NTNU

Norwegian University of
Science and Technology

Hybrid Peer-to-Peer Solution for MMORPGs

Frode Voll Aasen
Tom-Christian Bjørlo Johannessen

Master of Science in Communication Technology
Submission date: June 2009
Supervisor: Yuming Jiang, ITEM
Co-supervisor: Jan Erik Wold, Ablemagic AS

Norwegian University of Science and Technology
Department of Telematics

Problem Description

This thesis will examine the challenges of developing massively multiplayer online games. It is divided into a theoretical and a practical part.

The theoretical part will examine and discuss the various issues and difficulties in designing and implementing MMO games, as well at the state of the art solutions. The focus will be on the following areas:

- Performance
- Scalability
- Flexibility
- Functionality
- Distribution / Load

Using the theoretical analysis, the practical part will design and implement MMORPG network functionality for Ablemagic's game "'tisu". This task will consist of defining requirements, implementation design and software implementation. The end product should be a demo showing basic MMORPG functionality such as player movement and world state updates.

Assignment given: 14. January 2009
Supervisor: Yuming Jiang, ITEM

Preface

This thesis was carried out at the Department of Telematics at the Norwegian University of Science and Technology (NTNU). It was written at the request of Ablemagic AS, as research for the development of the networking part of their upcoming game 'tisu.

We would like to thank our supervisor, Professor Yuming Jiang for his guidance and advice. Also we would like to thank Jan Erik Wold and Nina Fjelnset at Ablemagic for proposing a thesis within an exciting area of research, and for great cooperation during our work.

Abstract

This thesis provides an introduction to the MMORPG genre, and the challenges of engineering a networking system supporting these games, as well as the state of the art titles that exist on the market today. Further it describes the design and development of a peer-to-peer networking system to support MMORPG games, and basic testing of this system.

It focuses on a broad theoretical approach to provide a solid background to understand the options available and choices made in the design of the system. The thesis presents a hybrid peer-to-peer concept that aims to reduce costs of operating an MMORPG, allowing smaller game developers to compete against major titles. It includes the main features that should be enabled in a distributed MMORPG architecture.

The concept is designed to be flexible in terms of further development, allowing new features to be added with ease and providing game designers with multiple options. A proof-of-concept demo is implemented in Java, displaying the features through a simple interface, and tests showed that the concept has potential to challenge the client-server solutions that are dominating the market today.

List of Figures

List of Tables

Abbrevations and Definitions

DEC - Digital Equipment Corporation
GUI - Graphical User Interface
LAN - Local Area Network
WAN - Wide Area Network
MMO - Massively Multiplayer Online.
MMOG - Massively Multiplayer Online Game
MMORPG - Massively Multiplayer Online Role Playing Game
MMG - Massively Multiplayer Game
Avatar [31] - An avatar is a computer user's representation of himself/herself or alter ego.
AOI - Area Of Interest
P2P - Peer-to-Peer
DoS - Denial of Service
DDoS - Distributed Denial of Service
SDL - Specification and Description Language
XML - eXtensible Markup Language
FIFO - First In First Out
UDP - User Datagram Protocol
TCP - Transmission Control Protocol
RPC - Remote Procedure Call
SDK - Software Development Kit
AoC - Age of Conan
API - Application Programming Interface

Contents

Chapter 1

Introduction

1.1 Introduction

One of the starting points for the computer game era was when Steve Russel and his team developed Spacewar. The game was only functioning on a PDP-1, an early DEC interactive mini computer which used a cathode-ray tube type display and keyboard input [4]. Later, in the 1970s, packet-based computer networking technology began to mature, allowing the creation of LANs and WANs. With network services available for commercial use, the computer game industry was quick to exploit the opportunities. The network technology led to a rapid growth of the popularity of computer games, because of the possibility to establish connections with other hosts and set up multiplayer games. One of the most important multiplayer games were MUD in 1978, which had significant input into the development of concepts of shared world design, having formative impact on the evolution of MMORPG's [2]. From this point the game industry has never looked back and evolved in great scale, and games became more available, sophisticated, complex and graphical over time.

With the powerful development of the Internet, the opportunity to make money from games became visible, and the rise of commercial MMORPG's started on the Internet. With the commercializing of games and the use of Internet, the name MMORPG really got introduced, since earlier games could not gather "massive" numbers of players. In 1997, Ultima Online was released and many claim that this game contributed strongly to the status and popularity of the MMORPG genre. Ultima Online featured 3D third-person graphics and was a more complex game than many of its predecessors. The MMORPG entered a new generation based on the development that games like Ultima Online strongly contributed to. So when the game market entered the new century, many new game genres had expanded from the MMORPG, and new game consoles such as PlayStation 2 and 3, XBOX and XBOX360, Nintendo Wii were invented. They quickly became popular, giving even new possibilities to the already expanding game market. The game companies wanted to exploit this expansion of genres and consoles, and were eager to capitalize on the new market [32].

This expansion of the gaming market, and the requirements of the new games led to enormous costs of running and maintaining the game. Millions of players requires thousands of servers and a huge amount of bandwidth to provide real-time interaction with the game world. Establishing new titles

in a market with such high starting costs requires either a lot of investors, willing to take a risk or starting extremely small and expand very slowly. In the fast-paced world that the gaming industry is, the game could be obsolete before even reaching a profitable number of users.

Even with the introduction of new and popular consoles offering new types of genres, the MMORPG stood firm and survived all the challenges from the new consoles and genres. Today, the most played and popular game in the world, World of Warcraft created by Blizzard, is the pioneer in the market and has millions of players, making Blizzard a multi billion company just based on this game. The MMORPG still appeals to the game playing people out there, and stands as the most powerful multi billion game market with unheard possibilities. The genre is still evolving and to satisfy the ever growing demands, it's getting even harder to enter the market for new companies.

Ablemagic wants to enter the old and strong MMORPG market with their ideas and concept of the game 'tisu. To be able to do so, they have to establish a firm foundation that will be able to cope with the massive number of players that is wanted. This foundation consists of the graphical game client and the engine behind it, but also the network concepts and architecture, making it ready to enter the MMORPG market. With an architecture that contains the properties to be flexible, reliable, scalable, functional and secure, the foundation for a good MMO-system is established. With functionality that will ensure an approved performance level, Ablemagic will stand prepared to conquer the MMORPG world and market.

1.2 Motivation

The motivation for this thesis was to promote a cost efficient MMORPG networking architecture. Ablemagic is a small company trying to take on big market leaders that are running established titles. This highly competitive industry requires a small developer to think and do things differently to be able to enter, and we aim to present a solution able to compete with the today's existing standards while operating at lower costs.

1.3 Objective

The objective of this master thesis is to present a design for a cost effective P2P networking solution for MMORPGs, and implement a working demo to give proof-of-concept for it. A theoretical study must be performed to achieve a solution that has the required features for a state of the art MMORPG title today, and to present the reader with the background to understand the challenges and choices made in the process. Additionally, the design presented here should be able to function as a traditional client-server scheme, making it usable for game developers who want this.

1.4 Methodology

During this work, we first did a theoretical study of both the technical aspects of MMORPGs and some of the state of the art titles on the market today. This was done as a foundation for specifying the functional requirements of the system and was based on both scientific articles on the subjects and books written by people with experience in the field. These requirements along with non-functional ones were used to design an SDL design for the system. Finally we implemented this design in Java and performed a set of tests on the system, and evaluated the empirical results.

1.5 Scope

The number of aspects and challenges in a MMORPG networking solution are numerous. Security, bandwidth and delay optimization, integration with a graphics engine etc. are essential in a final product. However, the development of functionality for making a playable game alone is a major task. This thesis therefore focuses on the features that are required to make the concept work, leaving the design open and flexible with regards to other essential features.

The scope is to form a basis for further development of a multiplayer

network architecture for Ablemagic's game 'tisu. This means to describe and highlight essential requirements and functionality that should be included to create a reliable and well functioning network architecture. Because of the time limit of the work with this thesis, a full functional solution integrated with the 'tisu demo will not be possible. The focus will rather be to, based on the theoretical research, create a flexible design that can be used as a foundation in the future and implement a proof-of-concept for visualising our research and thoughts.

For this reason, the aspects of security, details on mobile terminals and optimization have not been prioritized in this thesis. Furthermore, the hardware resources available makes thorough testing of the implementation with a high number of users impossible.

1.6 Related Work

The Universities in Mannheim, Stuttgart and Hannover have in collaboration performed research on Peer-to-Peer-based MMO gaming. This work has been made generally for all types of MMO games, and is based on a pure P2P topology. Their work outlines the requirements of such a solution, in terms of consistency, persistency, scalability, security, efficiency, etc. Their work is closely related to a project called peers@play which attempts to create distributed interactive game worlds with P2P, but the development is still in the prototype stage. [22]

There has also been developed a pure P2P MMO FPS game demo at the University in Mannheim, utilizing Skype for the infrastructure. This work is also planned to be used in evaluation of the peers@play project. [26]

Research at the University of Pennsylvania has been done to evaluate an approach to support massively multiplayer games on P2P overlays. Their research is on MMGs and therefore have lower requirements for persistency, scalability and self-organization as an MMOG. However, their work on pure P2P topologies and thorough testing concludes that the concept works in massive scale, but more work beyond their demo is required. [16]

1.7 Document Structure

The thesis consists of seven chapters in addition to references and appendices. The chapters of the thesis are structured as follows:

- Chapter 2: Describes different theoretical and technological aspects from a top to bottom approach that may be included in a MMORPG design.
- Chapter 3: Examines the requirements that should be considered when designing the 'tisu system. The chapter also evaluates existing MMORPG solutions, both server software and titles on the market.
- Chapter 4: Presents the proposed solution in detail.
- Chapter 5: Provides a walk-through of the SDL design and a description of the implemented demo and includes a section on how to run the software.
- Chapter 6: Describes the tests performed to evaluate the system and presents the results and an analysis of these.
- Chapter 7: Conclusion and future work.

Chapter 2

MMORPG Design Considerations

2.1 Introduction/MMO/MMORPG

2.1.1 Massively Multiplayer Online game

The Internet gave birth to massively multiplayer online games. MMOGs are operating over the Internet and creating connections between the different players participating, and are capable of supporting several thousands of players simultaneously. This gives players the possibility to socialize through the game and the Internet, and gives the MMOG concept a great advantage over singleplayer games [1].

A typical MMOG consists of at least one persistent world where players can cooperate and compete against each other on a large scale. The world will continue being functional if a user leaves the world and game, and let others continue their play unaffected by someone leaving. With the development of the MMOG concept, several different kinds of MMOG genres with different properties have evolved. Some of the most popular types of MMOGs are as follows [29]:

- MMO Role Playing Game
- MMO First-Person Shooter
- MMO Real-Time Strategy
- MMO Sports Game

2.1.2 Massively Multiplayer Online Role Playing Game

Since the explosion of popularity for the MMO games, it has been created several sub-genres. One of the oldest and most popular genres is the MMORPG. The MMORPG genre has been under constant changes since the development and importance of PC's, game consoles and the Internet. From simple plots and 2D graphics, we now demand a complex and meaningful story with rich and convincing 3D graphics. Today, the big

leader in the MMORPG genre is Blizzards World of Warcraft, with millions of players.

The main purpose in most of the MMORPGs is the progression and development of your own avatar, to make it strong and ready to explore new continents and areas of the total game world. The world can consist of several "continents" that are specific for some kinds of avatars, and you can travel around as you build up the strength of your character. Players are often allowed to choose how they want to improve their character's performance in terms of attributes, skills, special abilities, trade and buy equipment from other players and earn experience points. One common way to develop your character is that you need to socialize and cooperate with other players to be able to defeat evil opponents around the world [30].

2.2 Game Worlds and Load Distribution

With the great evolution of MMOGs and an explosion of concurrent players taking part, the game experience requires a server environment that distributes game related computations across different units that can process them. One well-used method for achieving this form for distribution is by adding functionality into the game that divides the world into regions that is managed by different server processes [1].

2.2.1 Game World types

Zoning is a term used for describing the dividing and distribution of responsibility of the game world into different zones and server processes. As the game evolves and gains more players it will be a functionality of great necessity to distribute the responsibility and processing power, so that the experience of the game continues to be at a valid level.

There are many different ways to zone the playing area of the game, and some zone constructions give less inter-server communication, but the downside could be a bigger complexity in the dividing-process and the computations involving player crossing borders. Another issue that should be counted for is the need for a contiguous game world or not.

2.2.1.1 Zoned Worlds

The term zoned worlds means that the game world consists of independent geographical areas between which there is no contact between players at different areas. This means that players only can interact with objects on the same server. This is a far less complex way to design your game, and creates less inter-server communication since players of different areas does not have knowledge of each other at all. So the only time different zones has to communicate with each other, are when players goes from one area to another.

This results in what is know as a free-format. This is an approach where there is no need for a contiguous world, and the different worlds and zones can be abstracted into separate areas and allows limiting the number of player entering any partition. Figure 2.1 at the side shows a potential free-format arrangement of zones and the connections between them. In this type of structure it is not possible for players in different areas to have any knowledge of each other. This format is a much less complex way to divide your game if there is no need for a contiguous world, but this is also a less flexible one. It can work as a solution for zoning players between continents or separate worlds within the game.

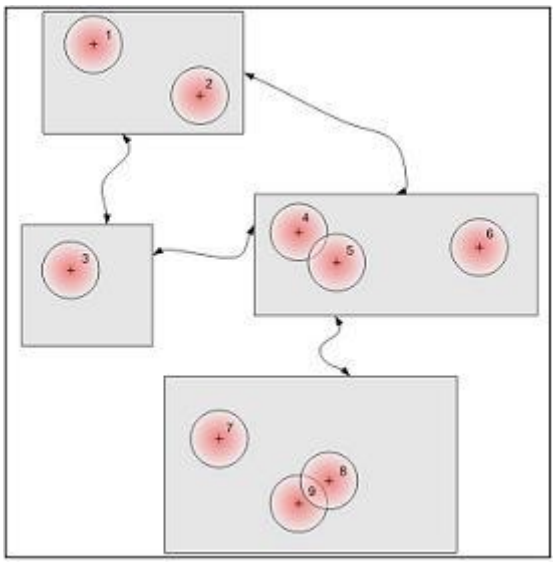


Figure 2.1: Free format partioning

2.2.1.2 Seamless Worlds

The idea behind a seamless world is to divide the game world into different zones, and add inter-server communication between neighboring zones that allow each server to work as an autonomous unit. The way to achieve this is by having servers notify the servers responsible for adjacent zones when objects are close to the borders of the respective zones, such that each server can interact with remote objects through their local proxies. Proxies will be described in detail in a later section. [1]

How the seamless approach will work, depends on the results from different decisions that has to be made. The zoning process affects the outcome of the seamless solution, and this process is a very game specific issue. This gives rise to the question of how to divide the world.

One method used when there is a need for a contiguous game world, is the division of zones into hexagon shaped areas. The positive side by using hexagons to divide the zones is that you are creating smaller maximal number of adjacent zones at the edges. This means that at most, the zone has to establish inter-server communication with two other zones when players are entering the area around the sharp edges of the zone. Figure 2.2 shows a game world that is divided into zones with a hexagon structure. From player4's perspective it has an AOI that enters zone F and G, but this is also the maximum number adjacent zones that the AOI can discover. This approach of partitioning the game world will have the benefit of less inter-server communication, but a disadvantage is that the geometric calculations are more complex than in other schemes.

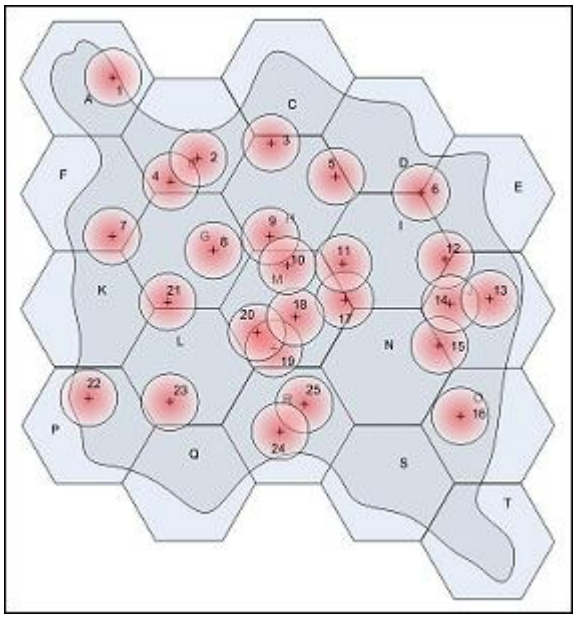


Figure 2.2: Hexagon partitioning of game world

Another and commonly used way to zone when there is a need for a contiguous game world, is squared structures. With the use of squares it will be potential for more inter-server communication at the sharp edges of a zone than by using hexagon structures. From figure 2.3 it is possible to see that player7 has a AOI that goes over zone E,F,I and J and explores player9. This means that the server process in charge of zone J and player7 needs to establish inter-server communication with zone E,F and I to get information about the

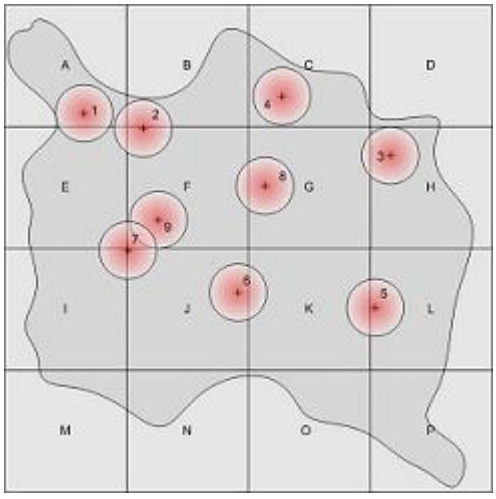


Figure 2.3: Square partitioning of the game world

content in the AOI of player7. This scenario shows that with a squared partitioning there will at most be needed to establish connection to three other zones, and when this needs to happen with 100 different players it would be generated great amounts of traffic between the respective zones. One benefit with this structure of partitioning is the simpleness of the calculation that has to be made when players move into borders.

