like AvatarProfile that developers need to define by themselves (see figure 6.8).
After the Slice definitions are done, use Gradle task “compileSlice” to generate the corresponding java code that will reside in build/generated-src folder (see figure 6.9).

The implement of the interfaces will reside in es.upm.fi/services folder (see figure 6.10).
The Ice runtime will dispatch the method and encapsulated the parameters here, developers only need to use the data do the needed operations and fill the return data structure.

After the above process, the new services need to register to Ice runtime so Ice can know how to dispatch, to do this a small piece of code is needed to add to the following two places: Server.java (see figure 6.12) and application.xml (see figure 6.13).
Because the online server code we write is not a standalone application, but services which will be loaded by icegridnode. To make it easier, in the folder tools (see figure 6.14) there are several script to do that. All .sh files are for MacOS and .bat files are for Windows. All the folder pathes are relative to the structure of the project, if the structure is not changed there’s no need to change these scripts.

First execute start.icegridnode and start.icegridadmin which will launch icegridnode and load the services jar we built. Then execute start.gate which will launch glacier2router to handle session management.

6.4 Begin to Implement New In-Game Logic

All the in-game development follows Unity workflow except those related to online server. For client and scene server to use Ice, C# generated code from Slice definition is needed (see figure 6.15).
To generate the C# code, a script slice2cs.sh resides in Server/Tools folder (see figure 6.16).

![Server Tools folder](image)

Figure 6.16 Server Tools folder

After getting the generated C# code, both client and scene server can use the interfaces defined by Slice. Figure 6.17 is the example of ChatManager to use ChatService.

```csharp
using es.upm.fi.rmi;

// ChatManager exists between game logic and database by interact with OnlineService using Ice.
// Once client connected to OnlineService, chat service becomes available.
public class ChatManager
{
    private ChatServicePrx chatPrx;
    private static ChatManager instance = null;
    private Queue<Message> msgQue;
    public static ChatManager GetChatManager()
    {
        if(null == instance)
            instance = new ChatManager(OnlineManager.GetOnlineManager());
        return instance;
    }
    private ChatManager(OnlineManager olManager)
    {
        chatPrx = ChatServicePrxHelper.checkedCast(olManager.StringToProxy("PaseoOnline/PaseoChatService"));
        msgQue = new Queue<Message>();
    }
    public void SendChatMessage(string receiverName, string message)
    {
        chatPrx.SendMessage(0, 0, 1, receiverName, message);
    }
}
```

Figure 6.17 ChatManager.cs

Other managers like AvatarManager, InventoryManager follow the same workflow.
7. Conclusions and Future Work

7.1 Conclusion

The Online server extends Unity low level network operations with network game functions such as account management, chat system, server list management, avatar state saves and restore function, etc. In addition, Online server also supports avatar teleport to another scene function, which will permit to divide one big virtual world into several small scenes to provide better performance. On the other hand, the game logic is still relying on Unity, which means that it will follow all the instructions from Unity.

Thanks to the Unity AssetBundle feature, we can achieve that client’s first version could be very small and the needed more scenes could be downloaded on demand later.

With the developed demo project, which has been tested on both Windows 10 and macOS, the main functionalities of the framework have been tested.

7.2 Future Work

The current work is only a demo project. It does not provide a good availability, because it doesn't keep an increasing user volume in mind. However, Ice provides some services that would make possible scale the system. For example, through IceGrid we would be able to use multiple IceNodes to distribute the workload among them.

On the other hand, Ice glacier2 handles the session management, therefore the easiest way to improve its load capacity would be by setting up several Ice Glacier2 instances, so that the client may use random or round-robin mechanism to choose one.

Next, we will describe briefly some possible extensions of the framework expressed as future requirements.

<table>
<thead>
<tr>
<th>Code</th>
<th>FVR1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Scene Server restore from crash</td>
</tr>
<tr>
<td>Description</td>
<td>When a Scene Server crashes, it should be possible to restore its last state</td>
</tr>
<tr>
<td>Inputs</td>
<td>Scene server stops accidently with clients in the scene</td>
</tr>
<tr>
<td>Outputs</td>
<td>Scene server restore its last state</td>
</tr>
</tbody>
</table>

Table 7.1 FVR1
<table>
<thead>
<tr>
<th>Code</th>
<th>FVR2</th>
<th>Name</th>
<th>More than one Scene Server supporting the same scene</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>To achieve scene server’s load balance, there will be multiple scene server</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>running the same scene</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Table 7.2 FVR 2</td>
</tr>
<tr>
<td>Code</td>
<td>FVR3</td>
<td>Name</td>
<td>More than one gate to improve reliability</td>
<td>Description</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>To deploy more than one glacier2router to get a better reliability</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Table 7.3 FVR 3</td>
</tr>
<tr>
<td>Code</td>
<td>FVR4</td>
<td>Name</td>
<td>More than one online server to scale up the system.</td>
<td>Description</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>To store online data in redis or shared memory to be accessible to multiple</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>online servers.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Table 7.4 FVR 4</td>
</tr>
</tbody>
</table>
8. References


9. Appendix

9.1 Asset Store Resource Used in the Project

Simple Home Stuff (3D Models)
https://www.assetstore.unity3d.com/en/#!/content/69129

Grass Road Race (3D Models/Environments/Race Tracks & Roadways)
https://www.assetstore.unity3d.com/en/#!/content/46974

Horse Statue (3D Models/Environments/Fantasy)
https://www.assetstore.unity3d.com/en/#!/content/52025

GAZ Street Props (3D Models/Props/Exterior)
https://www.assetstore.unity3d.com/en/#!/content/57285

Barbarian warrior (3D Models/Characters/Humanoids)
https://www.assetstore.unity3d.com/en/#!/content/75519

Strong Knight (3D Models/Characters/Humanoids/Fantasy)
https://www.assetstore.unity3d.com/en/#!/content/83586

Tileable Pack 01(Textures & Materials/Ground)
https://www.assetstore.unity3d.com/en/#!/content/49278

Medieval House (3D Models/Environments/Fantasy)
https://www.assetstore.unity3d.com/en/#!/content/24040

Town Houses Pack (3D Models/Environments/Urban)
https://www.assetstore.unity3d.com/en/#!/content/42717

Space Robot Kyle (3D Models/Characters/Robots)
https://www.assetstore.unity3d.com/en/#!/content/4696

Simple Home Stuff (3D Models)
https://www.assetstore.unity3d.com/en/#!/content/69129

9.2 Tools, Frameworks, Libraries Used in the Project

IntelliJ IDEA
https://www.jetbrains.com/idea/
Gradle
https://gradle.org/

Hibernate
http://hibernate.org/

MariaDB
https://mariadb.com/

Apache Log4j 2
https://logging.apache.org/log4j/2.x/

Ice
https://zeroc.com/products/ice

Nginx
https://nginx.org/

Unity3D
https://unity3d.com/