The next thing to consider is how large the shared boundaries between the adjacent zones should be. At a minimum, they need to be as large as the client's game world or player's AOI. By not keeping this in mind when designing the shared boundaries, the game will be subject to cheating and exploits. Figure 2.4 depicts this. Server A and B are neighbors and share a common border. P2 is inside the shared buffer zone, as P1 is outside. P1's AOI contains P2 as can be seen from the dotted circle. Since P1 stands outside the shared buffer zone, it means that P1 is not visible for server B, and then P1 will not be visible for P2 even though they have the same AOI range. P2 will not get any information about P1 since server B knows about the existence of P1. This scenario leads to the possibility that P1 may be able to attack P2, but the reverse is not possible.

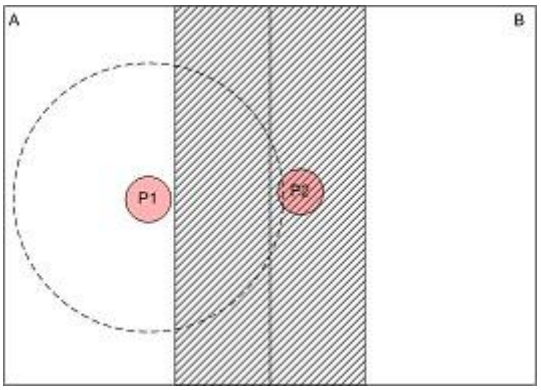


Figure 2.4: Buffer size scenarios

Another technical detail that has to be evaluated, is whether the server boundaries should be soft or hard. A hard server boundary means that the playing object will be transferred to the neighboring server as soon as it has crossed, with no delay. A soft boundary allows an object to "stray a little" across the border before initiating a transfer. This is a technique that can be a helping factor for minimizing the expensive inter-server communication that is generated with object transfers, but is a more complex solution to use.

Another major concern that should be evaluated, is whether the server boundaries should be static or dynamic while the game is running. In a

dynamic scheme, the size of the zones will dynamically change based on the game load while the game is running, causing the zone borders to be redefined. With a game that assumes dynamic adjustable server boundaries, gives a significantly more complex design and implementation. Dynamic borders can lead to transfer of large groups of players that should not cause any noticeable delay for the specific player. If the method itself is given, all the regulations that have to be established to determine when and how to adjust the server boundaries may be a complex task. With an algorithm that too frequently changes the borders can lead to a lot of overhead, and with an algorithm that changes the borders to infrequently can lead to overburdened servers. An example of the latter case would be if a lot of players gathered around the same place due to some new functionality being added to the game in this area. So the algorithm for borders changes at use should ensure a balanced load between servers.

With all these aspects in mind when choosing a solution of a seamless world, it is clear that there are challenges regarding design and implementation of MMORPGs.

An obvious advantage of a seamless world is the possibility to have larger contiguous areas to play the game. With a zoned solution, the server responsible for the zone can only support an area as large as the number of players it is capable to handle. Seamless worlds give a feeling of an artificial reality and a more real simulation for the players participating in the game. Another and crucial advantage with the choice of using a seamless world is the increased scalability. As the world grows in popularity and becomes more populated, the servers can adjust to handle the increased load. If dynamic borders are adopted into the game design, the case above can be treated at run time with a well working self-balancing algorithm. This also shows the multi-functionality a seamless approach can have on a MMOG. The result of a seamless approach is the potential for higher reliability. With a seamless solution, it will be possible to treat a breakdown from a server or server process, then the server boundaries can be adjusted to spread the load over the remaining servers. An extra-strength, dynamic server boundary implementation might even do this automatically when it detects a server crash. Dynamic server boundaries can give an extra benefit in the way that they can contribute factors for preventing the ability to exploit the extensive classes of bugs that involve moving data back and forth between severs. The reason for this, is that the players do not have knowledge about where the server boundaries are, and when they can be

found, the servers can readjust before an exploit can be triggered. Another benefit caused by the choice of a seamless world is the minimization of the loading time experienced when players change zones.

Even with so many obvious advantages of a seamless world, there will also be some disadvantages connected to every technical solution. With a seamless solution, the main problem will be the degree of complexity that a seamless design entails. Seamless worlds will create great implications for the design and implementation, and can be a source of error for many parts of the game. In a multiplayer game, like MMOGs, there are always some degree of inaccuracies from client to client. In a seamless server environment, the same type of uncertainty also resides on the server side. These proxies that are being created when players enter the shared buffer between different zones, introduces inaccuracies for remote objects. These proxies cannot be completely synchronized with adjacent zones without generating large amounts of overhead.

2.2.1.3 Proxies

As mentioned in the section above, the border regions between different zones contain objects that can interact across a server boundary. When a player enters zone buffer, the controlling server sets up a communication channel to its neighbors. With the information sent over the channel, the neighbors can create proxies to represent the remote object. When a player then crosses the border and enters an adjacent zone, the proxies are destroyed so that the objects only exist like a normal object, and not as proxies and objects. An issue that occurs with the generation of proxies with a seamless solution, is the amount of information that a proxy object should include. With the proxy object containing too little information, it will be very difficult to write any game code that does not involve asynchronous message-passing. The object containing too much information will lead to a need for much more data to be kept in sync, and the consequence is that large amounts of data can be out of sync [1]. With the use of a seamless world and proxies, it will be generated more inter-server communication. To minimize this sort of communication generated from the proxies, a set of prioritized properties for the objects could be established. That is, identify individual characteristics that need to be mirrored, and decide how rapid the update should be based on the

characteristic that changes.

2.2.1.4 Realms

When the structure regarding zoning, game world types and corresponding functionality of the game has been established, a well working instance of the game can be created. However even with a well designed system there are more issues that has to be considered. When the amount of players increases in the instance of the game, the density of players eventually will reach a maximum where the game experience will decrease dramatically. With too many players gathered in the game world at the same time, it will be a need to create new instances of the game world to spread players. These different instances of the game are called realms. A Realm is a kind of kingdom or domain in which something, like player characters, are dominant [21].

2.3 Game Synchronization

The main objective for the game server / game servers in an MMO is to keep the game state synchronized for all players. This means that the server needs to consider two scenarios that need synchronization:

- *The server makes a change to the game state, such as adding a new object to the game world.*
- *A client makes a change to the game state, such as dealing damage to another clients avatar.*

Both cases require that the server synchronizes the change with the clients that are aware of the area where the change is made. There are numerous ways of doing this, but the main objective in designing a synchronization mechanism is to limit resource use (i.e. bandwidth or CPU usage). The game state consists of all the dynamic objects in the game and the parameters that define them. Not all parameters should be constantly kept

synchronized, while others should. Keeping a parameter synchronized is referred to as state synchronization. Other objects could be synchronized using RPCs. State synchronization differs in reliability, whereas RPC always is reliable.

2.3.1 Reliable State Synchronization

Reliable state synchronization is done over a reliable connection, meaning lost packets will be resent. Resending packets will cause later packets to be delayed, which could cause latency jitter for the game. Using reliable state synchronization should therefore be limited to events that do not occur often, or to critical events. This method also optimizes the game for bandwidth, and causes potential latency issues.

2.3.2 Unreliable State Synchronization

Unreliable state synchronization causes potential packet loss. This could again cause a client not to be made aware of a change in the game state, but this potential loss can be compensated for if the state is synchronized often. Information about an avatars position and rotation are examples of such states. This will optimize the game for low latency, but use more bandwidth.

2.3.3 RPC

For some actions, like adding and deleting objects in the game, there are no parameters to synchronize. RPCs are commonly used for such actions, allowing the game server to call procedures on the remote clients object hierarchy.

2.3.4 Dead Reckoning

One problem that arises when using a fully authoritative game server is that the game clients will only update their states when they receive states from the server. This causes a delay from the player gives the game input to the client actually renders the action. This delay will make the game interaction feel very unnatural for the player. To avoid this problem most networked games using authoritative servers incorporates dead reckoning. This technique makes the client estimate the action as soon as the player enters input, and then compare this estimate with the state update from the server when it is received. Dead reckoning is applicable to other game objects as well as the players avatar. By smoothing the correction, this effect can often be unnoticeable to the player.

2.4 Security and Cheating for MMOGs

2.4.1 Security issues regarding MMOGs

The concept of a MMOG, is that a great amount of players around the world can gather and play the game which is one of the strengths of MMOGs. However, with this as basis the different involved players do not necessarily know each other, and could potentially expose each other for different kinds of treats. Players can be exposed for virtual crime, and experience to be hacked by other players to gain advantage in the game. In order to be a trustworthy game, the game developer must provide mechanisms that ensures secure user accounts and safe payment options. The consequence of players not trusting the operator to be able to keep credentials and financial data secure will lead to players abandoning the game.

Another choice that will affect the security level that the MMOG can provide is the choice of what authority the servers should have. There are two different server approaches that can be used; authoritarian and non-authoritarian. A non-authoritarian solution gives the server no authority, and brings the responsibility over to the clients. This is a

non-implementable solution if the case is a MMOG. In a non-authoritarian scenario the client will have the freedom to do whatever it wants, and will be free to break the regulations of the game. It will be naive in a MMOG scenario to believe that no client will exploit this freedom, and break the regulations to their best. On the other side, the authoritarian alternative will give the server or different responsible servers authority to correct illegal actions clients perform. To have a MMOG that works as intended with the best possible game experience, the servers should have large degree of authority.

While the game is running, it is important for players to be anonymous for other players, but the degree of anonymity will vary based on the game design and architecture. With a client-server based architecture, the player will only be registered on the controlling server, and the operator of the game is the only unit that holds information about the different players. The IP-address is the most essential information about a client, and should be available for a minimum units taking part in the game. A solution based on the client-server approach has the advantage of keeping the clients IP-addresses only known to the operator of the game, who should be a trusted party. The security aspect is a major concern when the architecture of the game is based on P2P, as described later, and there should be carried out thorough evaluations with the advantages and disadvantages it brings along. A game constructed with P2P-architecture will expose the players IP-addresses to units that are in control of the different zones the game consists of. Because of this property, P2P will risk the clients anonymity, and by sharing the IP-address between different zone masters, the clients will be in danger of becoming a victim from different kinds of security attacks, like DoS.

2.4.2 Cheating concerns regarding MMOGs

With the rise of MMOGs, the problem of online cheating has evolved. Cheating in online games are activities that modify the game experience to give one player an advantage over another player(s). There are several ways to cheat, and some are more complex, comprehensive and harmful than others. This section will highlight which challenges involving cheating that comes along with developing an MMOG.

The potential and possibility to be exposed to cheating will increase as the servers loose their authority. With an authoritative solution the IP-address will not be exposed, and possibility of cheating over the network decreases. With a P2P architecture as the foundation, the responsibility of the game will be divided and distributed to different clients based on the responsibility-scheme that has been established. This makes the game prone to scenarios of cheating depending on the power each zone master is in possession of. With a completely non-authoritative solution, all the playing clients taking part in the game can abuse the power and knowledge they possess. This allows them to perform illegal actions, like harming other players, without the possibility to correct the actions because of the missing authority of the main servers. At the same time players can take advantage of the power they possess and cheat by causing updates to game state that defy rules of the game or gains an unfair advantage [3]. With a semi-authoritative solution, the clients that only act as players, have less opportunities to perform different cheating scenarios, but will experience the opportunity to be exposed for cheating. This is the case since the different zone masters will be in possession of the name of the different players staying in their zone. With this name they can identify where in the zone the player stays and what kind of actions he performs. If the zone master decides that it will deny a player the intended degree of freedom, it has the potential of limiting the actions that the player tries to perform. Another way to abuse the power that zone masters possess, is to collaborate with other players, and tell them where objects of great benefits lie in the zone.

To cope with the problem of cheating, there are some areas where it should be put effort to minimize the damage caused. With a distributed solution like the P2P alternative the clients with zone master responsibility constitute the biggest threat to exploit any weaknesses of the architecture. One way to avoid this will be to write code that executes the zone masters tasks hidden behind a "black box", such that the clients with zone master responsibility do not have any knowledge about what happens in the code and are unable to take advantage of the information they possess. This must be implemented in combination with encryption between the zone masters and game servers. This will make it harder for a zone master to understand what kind of information that enters and leaves the "black box".

2.5 MMORPG Engineering

2.5.1 Flexibility in MMORPGs

To keep generating revenues after the growth phase, an MMORPG, like any other service, needs to keep its users returning, and do so by adding new content. Since the RPG genre is built around evolving an avatar, and completing various goals and achievements while evolving it, the game will run out of content at some point if no more is added. This means that an MMORPG should be designed to allow easy adding of game content. Making sure the game is modular will provide this flexibility, and the components in both the server- and client- parts of the software should be designed as separate modules.

2.5.2 State Machines

State machines contain a set of states, and actions cause transitions between the states. This simple functionality makes state machines easy to describe visually, and makes an application transparent and easy to develop. It also provides a clear separation between modules in an application, since state machines essentially just consumes signals describing an action. This requires clearly defined communication between state machines. The concept of state machines also suits the RPG genre well, since the player in controlling his avatar is basically transitioning the avatar between states (idle, moving, jumping, etc.) by performing input actions on the computer.

2.5.3 SDL and Real Time System Engineering

SDL is a language that is customized for designing systems that are:

- Reactive
- Concurrent

- Real-time
- Distributed
- Heterogeneous

SDL evolved from the need for a language to design new complex systems with high traffic load, real time requirements and a high operational time without malfunctioning. Because of this background, SDL has proved to have a wide utilization potential even for systems outside the intended purpose. The intentional use of SDL in system development is to specify the required system behaviour, to design and generate an optimal implementation and to document the provided behaviour. With the use of SDL it will be possible to create implementation-independent specifications of behaviour, but at the same time a description of the behaviour actually implemented, hence the name Specification and Description Language. [5]

The problem that needs to be answered with the use of SDL is reactive behaviour. The goal is to describe as little as possible about the internal, physical construction of the systems, while telling the full story about how external pressure and response are related at the interface. To achieve this goal, SDL is constructed as follows:

- Systems and blocks composed from blocks and channels.
- Blocks composed from processes and signal routes.
- Process behaviours composed from sub-sequence called procedure.

2.6 Protocols/Latency

2.6.1 UDP

UDP is one of the protocols in the transport layer of the TCP/IP model. UDP is a connectionless protocol, and requires no set up before transmission of the message. This makes it an unreliable protocol. Packets

can be lost, arrive out of order or become duplicated, but a packet is guaranteed to be whole if it arrives. UDP assumes that error checking and correction is done in the application layer, or is not needed by the application. The nature of UDP gives it lower transmission times and less overhead compared to TCP.

RUDP is an extension of UDP, providing additional features to gain reliability. These are acknowledgement, congestion control, retransmission and over-buffering. RUDP is not currently a formal standard.

	Bits 0-15	16-32
0	Source Port (optional)	Destination Port
32	Length	Checksum (optional)
64	Data	

Table 2.1: UDP Packet Structure

2.6.2 TCP

TCP is another transport layer protocol. Unlike UDP, TCP is connection-oriented, providing a reliable service. TCP provides an ordered stream of data, with acknowledgements, retransmission of lost packets, discarding of duplicates, error checking, flow control and congestion control. TCP also requires set-up to establish a connection. This is done using a three-way handshake. TCP has the advantage over UDP in reliability, but the cost of this reliability is bandwidth due to overhead and latency due to the additional processing and controlling.

2.6.3 Transport Protocols and Game Development

The choice of protocol in a game depends on the game type and the design of the game. Some games require reliability more than low response times, and vice versa. Games requiring low response times include RTS games, FPS games and some RPG games depending on combat systems, player interaction, and other real-time requirements. Turn-based games however will not have the strict real-time requirements, and reliability becomes more

	Bits 0-3	4-7	8-15	16-32
0	Source Port			Destination Port
32	Sequence Number			
64	Acknowledgement Number			
96	Data Offset	Reserved	Flag	Window Size
128	Checksum		Urgent Pointer	
160	Options (optional)			
160/192+	Data			

Table 2.2: TCP Packet Structure

important. When running a server park, having thousands of simultaneous connections, the bandwidth usage and hence the costs of operating can vary a lot depending on the choice of transport protocol.

2.7 Network Topologies in MMOGs

2.7.1 Centralized Networks

A centralized topology is by far the most common in MMOGs. A centralized server contains the game state, and the clients make requests to change the state. The design of the server can be either authoritative, non-authoritative or a hybrid. In the authoritative case the client is not trusted to make changes in the game state, but rather tells the server what actions it would like to take. The server then evaluates the request and decides whether the action can be performed. This puts heavy load on the server, particularly for 3D games, where the server is forced to perform collision-detection for all objects for all clients. In the non-authoritative case, the server allows the client to update the game state. The client can perform the necessary checks, and send the result to the server. This design causes a high risk of cheating clients. A hybrid scheme can therefore be an effective solution, leaving checks that to a lesser degree affect cheating to the client, while the server remains authoritative for important checks.

2.7.2 Decentralization

In a decentralized topology, there are several distributed servers connected by a central server. Each distributed server keeps a part of the game state and can act as an authoritative server regarding changes to the servers state. In an MMORPG, where the game world typically is split into zones. This means dividing the zones over a set of distributed servers and let each server manage the game state for its zone or zones. Having a decentralized system makes the game more scalable, both in terms of an increase in the number of players in the game world, and in terms of adding more game content (i.e. game expansions).

2.7.3 P2P

P2P topologies can be divided into two categories: pure and hybrid. A pure P2P network has no central servers, and all clients store a part of the service content and essentially act as both a client and a server. In a hybrid P2P network a central server manages the network, and keeps track of who is part of it and what content each client has stored. Also clients can be classified in a hierarchy. Using pure P2P solution in an MMOG would cause three critical problems:

- *The game network could risk getting split up in smaller network, lose parts of the game state or dissolve entirely.*
- *The clients would have no main point of connection, and hence have no stable way of connecting to the game.*
- *The game would have no means of a central authority, meaning any client would have potential to cheat.*

A hybrid P2P solution changes this. A central server keeps track of the clients which eliminates the risk of the network getting split up or dissolving. It also provides a connection point and a potential authority. However, the system would still be prone to losing data unless the central server at all times keep a backup of the game state. Forcing all clients to

backup changes to a central authority would defeat the point of the P2P network, hence the clients needs to be classified into different categories. The result is a centralized and decentralized hybrid P2P network. By using a selection algorithm to choose a client to serve as a master and authority for a part of the game state, and have this client backup changes made to the game state to the centralized server, the load on the centralized server is reduced while the risk of losing game state changes is eliminated.

2.8 Mobile Terminals

Mobile devices, such as cell phones and WiFi enabled portable MP3 players are becoming increasingly popular for gaming. These devices are characterized by small screens, limited input options and limited hardware resources (CPU, memory and battery). This often limits the games on such devices to simpler games, but hardware is improving and will likely allow more advanced gaming experiences in the future [27].

In addition to limited hardware, mobile devices often connect to the Internet using mobile phone networks. The connection depends on the device and the coverage of the operator. In Norway, Telenor covers most of the country with EDGE, and has UMTS and HSDPA in most urban areas [25]. This results in data transfer rates from about 200kbit/s to 3.6Mbit. Round trip delays average from 76ms (HSDPA R5) to about 120ms for EDGE. Both EGDE and UMTS have high variance for round trip time, causing delay jitter [10], [14].

Since real time MMOGs require low latency and more importantly stable latency, mobile devices are at the current time not well suited for real time MMOGs, but with future mobile data technologies mobile devices could prove usable. In this case for a peer-to-peer MMOG, mobile clients could easily be excluded from zone management and preserve its resources.

Chapter 3

State of the Art MMO Solutions

3.1 State of the Art MMORPG solutions

To understand the challenges of designing and implementing an MMORPG game, and to understand the competition and costs of operating in the business, the state of the art must be reviewed. Though most game development companies are highly secretive about their products, some knowledge can be gained from various sources. We chose to study some popular titles in the genre:

3.1.1 State of the Art MMORPG Games

3.1.1.1 World of Warcraft

World of Warcraft is an MMORPG game in the fantasy genre released by Blizzard in 2004. The game takes place in the world of Azeroth, which consists of two major continents, and a set of minor islands. In December 2008, Blizzard claimed to have more than 11.5 million subscribers [9], making it one of the most successful games of today.

Though Blizzard is sparse with details about the technical foundation of the game, it is still known that the game is based on a client/server scheme, and had in 2006 9000 servers supporting the roughly 6,5 million subscribers it had at the time. As of May 2009, World of Warcraft has more than 200 realms for each of its three main regions: U.S., Europe and Asia. This is more than 600 realms globally. Though the hardware of the servers is unknown, it still means that each of the realms are supported by multiple servers, in the area of 10-20 each [12], [28]. In addition to moving around in the world along with all the other players, World of Warcraft has a concept called "instances". This allows a group of players to enter a private area where players outside of the group cannot enter. This separates the group from the rest of the game world, and allows it to take on challenges uninterrupted by others. World of Warcraft uses zoning when the player moves between the continents, and when players enter "instances". This provides effective load balancing as the game world can be split over several server clusters. In addition to zoning, each of the continents are split into smaller zones. These zones have seamless borders, though several zones are

Figure 3.1: Ingame screenshot from World of Warcraft

likely run by one server. World of Warcraft relies on TCP transmission unlike most real time applications. Latency is reduced by sending small packets, which causes high overhead. The study shows median bandwidth usage of 6.9 kbit/s and 2.1 kbit/s for the downlink and uplink respectively [24].

3.1.1.2 Age of Conan

Age of Conan is a fantasy MMORPG released in 2008 by Funcom. Age of Conan is an example of how hard it is to establish a game in this genre. Only a short time after release, Funcom announced that one million copies had been shipped [11], but shortly after release it also became apparent that the game suffered from several problems, and many players claimed the game simply wasn't finished when released [19]. The problems didn't get fixed, and by early 2009, Age of Conan reduced the number of realms from 48 to 16 [7]. Though World of Warcraft in its early days also had issues such as server crashes, poor class balancing and lacking features, Blizzard's several years of development after release has improved the game. This has made World of Warcraft the "de facto" industry standard of MMORPGs in the fantasy genre. Blizzard reported late in 2008 that 68 per cent of the players that left World of Warcraft in favor of Age of Conan had returned [18].

Figure 3.2: Ingame screenshot from AoC

3.1.1.3 EVE Online

EVE Online was released in 2003, and is an MMORPG in the science fiction genre. EVE Online separates itself from a number of MMORPGs in the sense that it does not have multiple realms. The game has about 250000 subscribers, and all players play in the same universe, consisting of 5000 unique solar systems. Since the game takes place in space, and travel between the solar systems are done by going through "wormholes" eliminates the need for a seamless world. EVE Online is essentially a zoned, single realm MMO game running on a single supercomputer called Tranquility [20].

Figure 3.3: Ingame screenshot from EVE Online

3.1.2 Existing Server Software

Developing an MMO game can be an expensive project. Complex multiplayer solutions take a lot of time to engineer, and require thoroughly testing. Opting to use existing software can save a lot of time and money, but can on the other hand also limit the flexibility in the game. Analyzing the features of some potential server software is necessary for Ablemagic to make a qualified decision on how to proceed with development of 'tisu.

3.1.2.1 SmartFoxServer

"SmartFoxServer is a multi-platform socket server designed to integrate with Adobe Flash, enabling developers to rapidly develop multiuser applications and games. The server was created with multiplayer games in mind and provides powerful tools for creating a wide range of sophisticated turn-based and real-time games. There's really no limit to the number of applications that can be created with it."

However, SmartFoxServer lacks features needed for a large scale 3D MMORPG. No support for seamless worlds, no mention of dead reckoning, only client-server topology support and no unreliable data transfer options. SmartFoxServer's showcase of games only shows simpler web based online games. Though SmartFoxServer can integrate with Unity using its .Net API, this is more suited for such simpler games, such as 2D games with "3D look", not for an MMORPG like 'tisu. [13]

3.1.2.2 NetDog

"NETDOG is an out-of-the-box, highly scalable networking solution for developers of Massively Multiplayer Online Games and Virtual Worlds. NetDog is easy-to-use, flexible, and game-engine independent, helping developers bring worlds to market faster, easier and cheaper by creating scalable, high performance networks that support millions of users."

NetDog is designed to support MMOGs and Virtual Worlds, and has a

substantial set of features required for games, including:

- Both client-server and distributed topologies
- UPD and RUDP protocols
- Multi-zone worlds with dynamic resizing and seamless worlds
- Delta compression and dead reckoning
- Transparent encryption and iPhone support
- 3rd party game engine integration using C-bindings

NetDog appears to be a solid piece of software, and has all the main features required for a large scale MMOG such as 'tisu. However, these features come at a price. The Indie licence for NetDog costs 30000$ for a title, and is intended for "multiplayer and small MMO games" developed by small independent studios. The Pro license is based on revenue sharing. NetDog has only been on the market since early 2008, and at this time no titles using it has been released. This makes it hard to evaluate the stability and potential of the software. [15]

3.1.2.3 RakNet

"Raknet is a cross-platform C++ game networking engine. It is designed to be a high performance, easy to integrate, and complete solution for games and other applications."

RakNet describes its major features as lobby system, autopatcher, object replication system, RPCs, secure connections, voice communication, robust communication layer and NAT punchthrough. These are features required in many multiplayer games, however for an MMOG, there are no features supporting massive worlds. No support for seamless worlds or zoning exists. These features could be developed on top of RakNet, but again these features which includes synchronization of objects over zone borders are more complex features and would require a lot of development. The licencing is practical for small developers. The "Free Indy licence" is free

for developers with a gross revenue under 250000$. This licence can be used while developing, until the revenue exceeds this amount, and then the "Commercial Single Application Licence", which allows the use of RakNet as a single application on a single platform, is available for 5000$. [17]

3.1.2.4 Project Darkstar

"Project Darkstar is software infrastructure that aims to simplify the development and operation of massively scalable online games, virtual worlds, and social networking applications. Originally created by Sun Microsystems, it is today advanced as an open source project through the Project Darkstar Community."

Project Darkstar is currently in version 0.9.9, which is not intended for production. This version currently only supports client-server topology with a single host. This limits the use of Project Darkstar at the moment, but coming versions will support this. Project Darkstar offers core network services, and a different approach to scaling. It supports use of zones and realms, but the architecture is based on breaking down the game servers work into small units, which each can be processed on the server-side network. Each server on the network breaks down the the work unit for parallel processing in multiple threads. The goal is to achieve near linear scalability. The advantage of this approach is that no resources are wasted on zones with low activity, but can be used to process activity in high activity zones. But the architecture is limited to client-server, and Project Darkstar is using TCP for transmission, limiting the flexibility in development. But being open source is a huge advantage in that the software is freely available. [23]

3.1.3 'tisu requirements

'tisu is a game in the MMORPG genre. The game is in a concept / early development stage, which makes it hard to specify requirements for it. There will exist a game client for PC/Mac where you play an avatar in a 3D world. Figure 3.4 shows an ingame screenshot of the single player demo's character "Lube" and its interface. Players will in the final game control avatars like this in a massive world together with other players. These avatars will have the ability to interact with the environment and other characters in the game, both friendly and hostile. The game world will be huge, sustaining a large amount of players.

The bandwidth requirements of World of Warcraft are not significant in a client-server solution, but in P2P, managing nodes need to be able to sustain the accumulative bandwidth of all clients connected. Should a zone contain 50 players, a roughly 3Mbit uplink is required to sustain the median traffic, and more is required to sustain peak traffic. These are huge requirements for a user to meet. However, bandwidth can be reduced from the level of World of Warcraft by utilizing UDP or RUDP, and also by engineering better compression for data transfer. Zone sizes can also be adjusted to reduce bandwidth.

The number of users returning to World of Warcraft after initially leaving it for Age of Conan, shows the importance of differentiation in the market. If the game becomes a substitute for an existing title, loosing users can be a major threat. This can be seen in the Age of Conan case where the numbers of servers needed to be significantly reduced, which causes a major cost for the developer.

It should, however, be noted that there are several sub-genres within the MMORPG genre. Both World of Warcraft and Age of Conan are in the fantasy genre, while EVE Online differentiates itself by being a science-fiction game. This sort of differentiation is important for developers to keep in mind. The similarities and differences of these games are illustrated in figures 3.1, 3.2 and 3.3.

Since the single player demo already defines much of the functionality from the users point of view, it will rather be from a developers in this section. That is, networking functionality that are transparent to the end user, but

Figure 3.4: Ingame screenshot from 'tisu

will have a huge impact on the developers and game designers, such as seamless worlds or not. Non-functional requirements will also be of high priority, since the game is in an early stage, and much of the decisions regarding functionality is yet to be decided. These are basic requirements that are evaluated when considering the choice of networking system for 'tisu.

The single player demo of 'tisu has been developed using the game engine Unity. The Unity engine comes with an integrated editor to design 3D worlds, and supports various scripting languages. Development in Unity is focused on visual design, and relies on the scripting engine to create game events and behavior. Since Unity is used for 'tisu, a high priority requirement for a networking solution is that it integrates with Unity.

3.1.3.1 State Synchronization

Being a 3D world where the player is interacting with the environment in real time, the game needs some form of state synchronization. It should support decentralized solutions to provide low latency for a geographically distributed mass of users.

3.1.3.2 Load Balancing

'tisu is a game that aims to support thousands of simultaneous players in a realm. This requires some form of load balancing. The game needs to support zoning between the different worlds in the game, but should also support seamless worlds to provide extended load balancing. This will provide a better experience for the end user as he will be able to traverse entire continents without interruption, and the gaming experience will feel more real.

3.1.3.3 Mobile Terminals

'tisu is a game that will focus on mobile clients. First out are simpler games that can be supported by today's mobile technology. These should allow the player to improve his or her avatar by completing missions or objectives. This requires the database to be accessible by simpler clients.

3.1.3.4 Cheating

'tisu aims to become an MMOG with thousands of players. This requires the game to be cheat proof. The most common way of cheat proofing multiplayer games is using authoritative servers. 'tisu should use authoritative game servers to keep players from cheating. Using authoritative servers will also require the game clients to have a prediction mechanism such as dead reckoning.

3.1.3.5 Modularity

In a game that will evolve dynamically, new content must be possible to add without changing the dynamics of the network system. It must be possible to add new game objects, new events and expand the world in a separate content system without making changes to other parts of the game.

3.1.3.6 Cost

As a small new company it can be hard to compete against established games. Cost of operation can be devastating when releasing a new service, and server maintenance and bandwidth costs can become huge. If these costs were entirely variable, there would be no problem, but since servers must be bought and bandwidth lease must be negotiated, these costs will not be lowered immediately if the number of subscribers should fall, whereas income will. The system should hence keep the cost of operation at a level that makes it possible for a game title to dynamically expand into a solid competitor.
From these requirements, the one of the most important objectives for a small developer is to reduce costs and get an edge on the competition. To be able to launch a product at all, the product certainly needs to satisfy the user in form of functionality, but if the game is too expensive to maintain and service, the title may never see the light of day. We therefore choose to focus on a P2P concept solution, to evaluate how this performs. Since none of the existing server software we reviewed has this feature, we will compare the results of the work with existing software for Ablemagic in appendix B.

Chapter 4

The Concept

4.1 Concept

The main focus of the concept in this chapter is to present a cost saving solution. By using P2P networking solutions in an MMORPG, huge amounts could potentially be saved on maintainance and lease of servers and bandwidth. The concept presented below ensures the requirements highlighted for 'tisu, and could work as a basis for further development for a functional full scale MMORPG platform.

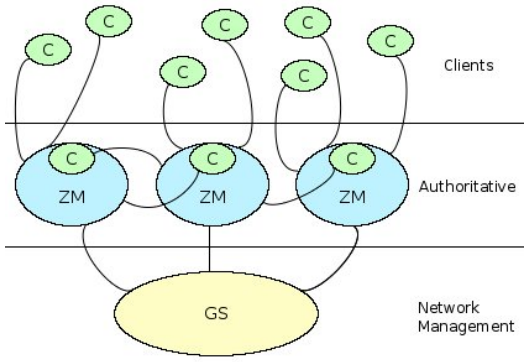


Figure 4.1: Topology layers of the concept

This illustration shows the proposed concept. The game world is split into zones, and each zone is managed by an authoritative unit called "zone master" that ensures that the players are not cheating. These zone masters send backups to the game server containing the changes made to the world state (i.e. delta compressed). This ensures no data is lost should a zone master disconnect. The zone masters can either be run by the game server itself or by connected clients. The game server manages the network, deciding which clients should manage zones. It can change zone masters for a zone at any time. When a new zone master is appointed, the game server provides it with the game state of the zone. The game server also provides zone masters with game objects of new avatars that enters the game when clients connect, and has authorization information about clients. This results in a hybrid P2P scheme. Some clients are "super nodes" in the network, and constitute the authoritative level in the structure. The top level is basic clients, that still have the capability to act as zone masters, while the bottom level manages the entire structure. The ability to run zone masters on both dedicated servers and on the game server itself, provides the flexibility to run the game both as a centralized or decentralized scheme in addition to P2P.

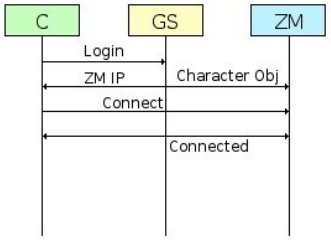


Figure 4.2: MSC diagram of client connecting

Figure 4.2 shows a client successfully connecting to the game, highlighting the role of the game server as the network manager. The client first logs in to the game server, which provides it with the IP address of the zone master it should connect to based on the location of the players avatar. The avatar game object is then sent to the zone master. The client now connects to the zone master, which has the required information. To ensure that the system is secure, clients should authenticate themselves with the game server, and a challenge sequence should be issued for the zone masters to authenticate clients.

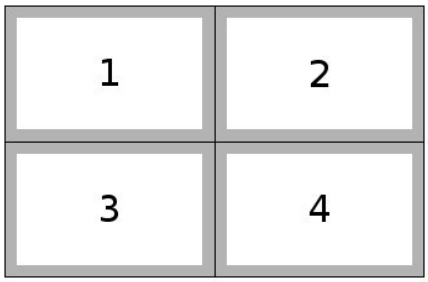


Figure 4.3: Zone borders and buffers

The concept includes a seamless zoning approach of the game world. This enables the players to see, and even do ranged actions across zone borders. The concept therefore requires communication between the zone masters. We choose a simple solution where the buffers are the size of the range a player can see, and establish proxies on the adjacent zone masters when an object enters the buffer. This increases complexity, and establishes a

network between zone masters for keeping proxies synchronized with their real objects. Figure 4.4 illustrates this network.

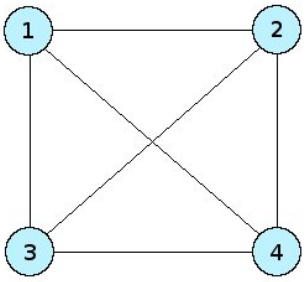


Figure 4.4: Network between adjacent zone masters

Even though the requirements are fulfilled, we see that there exist weaknesses compared to the commercial solutions used today. The main weaknesses the concept faces is cheating and keeping the players anonymous. To reduce cheating it is essential that a client that acts as a zone master does not have conflicting interests. This would occur if a client is playing in the same zone as it is managing as a zone master, or nearby zones due to ability to alter objects through proxies. The concept therefore requires the game server to swap zone masters should a client enter a zone adjacent to the zone it is managing. Though it does not completely eliminate cheating, it reduces the potential. As the priority has been to develop a functional concept, the security issues have largely been left to future work. End-to-end cryptography can improve security by disallowing the zone masters insight into the data they are processing, but solutions for keeping the clients anonymous in a P2P scheme does not exist as of today. Security functions can, however, easily be implemented into an end product without requiring major changes to the core functionality of the system.

As mentioned above, this solution is designed with cost saving in mind. This is the main argument behind the choice of a P2P topology. Using true P2P with no centralized server could cause loss of data when peers disconnected, and would make it hard for the game producers to generate revenue from the game. This is why a game server manages logins and backs up the world state. The game server keeps the zone masters at a "need to know" basis, providing them with only the necessary information about the state in their zone. This is done to ensure limited possibilities of

cheating, and also reduce the amount of data that needs to be transferred to the zone masters at the time of establishment. Seamless zones are required in a concept like this. Game clients run on typical consumer computers with limited resources available for additional tasks. Bandwidth are also often limited. This requires the size of the zones to be adjusted to the resources a client has available, and this could mean small zones. Loading a new zone often could greatly annoy the players and cause them to abandon the game for competing titles, and it is therefore very important to ensure a smooth transition between the zones in such a concept. Even though the increased network traffic causes extra load on the zone masters, it is required. It should however be noted that the concept developed uses rectangular zones. Using different shapes such as hexagons can reduce the number of connections to the hosts, but due to limited time for development we had to choose the simpler implementation.

Chapter 5

Design and Implementation

5.1 Design and Implementation

To implement complex network software a solid design is required to coordinate the process. A lot of signals are sent between the modules and components in such a system, making it essential that all developers know what input and output is sent to and from the software part he is working on. To establish this, we chose to produce a detailed implementation design in SDL. SDL has a lot of the required properties for a distributed real-time system with high concurrency demands like 'tisu [6]:

- The functionality of the system can exist and be understood separately from the implementation of the system.
- The functionality is independent and can be used for different implementations.
- With a complete functional design a automated transformation to an implementation is possible.

This allows the game developers to utilize the design to implement the system in a different language than the conceptual demo we developed, and provides an excellent overview of the signalling between the components.

As SDL is based on state machines, we implemented a simple run-time framework for state machines in Java. State machines provides great modularity, and implemented in an object-oriented language, each process can easily be implemented in a separate class. This results in a transparent solution that is easy to maintain and develop further. Java was chosen as the implementation language as it has a range of useful libraries that makes it easy and fast to develop. Swing was used for the simple GUI. Though not an optimal solution, network communication was implemented using Java sockets with TCP transmission which provides a stable transmission, but consumes more resources than a solution utilizing UDP for certain transfers. Where objects were needed to be sent, they were serialized to XML, and sent as string arguments. XML causes a lot of overhead, and is not suitable in a final product, but ease of implementation was prioritized. State machines were implemented in an action oriented style and scheduled in a separate thread, with a synchronized method "addInput", that allowed other state machines to add signals to the machines FIFO queue.

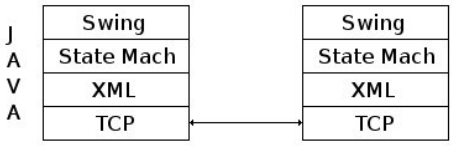


Figure 5.1: Technology used in implementation

5.1.1 System overview

The overview of the 'tisu implementation design is divided into two main blocks. With these two blocks, a connection and different communication channels can be established between the playing client and game server in a way that the game will work as intended.

The client part of the system consists of two separate modules that will make the client capable of taking on different roles depending on the need of its processing power. The first module the client contains is the client module. This is the part that establish the connection with the game server, 3D engine and zone masters of the game. [1]

The C-GS Connection channel and GS-C and C-GS ports, are generated to initiate the authentication and login sequence, and hold the connection to game server throughout the game. This connection enables the game server to inform about architectural changes that happens during the game, like information about new zone masters, new adjacent zone masters or that you should take on the role as a zone master.

The Game channel with Gout and Gin as ports have the main function of disclosing game information out to 3D engine. This means that game related information like state changes and procedure calls that are being performed, and affect the respective player or players in the AOI, are distributed to the correct instances.

With connection sat up with 3D engine and game server, the next step for

[1]The figures showing the SDL design in this chapter are extracts from appendix A, which contains the complete design. For greater detail, the appendix should be reviewed.

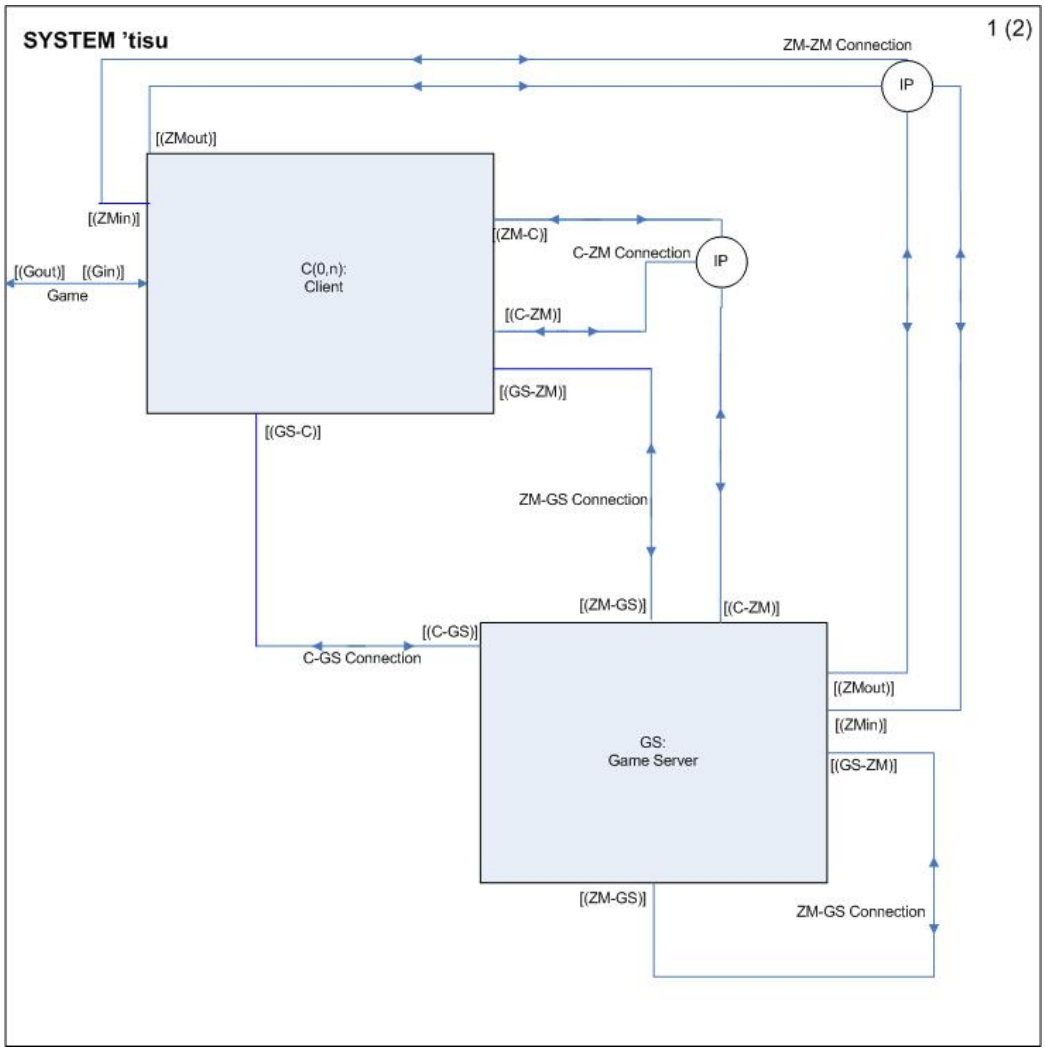


Figure 5.2: Top level SDL design of the system

a client will be to initiate contact with the zone master responsible for the zone where the client's character is staying. Before game information can be shared between client and zone master, the client has to be authenticated to show his/her trustworthiness. The authentication challenge will go through the C-ZMCH and ZMCH-C ports and on the ZM Connect channel. Depending on a successful authentication sequence the client can start to initiate in-game exchange of information on the C-ZM Connection through the ZM-C and C-ZM ports. The essential in-game information to exchange with zone master is the change of states and

procedure calls that will affect objects that are in one characters AOI, and objects that need to be added to the AOI of a character, and general AOI updates for a synchronised gameplay between different clients.

The second module that the client contains is the zone master module. This is a more complex module, and gives the client the possibility to function as a zone master when the game server is exposed to high demands of processing power. The zone master module is not visible in the system overview, but it is an integrated part of the client and game server. From a client's perspective it receives information that it should take on the role a as zone master from game server, and then initiates a sequence of processes to be able to function as a zone master. The first step is to generate a new communication channel, ZM-GS Connection, with the game server where zone critical information can be exchanged through the GS-ZM and ZM-GS ports. The zone master module will receive information about the zone that it shall control, arrival of new adjacent zone masters or to stop functioning as a zone manager.

With the connection to the game server established, the next step will be to initiate contact with the adjacent zone masters, such that in-game information could be shared between zones. The ZM-ZM Connection channel are constructed for this purpose and the ZMin and ZMout ports will be able to distribute information about the creation, deletion and updates of proxies. There have been added IP-notations to the system illustration, these are meant to illustrate the possibility to differentiate between which participating part in the game architecture that should receive the information. The case with the ZM-ZM Connection is to differ between which hosts with zone master responsibility that should get information about proxies.

The game server side of the system overview also consists of two parts. One module will contain the main game server properties, this includes the login sequence for the clients, distribution of information about the zone that the different clients should connect to, delegation of zone master responsibility and keep a connection to every zone master. This module will establish connection between clients and zone masters through already mentioned channels and ports. C-GS Connection will see through the communication for the login sequence. The ZM-GS Connection will distribute architectural information to the correct zone master. ZM-GS Connection will distribute architectural information and changes differentiated by zone master

IP-address during gameplay.

The second module of the game server is a zone master module equal to the one that is integrated in the client. The only difference between these two is the relationship that the zone master module needs to keep to the game server module which is a part of the same unit. For the game server there exists two channels with the name ZM-GS Connection. One of the channels keeps the connection to external zone masters for exchange of architectural information, the second channel will exchange the same type of information, but to the zone master module in the game server that is in control of a zone. The game server will be responsible for all the zones up to a certain point where it needs more processing power and distributes zone responsibility to external hosts. This is the reason why it needs to have a connection between two internal but different modules. The C-ZM Connection will exchange information about the changes that affect clients during gameplay, and the ZMConnect channel will be the channel at use when clients connect to the zone. The ZM-ZM Connection will be the channel to communicate inter-zone information.

5.1.2 GameServer

The GameServer block contains, in addition to the common ZoneMaster processes which will be detailed below, the five GameServer types of processes:

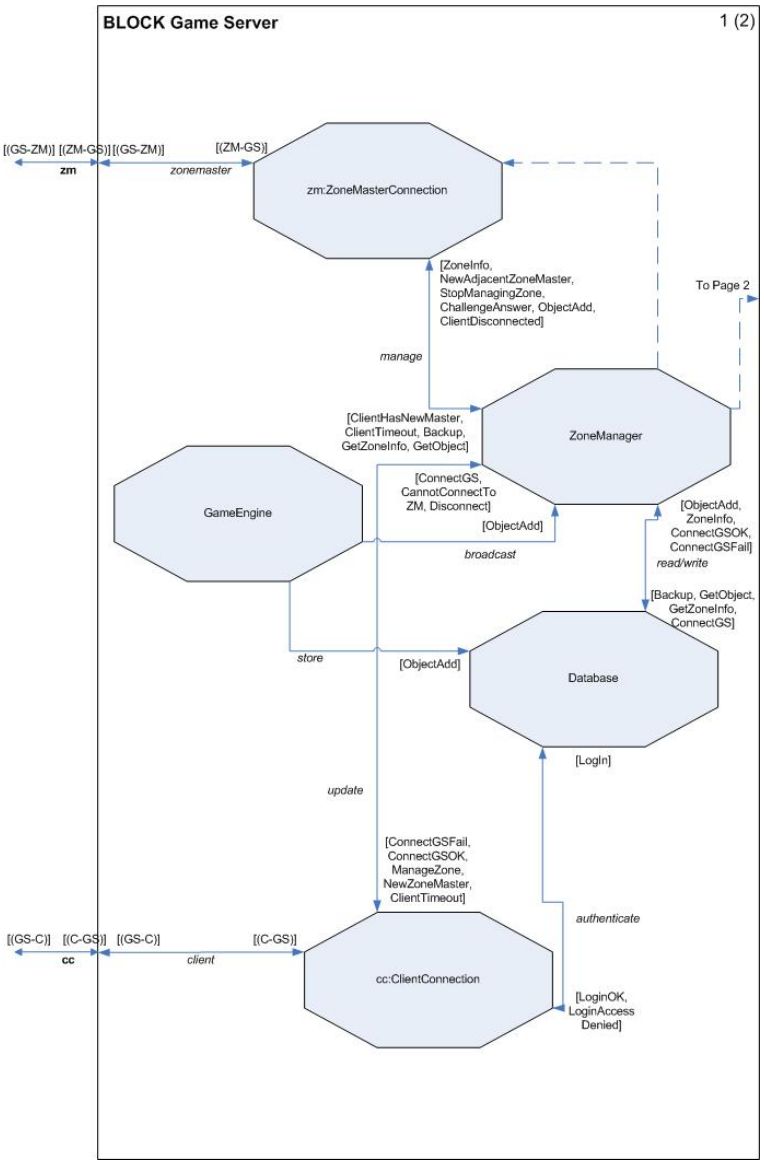


Figure 5.3: SDL design of the Game Server block

ClientConnection
The purpose of the ClientConnection is to maintain communication between the gameserver and the clients. There is one process for each connected client, and each process provides authentication of the client, authorization for the chosen avatar and the following network messages: ManageZone - Tells the client to start a zonemaster and manage a zone. NewZoneMaster - Informs the client that it has a new zonemaster. This is used when a zonemaster drops out suddenly. CannotConnectToZM - Used for the client to inform the gameserver that it cannot connect to its zonemaster. This is required to detect faulty zonemasters.

GameEngine
The GameEngine process is not described in detail because the inner workings of it needs to be made by game designers. However, it has the ability to send AddObject signals to both the Database and the ZoneManager processes, which is the only signal required. The GameEngine is intended to be the process where the game content is managed, and should keep track of game world events that requires new objects to be added to the world.

Database
The Database is a process that acts as an adapter to the underlying database, which at all times contains the current state of the game world as well as account information for clients. The process authentication and authorization, stores the backups the gameserver receives from zonemasters and provides the ZoneManager process with objects and zone information.

ZoneManager
The ZoneManager contains the mapping between clients, zonemasters and zones. When a client connects, it provides it with connection information about the zonemaster it should connect to. It selects clients to be zonemasters, and replaces zonemasters when needed. When a zonemaster is started, it connects to the gameserver, and the ZoneManager process provides the zonemaster with the gameobjects that are currently in the zone, and maps the zonemanager to the zone. Backups are then received regularly from the zonemaster, and stored in the database by the Database process. In addition, the ZoneManager receives notifications when a client changes zonemasters or a zonemaster loses connection to a client.

ZoneMasterConnection

Communication between the GameServer and zonemasters are done through the ZoneMasterConnection processes. One process for each connected zonemaster provides zone setup and transfers messages between the zonemaster and the ZoneManager process.

5.1.2.1 Game Server implementation

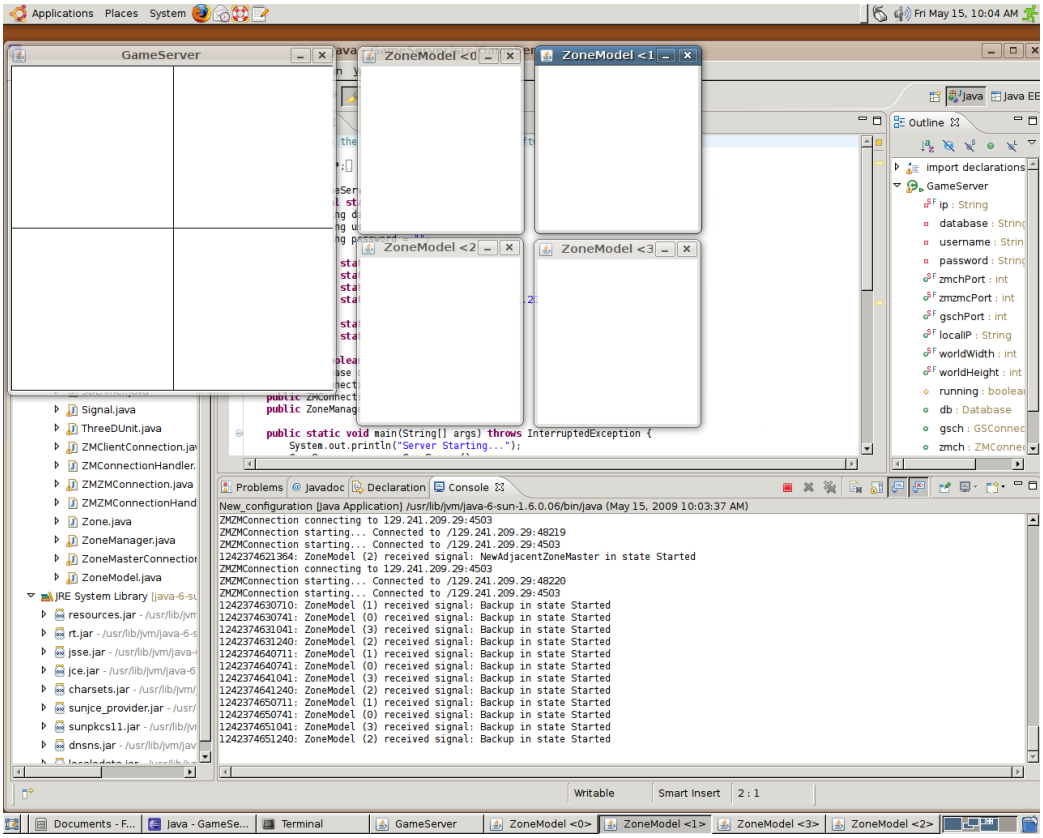


Figure 5.4: Screenshot of GameServer with no clients connected

The GameServer initially defines a world map of 100x100, then defines four zones of 50x50 and starts local zonemasters for these. Figure 5.4 shows a Game Server started, with no clients connected and hence managing all four zones itself. A window is displayed, showing the world map and where the zone borders are. If the GameServer is managing any zones itself a window with the dimensions of the zone is displayed for each, showing any players connected. The zones were in the tests statically defined as 0 to 3.

The GameServer will manage these zones until 5 clients are connected. When the 5th client is connected, a client is selected at random to manage zone 0. When the 6th client connects, the process is repeated for zone 1, at the 7th for zone 2, etc. This is done to set up the zonemasters gradually, as we experienced some problems with setting up the connections between the zonemasters if multiple adjacent zonemasters were started at the same time. This could be solved by adding a delay between the establishment of new ones, but gradually setting up the zone masters illustrated the concept better, and was chosen as a solution in the demo. The GameServer will not let a client manage more than one zone. When all the four zones are managed by clients, the GameServer provides little interesting information to the user except for the backups received by zonemasters and managing clients connecting and disconnecting.

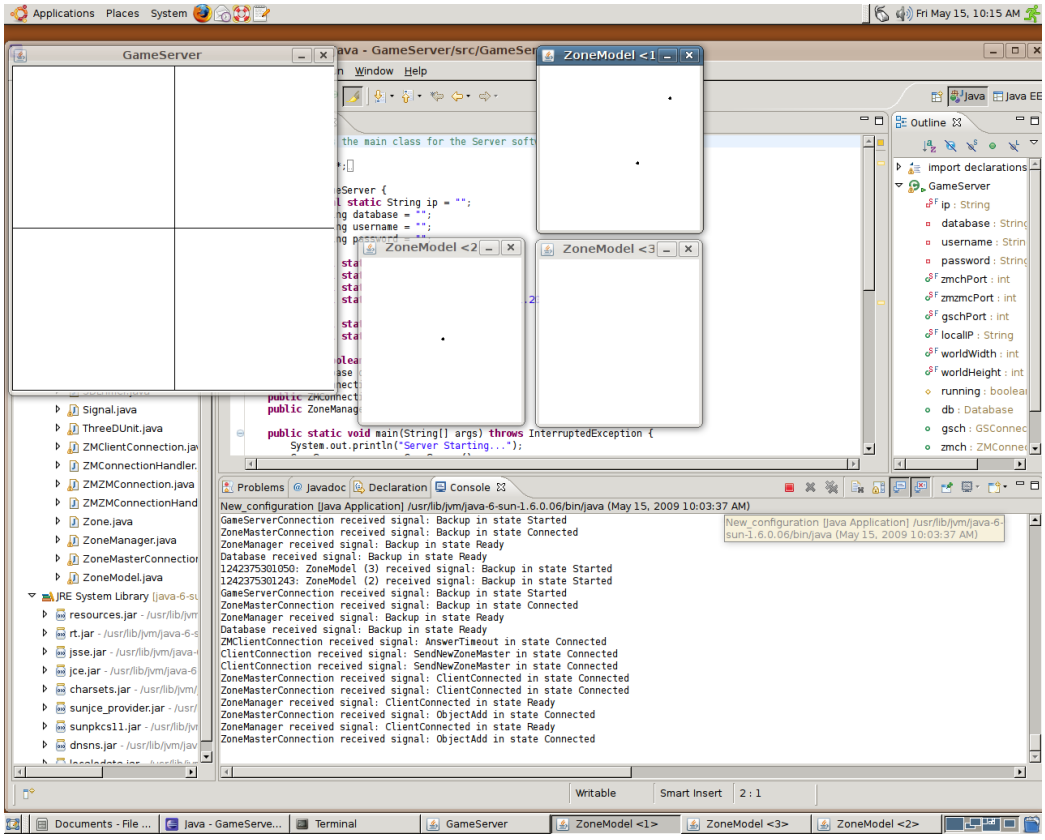


Figure 5.5: Screenshot of GameServer with 5 clients connected and zone 0 managed by a client

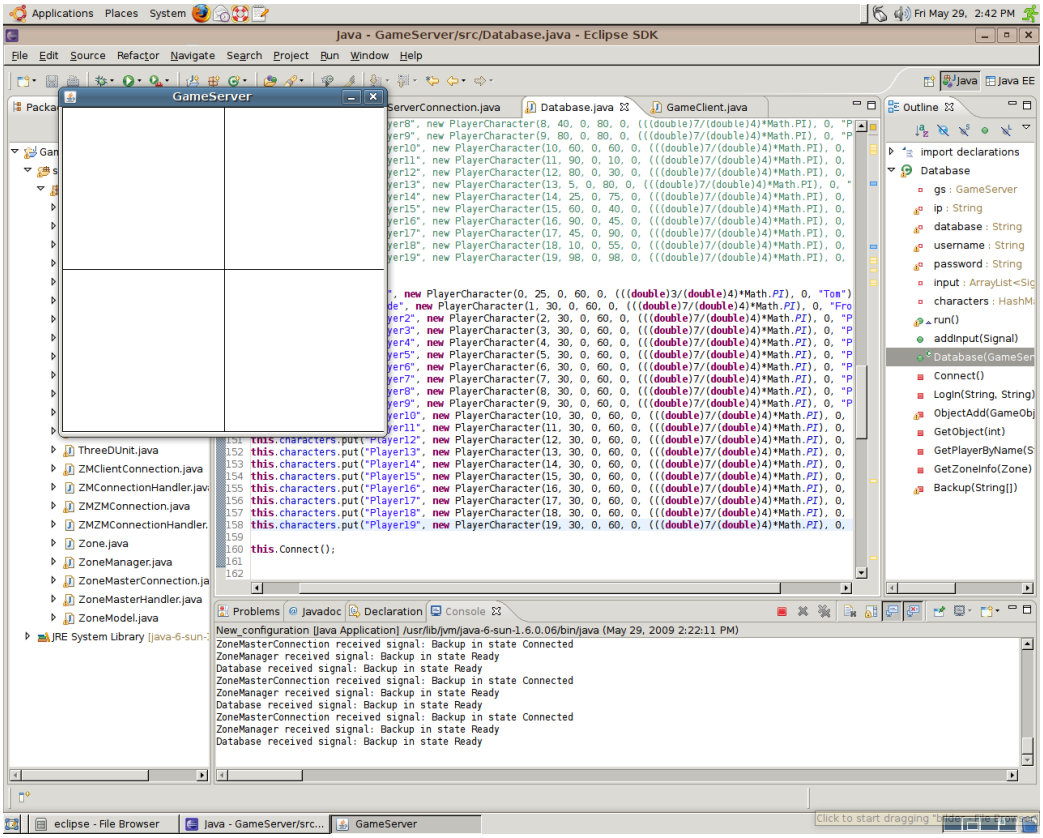


Figure 5.6: Screenshot of GameServer when all zones managed by clients

5.1.3 GameClient

The blocktype Client in the system overview consists of three game client related processes with different responsibility and tasks, but also consists the zone master processes that will be described later. The three main game client processes are divided into:

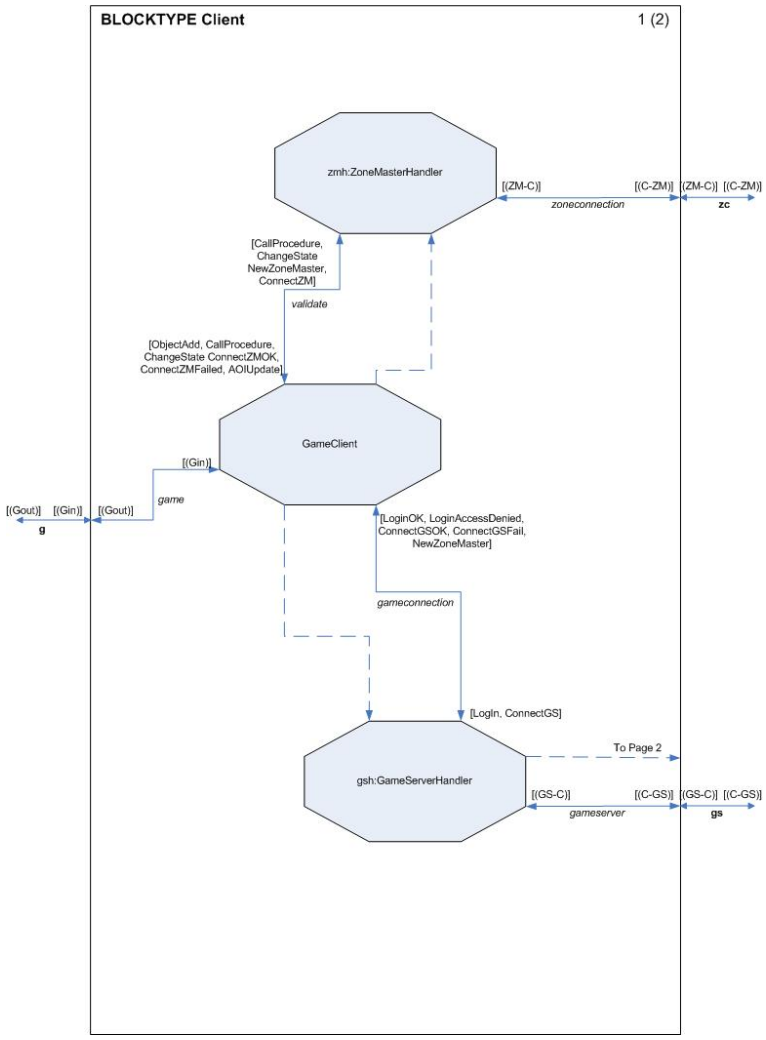


Figure 5.7: SDL design of the Client block

GameServerHandler
The GameServerHandler is the process that maintains and manages the connection to the game server, and will be generated at startup from the GameClient process. It works like a router and forwards signals either to the GameClient process or over the network to the game server, depending on whether the signals are internal or not, through the communication channels gameconnection and gameserver. The GameServerHandler are responsible for seeing through the login sequence over the network with the game server, forward information about new zone masters that are taking control of the different zones and if the client itself should take on the role as a zone master.

ZoneMasterHandler
The other router process that the client consists of is the ZoneMasterHandler, and is also generated from the GameClient process. This is a process that holds the connection between zone masters and the client, through the validate and zoneconnection channels. Based on the type and status of the incoming signal and the state machine implementation, the ZoneMasterHandler decides whether it should forward the signal to external zone masters, or the client. Setting up connections to the different zone master throughout the game, forward updates on actions performed that affect AOI for the respective players in the different zones, are some of the main tasks performed by the ZoneMasterHandler.

GameClient
The GameClient process is the main process of the client and keeps a connection out to the 3D engine and to the two other processes in the Client block. The GameClient process functions as the brain of the Client, and reacts depending on the signal and state machine implementation. This is the process that initiates the different sequences for setting up valid connections for the client. This include login to game server, establishing connection to zone master, start up the integrated zone master module if triggered and forward actions performed from either own player to zone master or actions performed from external players that affect its own AOI.

5.1.3.1 Client implementation

The GameClient displays a window the size of the world map as illustrated in figure 5.8. This size is defined in the client, and must be equal to the size defined in the GameServer. The window implements a keyboard listener that reacts to changes in the arrow keys. When an arrow key is pressed or released, this changes the boolean value "moving" for the players character and a signal is sent to the zonemaster, notifying it of the state change. As long as a characters "moving" value is true, the GameClient will move the character in the direction it is rotated every time it receives a "CheckStates" signal. This signal is on a 20 ms repeating timer. The distance of movement is based on the time elapsed since the last move. The client accepts game objects from its zone master and displays the objects that are within the players AOI. Any state changes, RPCs or AOIUpdates received, updates the objects and keeps the client synchronized with the zonemaster.

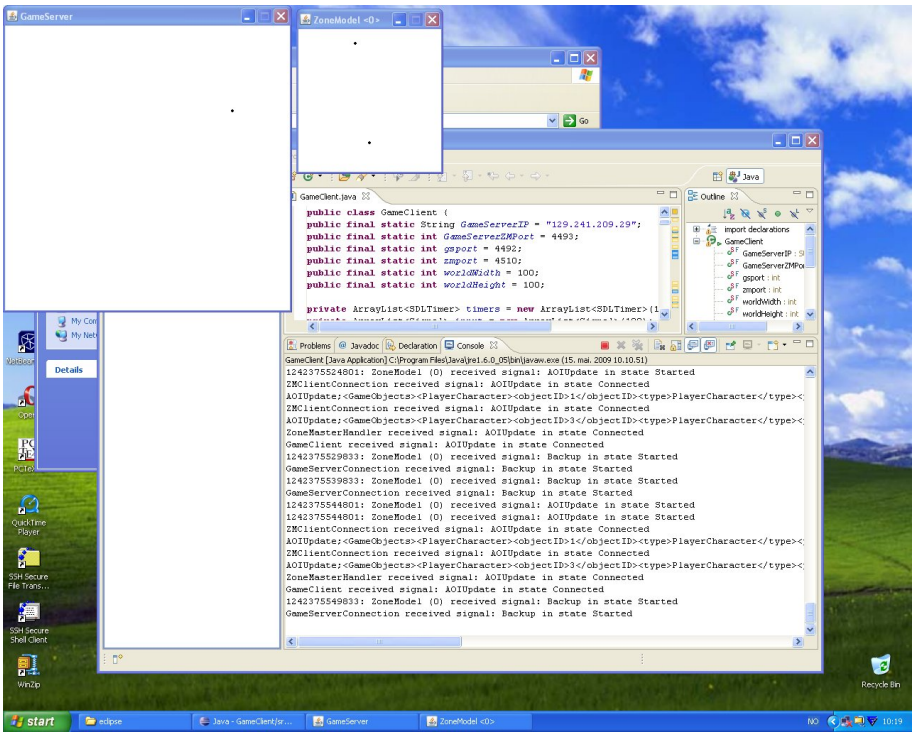


Figure 5.8: Screenshot of GameClient managing zone 0

5.1.4 ZoneMaster

Both the GameClient and the GameServer blocks contain several common processes. These are the ZoneMaster specific processes, which provides both the server and the clients with the capability to be a zone master.

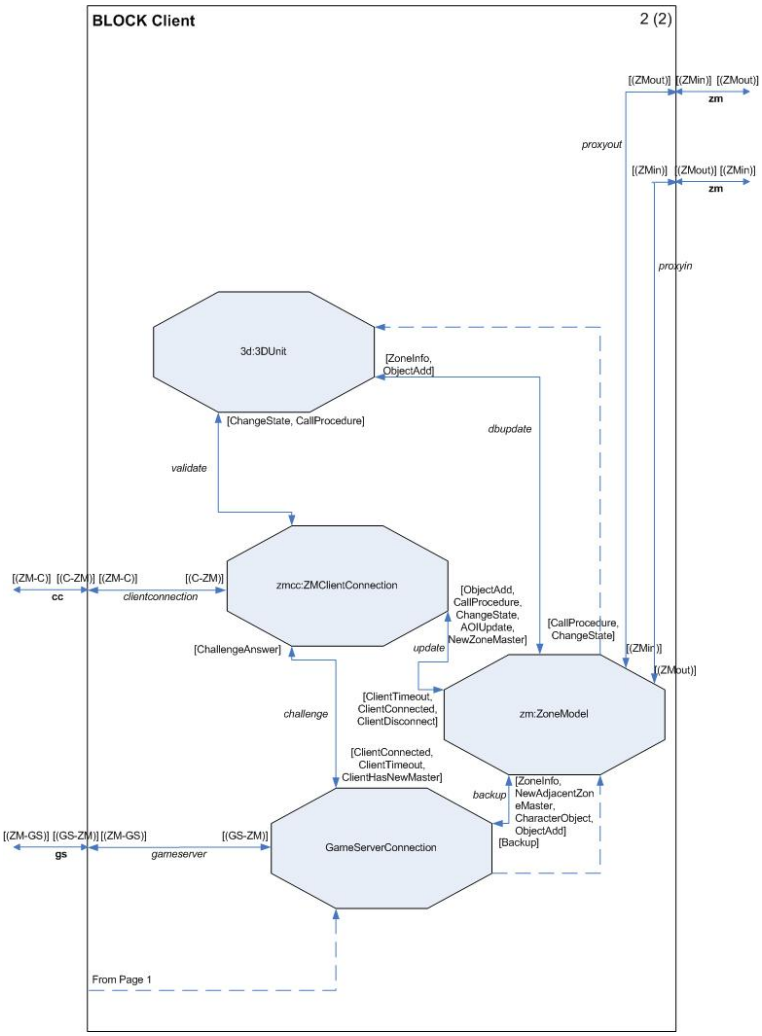


Figure 5.9: SDL design of the ZoneMaster part of the top level blocks

Both the GameClient and the GameServer blocks contain several common processes. These are the ZoneMaster specific processes, which provides both the server and the clients with the capability to be a zone master.

The GameServerConnection process is instantiated by either the ZoneManager process in the GameServer or the GameServerHandler process in the GameClient. The GameServerConnection then instantiates the ZoneModel process, which instantiates the 3DUnit process. For each client connected to the zone master a ZMClientConnection is started.

GameServerConnection
The GameServerConnection process establishes a connection to the gameserver and sets up the zone. After setup, it transfers signals to and receives signals from the gameserver. This includes authorization challenges for clients, changes in adjacent zonemasters, backups, new objects and client changes.

ZMClientConnection
ZMClientConnection processes maintain connections to the clients and works essentially like the ClientConnection in GameServer, except for the authorization, which is done by waiting for the challenge to be received from the GameServerConnection before authorizing clients. The ZMClientConnection receives changes to the game state from both the client and the ZoneModel process. Client changes are forwarded to the 3DEngine for validation, and changes from the ZoneModel are directly forwarded to the client.

3DUnit
3DUnit is a process to enable authoritative capabilities. Any signal that causes change to the game state made by the client is forwarded to the 3DUnit, and evaluated. Only if the change is valid, will the change be forwarded to the ZoneModel. An invalid change is simply ignored, and the error synchronized by regular AOIUpdate signals from the ZoneModel.

ZoneModel
The ZoneModel is the heart of the ZoneMaster functionality. It keeps track of connected clients, adjacent zonemasters, proxies in adjacent zones and all game objects in the zone, and most importantly synchronizes all changes to the zone state between the clients that are aware of the area where the

change took place. The ZoneModel regularly checks states to determine if players are entering buffers and proxies need to be created in adjacent zones, or if clients have left the zone as well as to determine if there are new objects in the players AOI. The ZoneModel notifies clients of their new zonemaster, and also notifies the gameserver of the change when a client leaves the zone. ZoneModels of different ZoneMasters communicate, and update each others proxies to achieve seamless world functionality.

5.1.4.1 ZoneMaster implementation

The ZoneMaster code is common for both the GameServer and the GameClient. The ZoneMaster displays a window showing the zone and all players in it. It runs a repeating timer with 20 ms delay, sending a "CheckStates" signal. On this signal, all connected players that are moving are checked to see if they are entering the buffer at the edge of the zone in which case the adjacent zonemaster is signalled to create a proxy of the player object. Also connected, moving players are checked to see if they are leaving the zone, in which case a "ClientLeftZone" signal is added. Then when the player has been moved, players and proxies within range of the player is calculated, and any changes to other players a player can see are signalled to both the player and the other players. Any changes to proxies the player can see are only signalled to the player. The size of the buffers are equal the distance players can see, avoiding complexity in the proxy mechanisms. The 3D unit is not implemented, but just passes all signals through to the ZoneModel, which makes the server less authoritative. Players will still not be allowed to leave the edge of the world map.

5.1.5 Unimplemented features

Many of the features required to run a full scale MMORPG on this system have not been implemented in the conceptual demo. This is a direct result of limited time to implement, forcing prioritization of features. The implementation design however, was made to be as specific as possible, making sure the design supported all features required for a full scale game. The result is that the Java implementation in some areas differs slightly for the implementation design, and the design specifications have been replaced

by static code that allows the demo to run without the features implemented.

Zone Master selection algorithm
This has simply been done by selecting a zone master at random, and checking that it is not already managing a zone. An effective algorith should be implemented in a final product. Making sure managing clients do not have conflicting interests is a measure to limit cheating, and selecting low-latency clients can improve performance greatly.

UDP
All network communication in the system have for simplicity been implemented using TCP. This was a choice made because time was limited and implementing the system using TCP sockets was the simpler choice. However, messages transmitted between the clients and the game server are limited, and should be implemented using UDP to reduce the number of open connections on the server. As for communication between clients and zone masters, this implementation sends changes to the object properties, which if not received will cause the client and zone master to be out of sync. This makes TCP a natural choice for its reliability. But state changes could be sent using short interval updates, like the AOIUpdate signals but sent every 100ms or more often, making lost or delayed packets useless, and UPD a natural choice. An end product needs to incorporate both UPD and TCP solutions for effective performance.

Dynamic zoning and zone shapes
The zones in the demo are implemented statically. Each zone is defined by two-dimensional coordinates and its width and height. A dynamic solution would move the borders, and repartition the zones while the game was running. There was no time to implement this feature due to the extreme complexity of it, but the signalling in the design would have no problems supporting implementation of it. Also non-rectangular shapes are not implemented. This would require more complex geometry in the implementation of the Zone class.

Zone Edge Detection
Detection of players entering zone edges or buffers is implemented to directly support the four zones set up in the demo. Changing the number or size of the zones set up will require changing of this detection, again due to lack of implementation time.

GameEngine
The GameEngine process is as mentioned left to implement. This should be implemented as a scripting engine to allow game designer to implement scripted events in the game.

Client authentication challenges
Client authentication challenges are sent to the zone masters, but not generated. A challenge generation algorithm must be implemented to create unique challenges for each client.

Database communication
The Database class is not set up to read or write data to a database. All game data is statically added in the class to allow the demo to run.

5.1.6 Instructions for running

The demo was implemented in Eclipse SDK, and run through the Eclipse interface. Before running the demo it must be configured through the following parameters:

- GameServer.java: The string *localIP* must be set to the IP address of the desired network interface.
- GameClient.java: The string *GameServerIP* must be set to the IP address of the game server.
- GameClient.java: The string *CharacterName* must be set to the name of a valid character.
- GameClient.java: The integer *CharacterID* must be set to the matching object ID of the character.

Valid characters can be found in Database.java in the class constructor at line 110.

The characters are added with the following syntax:

this.characters.put("*CharacterName*", new PlayerCharacter(*CharacterID*, 30, 0, 10, 0, (((double)3/(double)4)*Math.PI), 0, "*CharacterName*"));

For example:
this.characters.put("Tom", new PlayerCharacter(0, 30, 0, 10, 0, (((double)3/(double)4)*Math.PI), 0, "Tom"));

Every client must use a unique character.
The servers main method is located in GameServer.java, and the clients in GameClient.java.

Chapter 6

Testing and Evaluation

6.1 System Tests

The system was developed to test and research P2P possibilities in MMORPGs. Basic testing showed that the system worked stable, enabling:

- User login
- Moving an avatar
- Moving between zones
- Updates of avatars in clients AOI
- Clients to manage zones
- Zones to receive new ZM when old ZM disconnects

To evaluate the system, performance must be evaluated. The goal is to discover potential problems with the solution, and evaluate how it performs under the stress we are able to put on it. However, it is important to note that the system is not equal to a full MMORPG. A client running a 3D accelerated game is consuming a lot of resources on its computer, leaving less resources for running a zonemaster. Also an authoritative server will consume more resources than this non-authoritative solution. We are in other words not able to conclude whether the solution is capable to run on an average personal computer in terms of CPU and memory. We will evaluate how the system scales and the latency of the most critical tasks of the solution: Changing zones and zonemasters disconnecting. The tests are performed by measuring the clients perceived delay in the different scenarios. This is important because it is the delay the player is experiencing that determines the acceptable limits, and should be useful to evaluate whether the system can perform sufficiently.

6.1.1 Environment

The tests were performed in the Sahara computer lab at NTNU, which consists of Dell Optiplex GX620 computers with following specifications:

- 2 x Intel Pentium D CPU 3.00GHz
- 1,99GB of RAM
- Broadcom NetXtreme 57xx Gigabit Controller

The GameServer software ran on Ubuntu 8.04, in Eclipse 3.4.2 with javac 1.6.0_06. The GameClients ran on Windows XP SP2, in Eclipse 3.4.2 with javac 1.5.0_06.

6.1.2 Test Setup

There were two main test cases defined, with several different under-scenarios, to evaluate the system performance. Each test was carried through with different amount of clients logged on, to experience if the system's behaviour differed with different stress levels. Each test scenario was performed five times to reduce impact of random factors such as other processes in the system to affect the results.

1. Measure delay when a client switches zones
Reason for test:
This test will give indications of how the system handles to be exposed to different amounts of stress, and will give concrete response to how quick the system responds to the actions performed. The results regarding the system's performance level of stress handling will give indication of where to put an effort for further development and improvement.

How to perform test:
This was tested by moving a player from one zone to another, and measuring the time from the client received NewZoneMaster to it received ConnectZMOK. Also the time from NewZoneMaster to the last ObjectAdd was received was measured. These times describe the latency the client experiences when switching zones, and was measured for four scenarios:

Scenario:
1a) 10 clients connected, but not in the moving clients AOI when it entered the new zone.

Result:

	1	2	3	4	5
1a	2046ms	2046ms	2046ms	2062ms	2046ms

Table 6.1: Test result scenario 1a

The average processing time for scenario 1a is 2049 ms. Table 6.1 shows no major changes and indicates consistency in the system's performance level with 10 clients connected, but not affecting the moving players AOI when switching zone.

Scenario:
1b) 10 clients connected, and all 10 in the moving clients AOI when it entered the new zone. NewZoneMaster to ConnectZMOK.

Result:

	1	2	3	4	5
1b	2077ms	2046ms	2062ms	2046ms	2046ms

Table 6.2: Test result scenario 1b

The average processing time for scenario 1b is 2055 ms. The results above implies slightly more variation in the system's performance when a client connects to a new zone master, and the 10 clients affects the moving players AOI.

Scenario:
1c) 10 clients connected, and all 10 in the moving clients AOI when it entered the new zone. NewZoneMaster to last ObjectAdd.

Result:

The results from scenario 1c show an average processing time of 2421 ms, and shows a rather markedly increase in time from just switching zone to

	1	2	3	4	5
1c	2592ms	2327ms	2437ms	2359ms	2390ms

Table 6.3: Test result scenario 1c

add the 10 objects affecting the incoming client's AOI. The time difference between scenario 1b and 1c based on the average values are 366 ms.

Scenario:
d) 20 clients connected, but not in the moving clients AOI when it entered the new zone.

Result:

	1	2	3	4	5
1d	2093ms	2062ms	2062ms	2062ms	2077ms

Table 6.4: Test result scenario 1d

The average processing time for scenario 1d is 2071 ms. Even though the 20 clients connected do not affect the moving players AOI, the response from the system shows an increase and larger variation in time when the system gets stressed with the double amount of clients, but is not threatening.

Scenario:
1e) 20 clients connected, and all 20 in the moving clients AOI when it entered the new zone. NewZoneMaster to ConnectZMOK.

Result:

	1	2	3	4	5
1e	2061ms	2062ms	2061ms	2046ms	2046ms

Table 6.5: Test result scenario 1e

The results from scenario 1e show the same average processing time as scenario 1b, 2055 ms, even though the amount of clients affecting the

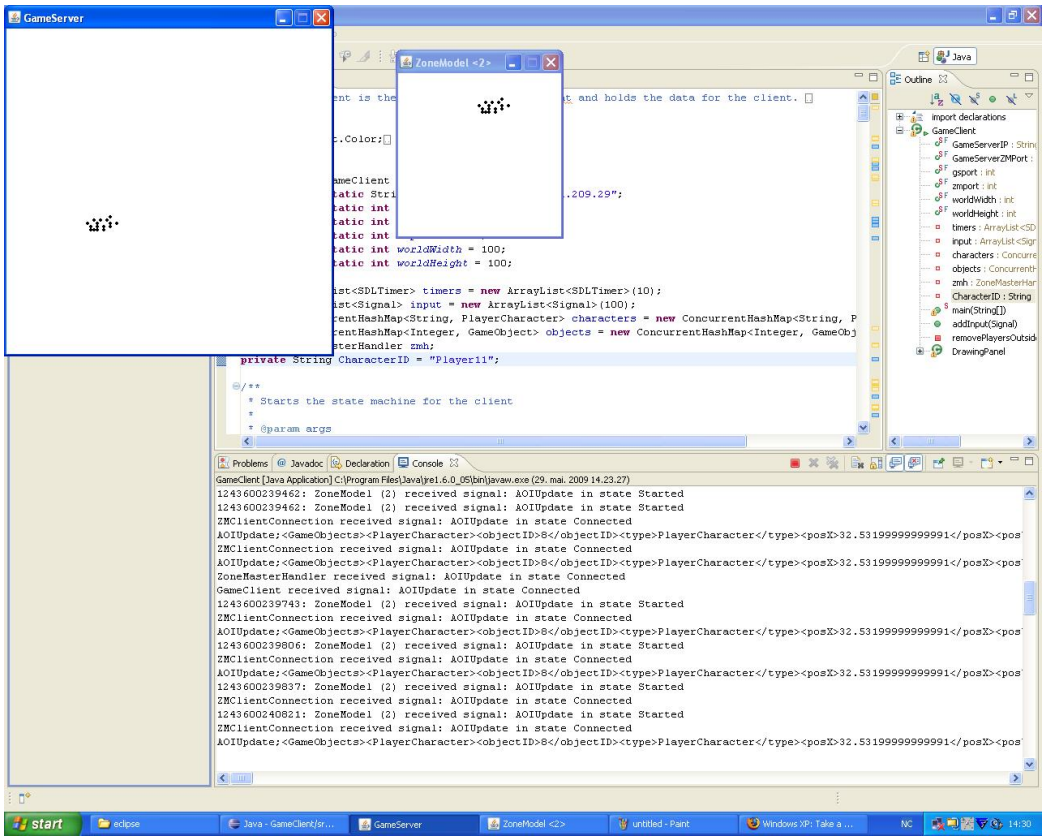


Figure 6.1: Screenshot of clients gathered in the top buffer of zone 2 for test 1e)

moving player's AOI is doubled. This may indicate that the results from scenario 1d were affected by some inner processes of the system, since it indicated a lower performance level when 20 clients were logged in.

Scenario:
1f) 20 clients connected, and all 20 in the moving clients AOI when it entered the new zone. NewZoneMaster to last ObjectAdd.

Result:

Scenario 1f shows an average processing time of 2555 ms, which is an increase of 134 ms from scenario 1c which included just 10 clients. There is also a noticeable increase in the time, 500 ms, between switching zone with

	1	2	3	4	5
1f	2561ms	2515ms	2561ms	2624ms	2514ms

Table 6.6: Test result scenario 1f

20 clients affecting the AOI, scenario 1e, and add all objects affecting the moving player's AOI.

2. Measure delay when a zone master disconnects
Reason for test:
With a player switching zone in different scenarios above, the other main interest to test with the P2P concept will be the experienced delay from a clients point of view, when a zone master disconnects and the set up of a new zone master gets initiated. This will give a concrete response how well the system handles switching of zone masters with different amount of clients connected, and can indicate where to put an effort to make future improvements.

How to perform test:
This was tested by disconnecting the client that managed a specific zone. Delay was measured from the client received NewZoneMaster to it received ConnectZMOK. This describes the perceived delay the client experiences when its zone master it swapped.

Scenario:
2a) 1 client in the zone.

Result:

	1	2	3	4	5
2a	2062ms	2062ms	2061ms	2062ms	2046ms

Table 6.7: Test result scenario 2a

The average time to set up a new zone master with one client in the zone was 2059 ms. The variation between the different results from this scenario is at most 16 ms, and shows a consistent performance level.

Scenario:
2b) 10 clients in the zone.

Result:

	1	2	3	4	5
2b	2061ms	2062ms	2061ms	2062ms	2062ms

Table 6.8: Test result scenario 2b

With 10 clients connected to the zone that experiences a zone master switch, the average processing time to set up a new zone master is 2062 ms. The results from this test scenario show almost no variation internally and from test scenario 2a, and thus strongly indicate a consistent performance level.

Scenario: 2c) 20 clients in the zone.

Result:

	1	2	3	4	5
2c	2062ms	2062ms	2046ms	2062ms	2077ms

Table 6.9: Test result scenario 2b

The test initiating a zone master switch with 20 clients connected, has an average processing time of 2062 ms. It contains some internal variations, but the total average is very close to scenario 2a and b. So when comparing the three scenarios, the system shows no change in performance level independently of the stress level put on it.

Main results from all scenarios:

The first scenario shows that the delay from the time the client receives NewZoneMaster until it receives ConnectZMOK, isn't affected by neither the amount of clients connected nor the other clients in the clients AOI when it enters a new zone. When measuring the delay until all objects in

the clients AOI are received (last ObjectAdd), the results show differences based on number of players in the AOI when the new zone is entered. For test 1c (10 clients) the average delay was 2421ms, for 1f (20 clients) the average increased to 2555ms. An increase of 134ms.

The second test, when disconnecting zonemasters, shows that there is no relation between the delay the clients perceive when a new zonemaster is instated and the number of clients in that zone.

6.2 Evaluation and analysis

6.2.1 Performance

With regards to performance, our tests showed a constant delay of approximately 2000ms. This delay is experienced in all clients, regardless of the amount of stress put on the system. This is an unacceptable level for a real time MMORPG. But the base delay is likely to be a result of the implementation of our state machine run-time system, which causes too much delay internally in each thread. Using a better framework for running the state machines would likely provide increased performance. Also an improved signalling system between the state machines would likely increase performance. As for transferring objects to a client, this delay increases with the number of objects a client receives, potentially giving the client a high delay in highly populated zones in a game. This, however, would likely be a problem in any topology, including client-server, and the design of the game world should take into consideration these issues to avoid high density populations of players. The tests performed are limited due to the limitations of the test environment, and the nature of the concept. With only a small computer lab available on a local network, the environment is not comparable to a normal game setup, usually consisting of a DSL link, and a computer with different specifications and players spread geographically over a large area, causing high delays. The nature of the concept, in lacking authoritative properties and not performing 3D rendering reduces the stress of the clients dramatically, providing them with more resources to manage zones than what would be the case with a full game, both with regards to CPU and bandwidth (due to simple game

objects). A number of options exist, however, to increase performance: Reduction of overhead by replacing XML with a proprietary format for messages, and using UDP for data transfer would reduce overhead on the network. Reducing the detail of proxies can also reduce traffic between zonemasters.

6.2.2 Scalability

This system scales very effectively due to its decentralized nature. Allowing for multiple realms, by keeping multiple gameservers. We did not have the proper equipment to put a high amount (1000+) of connections on the system, making us unable to conclude on how many connections the system can handle. Also the issues with no authoritative properties would make it hard to conclude how well a final product would scale even if we were able to stress the system to this amount. But the system allows any machine to run a zonemaster, creating several options for the game designers. Highly populated areas can be run on dedicated hardware with high speed connections to the game server, while sparsely populated areas can run on P2P. The results show that the system seems to scale well with the numbers of users put on it, when the population is well distributed over the game world.

6.2.3 Flexibility

The use of state machines provides the system with a high degree of flexibility. It provides the developers with a visual representation of the signals, which means unexpected actions are unlikely as they would be ignored by the state machines. Adding game content is done through the game engine module, which easily could be realized as a scripting engine for easy use for game designers. Flexibility is highly maintained in choice of topology for the game by allowing hybrid P2P. The designers have the options to run the entire system in a dedicated server network, or even on a single centralized server.

6.2.4 Functionality

The system consists of a set of MMO server functions. These, have under our tests performed very well. The seamless world is unnoticeable from the clients perspective except for the delay that is experienced due to the state machine run-time system. Also proxies provide the visibility of players in adjacent zones. Despite the roughly 2 second delay in the system, the dead reckoning functionality provides instant response at the client end, and world updates correct the players position. The zones in these tests were statically divided, but dynamic zones could be implemented and is supported. The mechanism for choosing zonemasters is performing well, in this test simply choosing a random client that is not already a zonemaster. Zonemasters are effectively replaced, with clients experiencing no delay except the 2 seconds.

6.2.5 Security

The conceptual demo was implemented without concern for security. Clear issues arise when clients are given the responsibility of managing zones. Main concerns are potential cheating by zonemasters, and the lack of anonymity of clients. This causes potential direct attacks on the clients computers. Anonymous P2P solutions exist, but have not been tested for this purpose.

6.2.6 Mobile Terminals

The P2P solution has not been tested on mobile terminals, but as previously mentioned, the mobile networks does not provide the necessary stability in latency for real time games, and neither does mobile devices provide the required computing power for 3D games like 'tisu. However simpler games that allow the player to develop his avatar in the main game through objectives on a mobile device would integrate easily through direct modification of the database on the game server.

6.2.7 Stability

The P2P solution developed is highly redundant. As long as the gameserver is running, each zone is managed by a new client as soon as the old managing client disconnects. Should the number of clients get too low for clients to run zones, the gameserver will have available capacity to manage all zones itself. Backups of the game state in the zone are sent regularly to the gameserver, ensuring that the game state is updated when a new client becomes a zonemaster. However, when a zonemaster disconnects abruptly, no backup is sent to the gameserver right before disconnect. This can cause loss of game state changes, and when the new zonemaster starts managing the game state in the zone will be that of the last received backup. Finding an acceptable backup interval will be a challenge for developers. An issue that can occur in a P2P solution is that when a zone contains a high number of players and the zone is to receive a new zonemaster, the new zonemaster will receive connection requests from all players in the zone at the same time. This could in the worst case act as a DDoS attack on the client, essentially blocking it and lead to a new client getting appointed to managing the zone. Leading to a string of DDoS attacks. To combat this issue, dynamic zones should be implemented. But in the case of very high density populations of players, such zones should be managed by dedicated servers as clients would likely not have the resources to manage them. Another solution could be to implement a back-off algorithm to delay the connections, but this would impose the same delay on the clients, potentially disrupting gameplay.

6.2.8 Integration

Integration with the Unity 3D engine requires development of bindings to one of Unity's supported languages, like C# or ActionScript. This is required to develop regardless of how the networking system is developed, and is how other network engines are integrated with Unity.

Chapter 7

Conclusion

7.1 Conclusion

The work done in this thesis was made on behalf of Ablemagic, the target was to create a foundation for an MMORPG networking engine that could be developed further in the future. Our main focus, as Ablemagic is a small company trying to enter a tough market, was to research a cost effective solution based on P2P to give Ablemagic an edge on its competitors.

The concept the practical part of this thesis is based on was worked out after a thorough study of the MMORPG genre, and the engineering techniques of it. It incorporates an original, cost saving architecture based on P2P, that still supports both centralized and distributed architectures. The concept could help small game developers to release titles that using the traditional thoughts of client-server would require huge investments in bandwidth lease and server hardware, as well as maintenance of this.

Further, the thesis involved designing a network system for the MMORPG 'tisu. This design was based on the concept of hybrid P2P that is described throughout this work. Using SDL as the design tool made the design flexible and separated it from the implementation. This will allow developers to utilize the design in future implementations, regardless of the programming language chosen and the platform developed on. During research and design of this system, it became clear that implementing all desired features in a demo would be impossible due to the complexity of the functionality and limited time for this work. However, the design supports implementation of remaining features in the future. With 'tisu being in early development stages, and no absolute requirements for functionality in the game, the requirements for the system needed to be general and based on similar games. This required a general design, with multiple options. The proposed design should be able to support any design choices made.

The implementation of the design incorporated many of the features of a modern MMORPG. Zoning with seamless worlds, synchronization of movements and objects. A simple state machine run-time framework was implemented, and the processes from the design were implemented for signalling. The implementation worked well, with some unexpected errors occurring randomly, but overall performing excellent. The implementation proves the stability and flexibility of the design, and that the concept proposed works in practice. In our latency tests, however, we noticed a

constant delay of roughly 2 seconds. This is unacceptable for any game, but is not a flaw in the design. The framework for running the state machines is largely based on actively checking the input queues with a minor delay for every loop rotation, which caused on of the processor cores to work at 100%. It is most likely that this framework is causing the delay. This issue should be solvable by improving the framework or utilizing an existing state machine API.

As a final conclusion, the work done in this thesis should provide Ablemagic with knowledge on the subject of engineering network systems for MMORPGs. As well as provide a design, and proof-of-concept that could be utilized in future research or as a base for development of 'tisu as it stands today.

7.1.1 Future work

This study has revealed that P2P solutions for MMORPGs have a promising potential, especially with the benefits it provides small developers. But many challenges exist that needs more research before a solution is usable in an end product.

A more complete solution than the one developed in this thesis should be tested, and it should include both authoritative server properties and 3D processing on the clients to gain better insight on how a more realistic solution scales and performs. Solutions for managing high density populations of players without using dedicated servers are also an area of interest. Research into effective zone dividing and load sharing solutions has shown great performance increase [8], and could be used in P2P solutions. This area should be further researched in combination with P2P.

Security issues also require further work. Some work has been done to develop cheat-proof protocols for online games [3].
However, to secure the solution enough for commercial use the following areas require research:

- Evaluate the use of anonymous P2P solutions to protect clients from direct attacks.

- Techniques to better hide the mechanism of a program from the users.
- Combine the above with strong encryption schemes to avoid man-in-the-middle attacks.

Bibliography

[1] Thor Alexander. Massively multiplayer game development, November 2003.

[2] Richard Bartle. Early mud history. http://www.mud.co.uk/richard/mudhist.htm, 1999.

[3] Nathaniel E. Baughman. Cheat-proof playout for centralized and distributed online games. `http://prisms.cs.umass.edu/brian/pubs/baughman.infocom01.pdf`, 2001.

[4] Mary Bellis. Inventors of the modern computer. `http://inventors.about.com/library/weekly/aa090198.htm`.

[5] Rolv Braek. Sdl basics. `http://www.cs.tut.fi/kurssit/TLT-9806/RolvBraekSDL.pdf`.

[6] Rolv Braek and Oeystein Haugen. Engineering real time systems an object oriented methodology using sdl.

[7] Frank Caron. Age of conan server merges more than half of server list. `http://arstechnica.com/gaming/news/2009/01/age-of-conan-server-mergers-more-than-half-of-server-list.ars`, 2009.

[8] Jin Chen. Locality aware dynamic load management for massively multiplayer games. `http://www.cs.toronto.edu/~jinchen/ppopp.pdf`, 2005.

[9] Blizzard Entertainment. World of warcraft® subscriber base reaches 11.5 million worldwide. `http://eu.blizzard.com/en/press/081223.html`, 2008.

[10] Ericsson. The evolution of edge. http://www.ericsson.com/technology/whitepapers/3107_The_evolution_of_EDGE_A.pdf, 2007.

[11] Funcom. Age of conan reaches one million milestone. http://www.funcom.com/wsp/funcom/frontend.cgi?func=publish.show&func_id=1301&table=CONTENT&item=1004, 2008.

[12] Vivendi Games. Vivendi investor presentation, 2006.

[13] gotoAndPlay(). http://www.smartfoxserver.com/.

[14] Harri Holma. 3 gpp release 5 hsdpa measurements. http://ieeexplore.ieee.org/stamp/stamp.jsp?arnumber=04022310, 2006.

[15] PX Interactive. http://www.netdognetworks.com/.

[16] Björn Knutsson, Honghui Lu, Wei Xu, and Bryan Hopkins. Peer-to-peer support for massively multiplayer games, 2004.

[17] Jenkins Software LLC. http://www.jenkinssoftware.com/.

[18] Stein Jarle Olsen. 2 av 3 tilbake til wow. http://www.tu.no/it/article188183.ece, 2008.

[19] Stein Jarle Olsen. Trøbbel for conan. http://www.tu.no/it/article174004.ece, 2008.

[20] Eve Online. Eve online launches largest supercomputer in the gaming industry running on ibm server technology. http://www.eveonline.com/pressreleases/default.asp?pressReleaseID=25, 2006.

[21] Princeton. http://wordnetweb.princeton.edu/perl/webwn?s=realm.

[22] Gregor Schiele, Richard Süselbeck, Arno Wacker, Jörg Hähner, Christan Backer, and Torben Weis. Requirements of peer-to-peer-based massively multiplayer online gaming, 2007.

[23] Inc. Sun Microsystems. http://projectdarkstar.com/.

[24] Philipp Svoboda. Traffic analysis and modeling for world of warcraft. http://publik.tuwien.ac.at/files/pub-et_12119.pdf, 2007.

[25] Telenor. Dekningskart. http://www.telenor.no/bedrift/produkter/mobil/dekning/dekningskart-mobildekning/.

[26] Tonio Triebel, Benjamin Guthier, Richard Süselbeck, Gregor Schiele, and Wolfgang Effelsberg. Peer-to-peer infrastructures for games, 2008.

[27] Unity. Unity iphone 1.0.2. http://unity3d.com/unity/whats-new/iphone-1.0.2.html.

[28] WoW Wiki. Realms list. http://www.wowwiki.com/Realms_list, 2009.

[29] Wikipedia. http://en.wikipedia.org/wiki/Massively_multiplayer_online_game#MMO_role-playing_game.

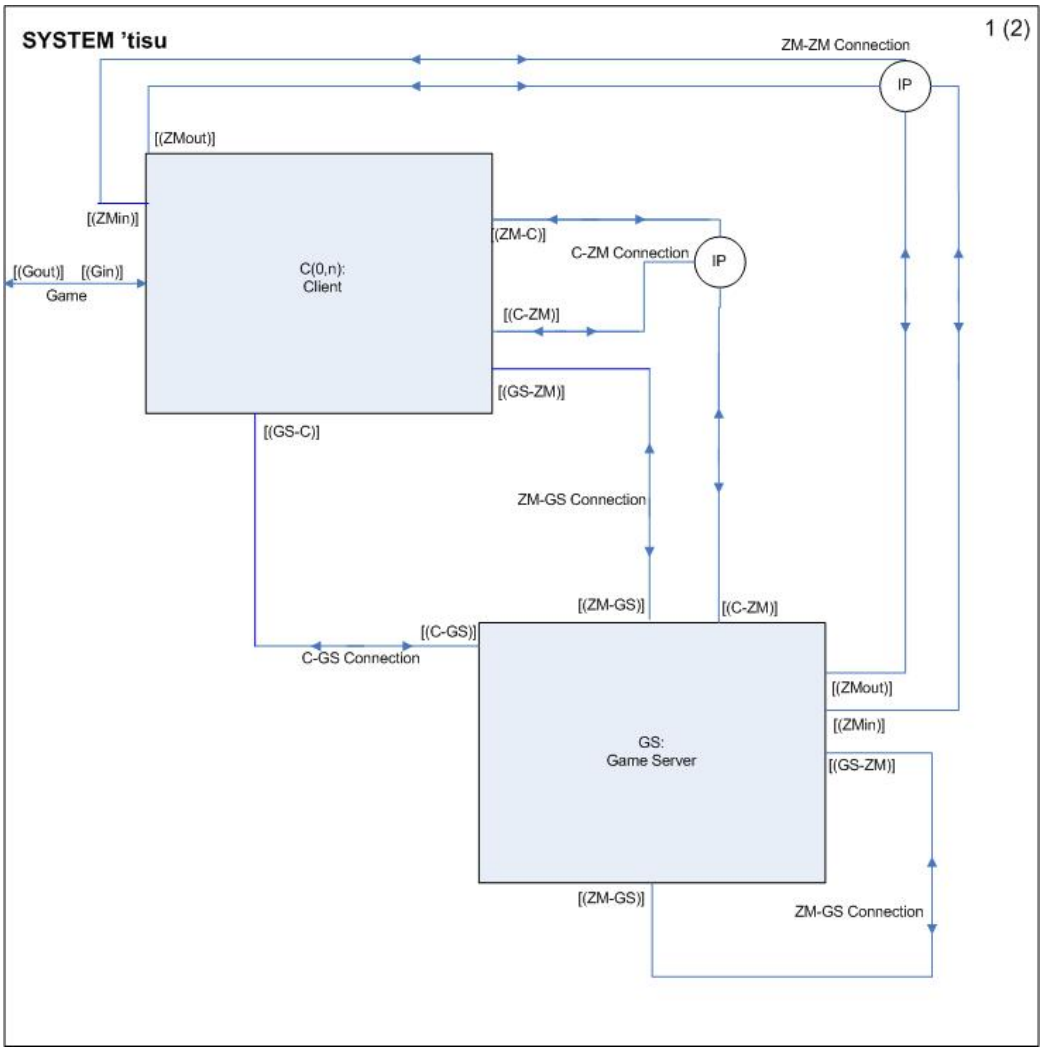
[30] Wikipedia. http://en.wikipedia.org/wiki/MMORPG.

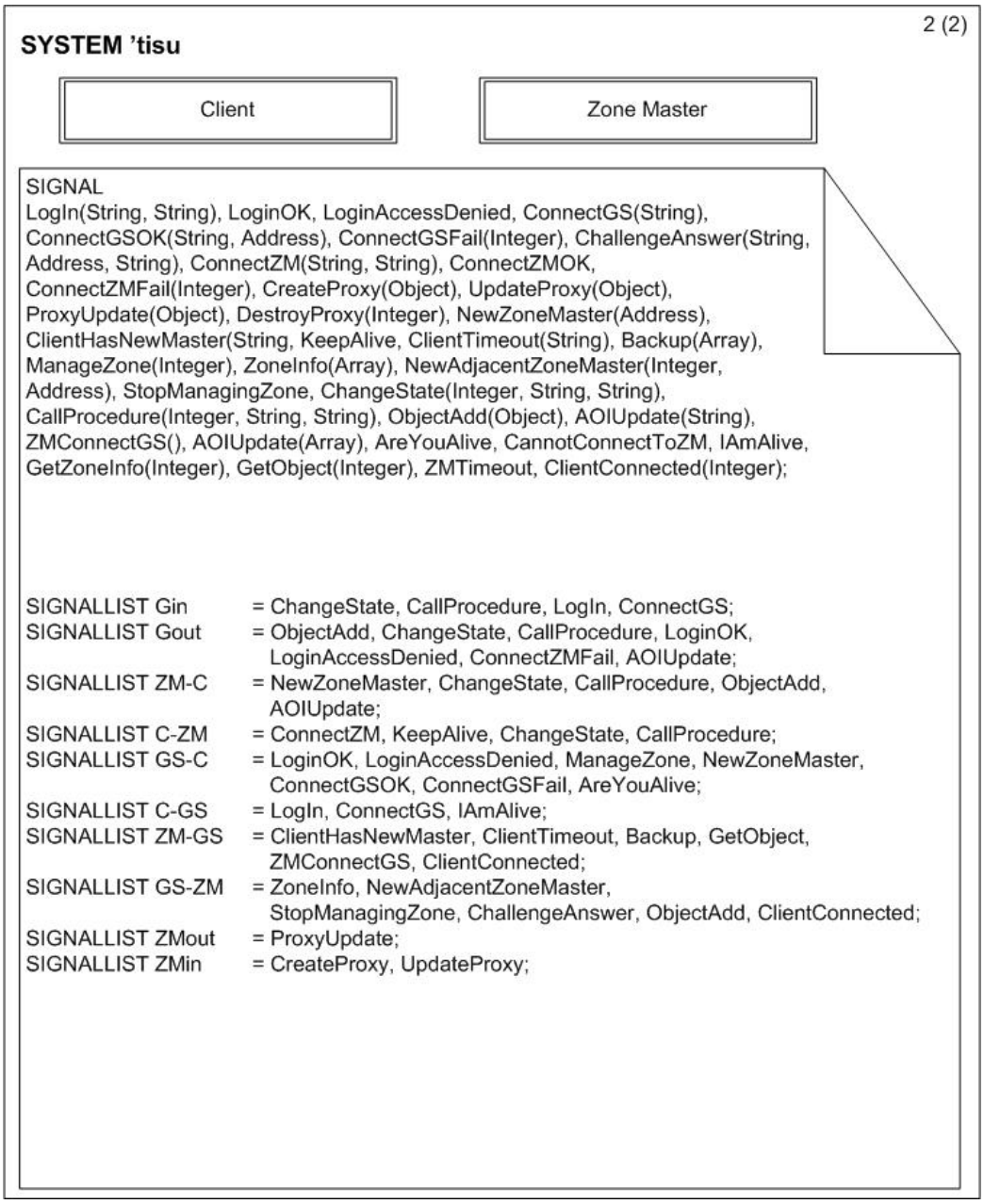
[31] Wikipedia. Avatar (computing). http://en.wikipedia.org/wiki/Avatar_(computing).

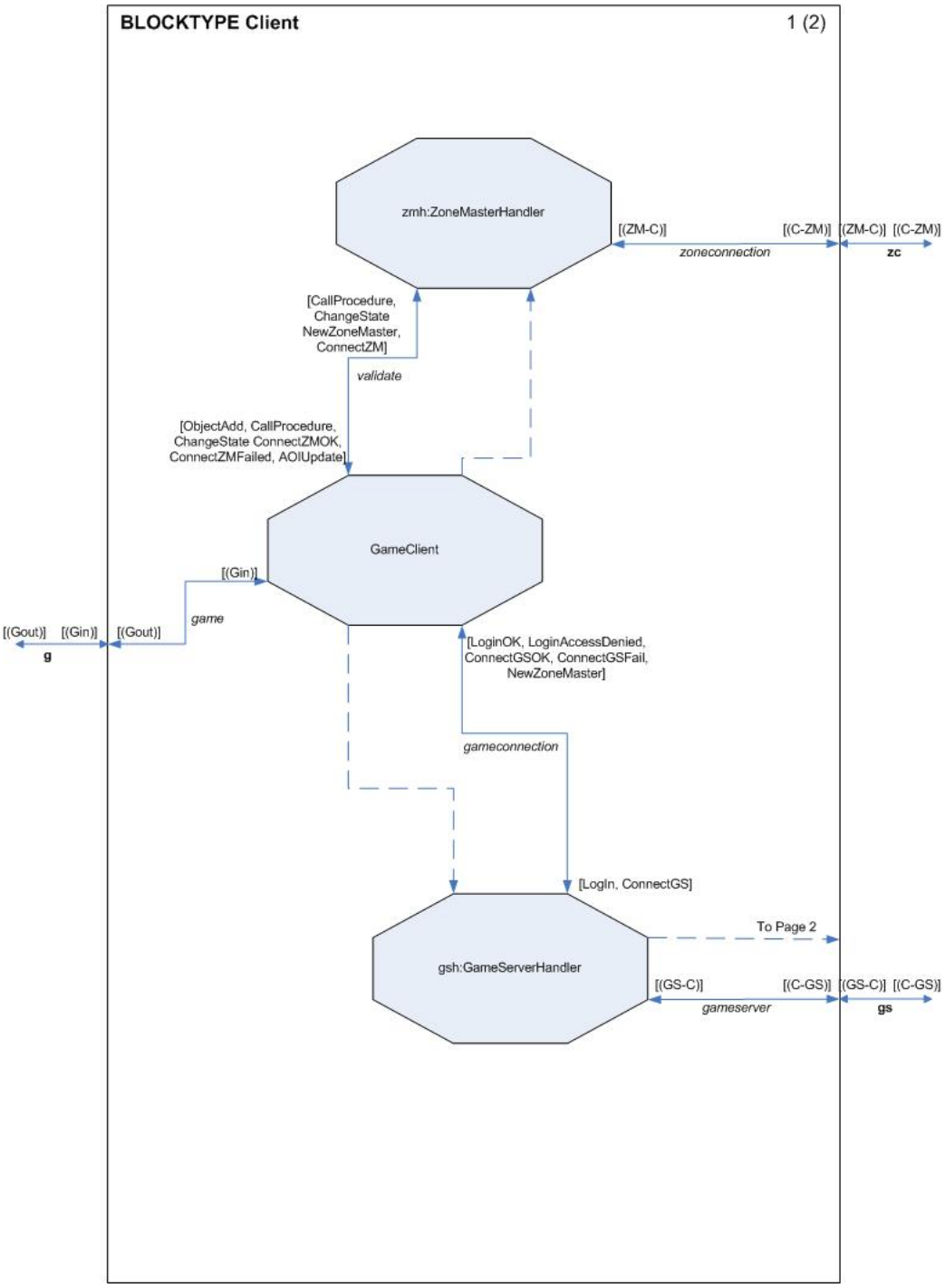
[32] Wikipedia. History of massively multiplayer online games. http://en.wikipedia.org/wiki/History_of_massively_multiplayer_online_role-playing_games.

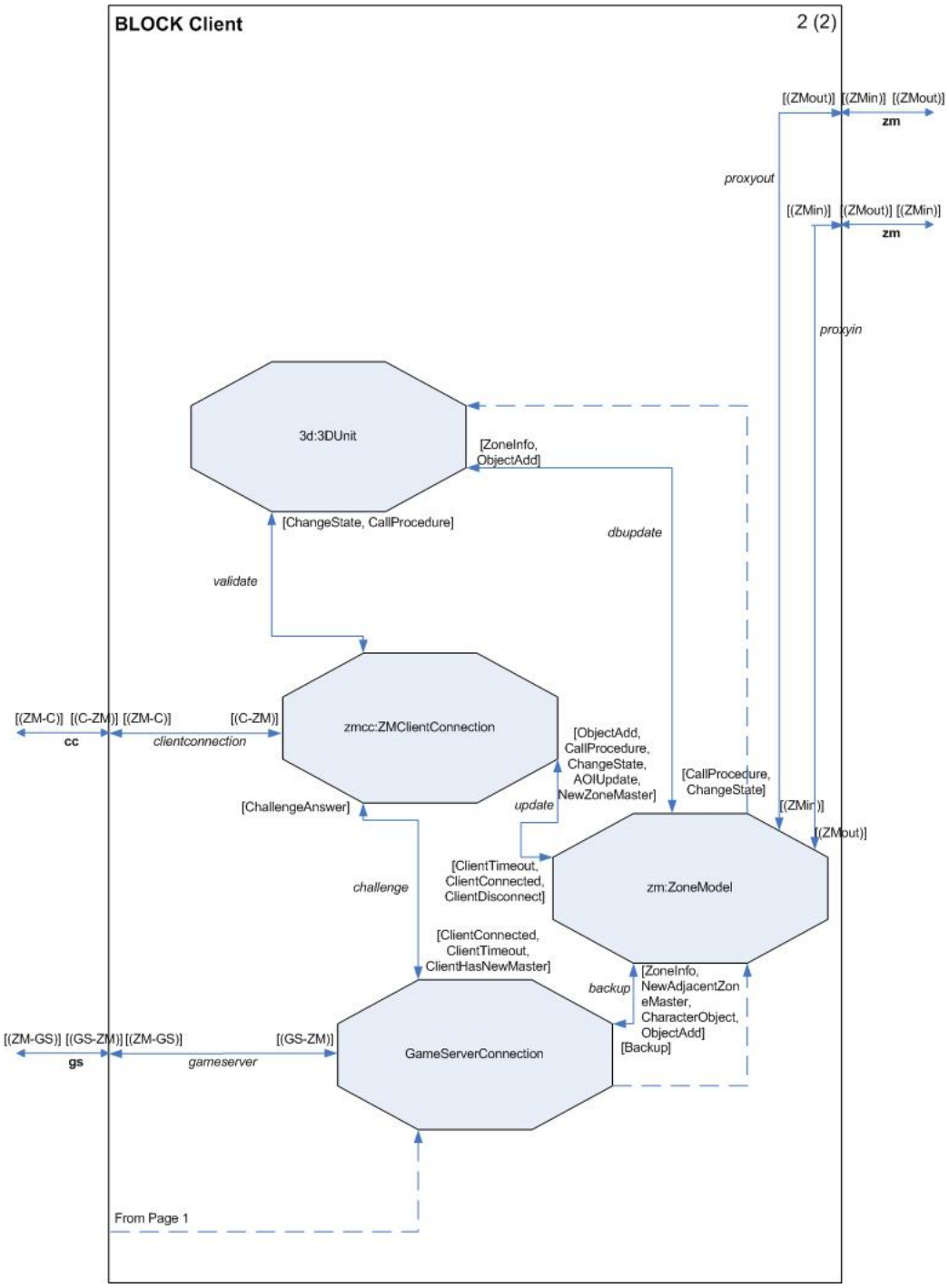
Appendix A

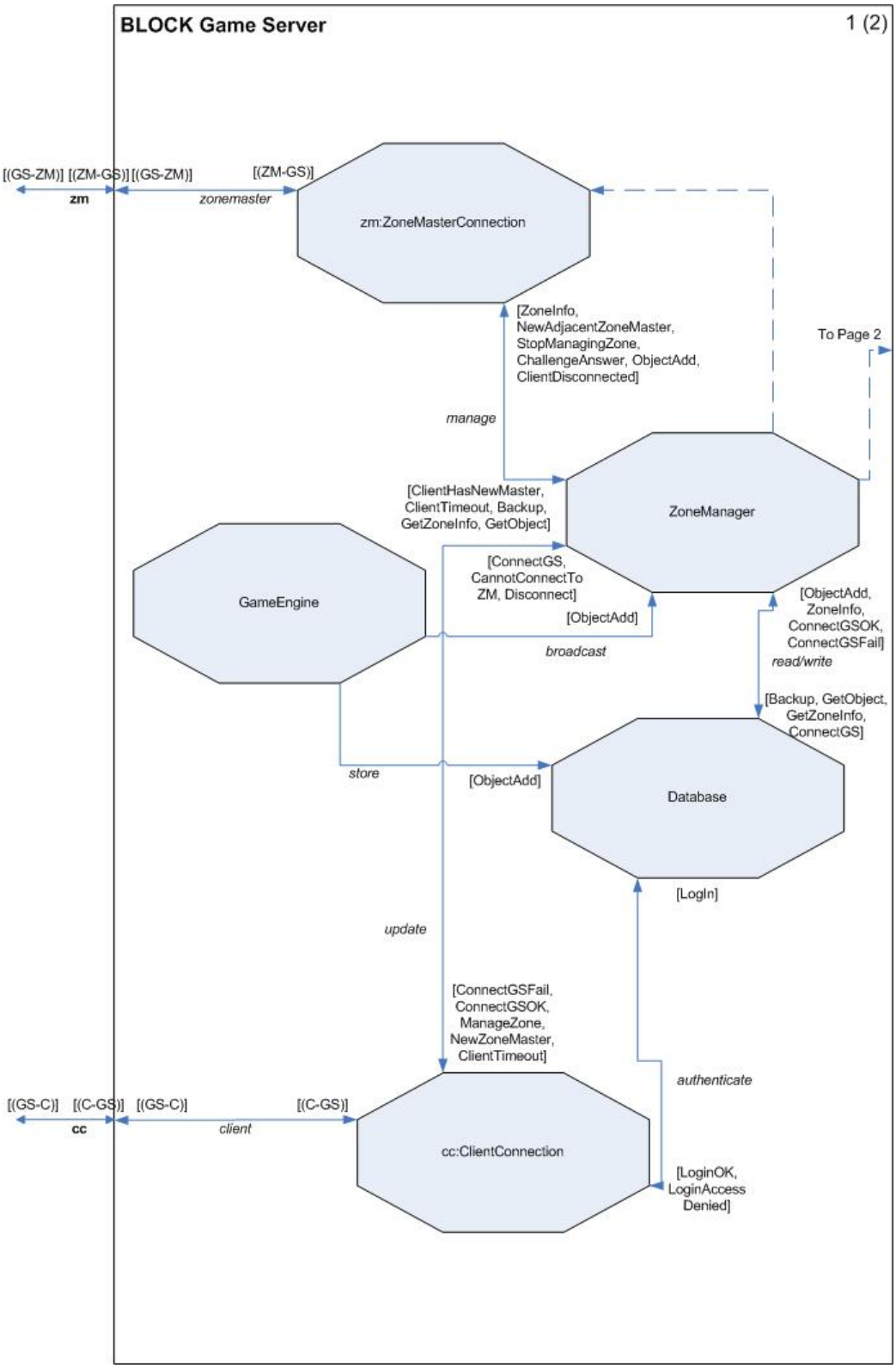
SDL Design of the 'tisu system

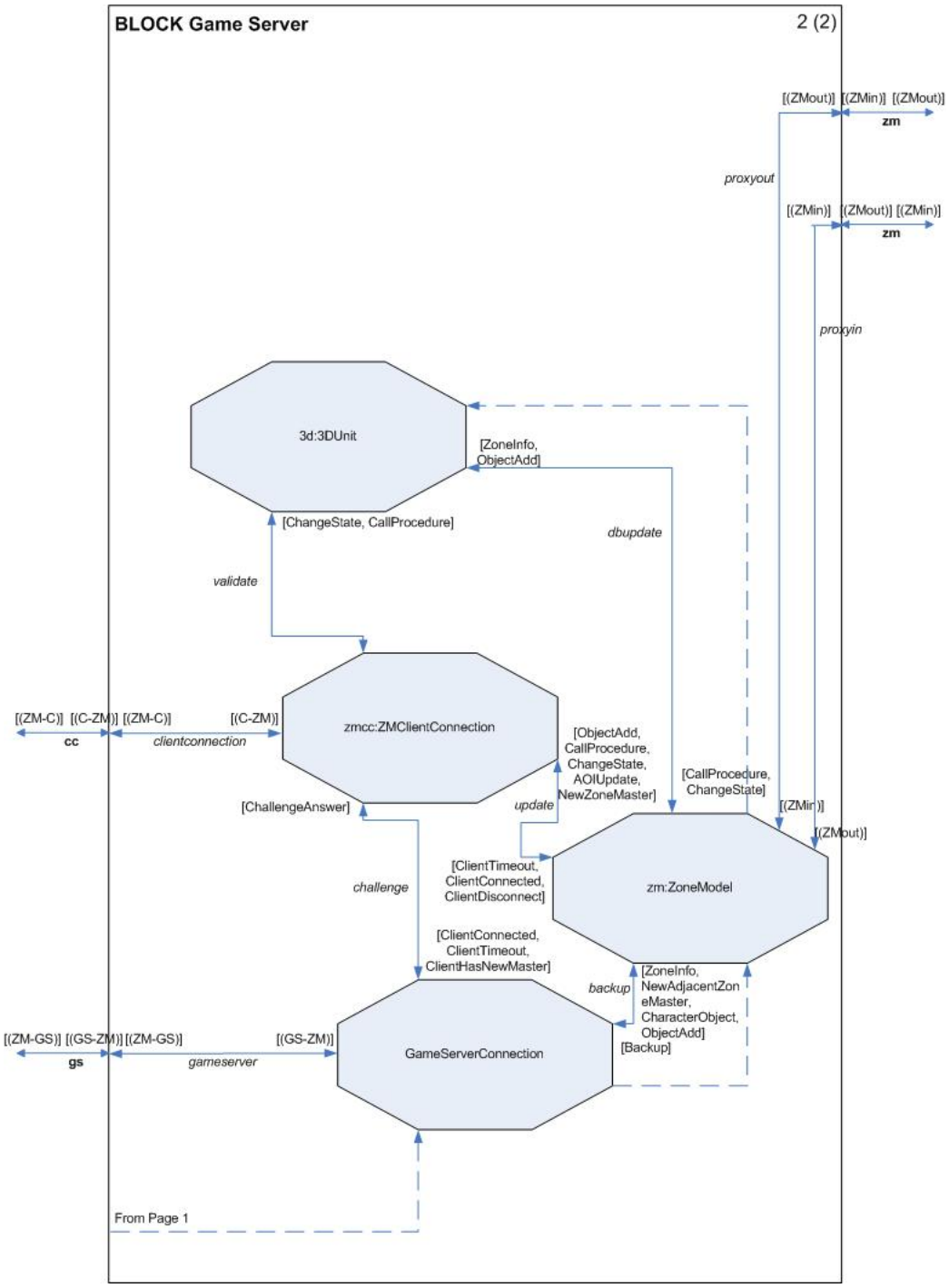

SYSTEM 'tisu
1 (2)
ZM-ZM Connection
IP
[(ZMout)]
[(ZMin)]
[(ZM-C)]
C-ZM Connection
IP
[(Gout)]
[(Gin)]
Game
C(0,n):
Client
[(C-ZM)]
[(GS-ZM)]
[(GS-C)]
ZM-GS Connection
[(ZM-GS)]
[(C-ZM)]
[(C-GS)]
C-GS Connection
[(ZMout)]
[(ZMin)]
GS:
Game Server
[(GS-ZM)]
[(ZM-GS)]
ZM-GS Connection

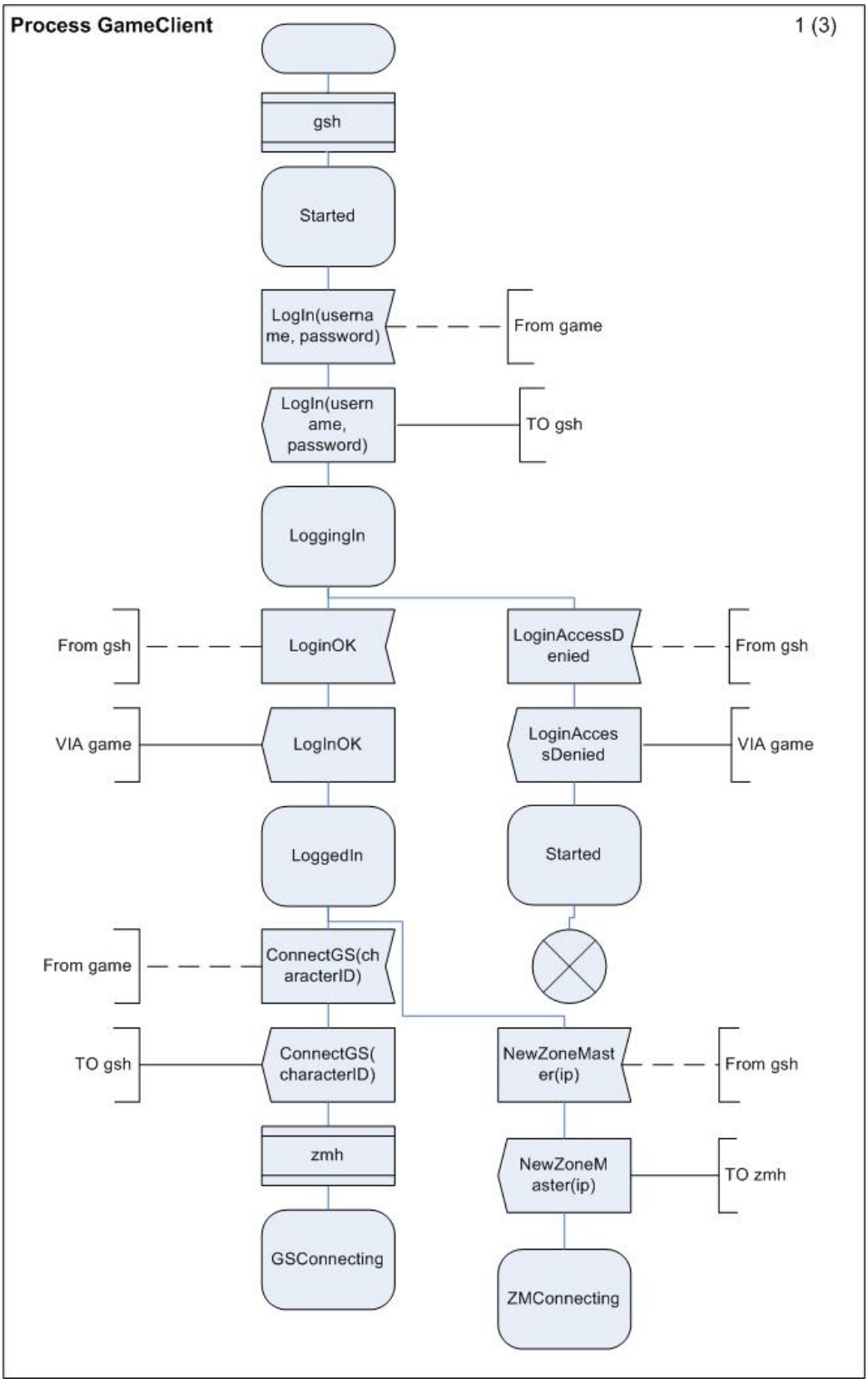

2 (2)
SYSTEM 'tisu
Client
Zone Master
SIGNAL
LogIn(String, String), LoginOK, LoginAccessDenied, ConnectGS(String),
ConnectGSOK(String, Address), ConnectGSFail(Integer), ChallengeAnswer(String,
Address, String), ConnectZM(String, String), ConnectZMOK,
ConnectZMFail(Integer), CreateProxy(Object), UpdateProxy(Object),
ProxyUpdate(Object), DestroyProxy(Integer), NewZoneMaster(Address),
ClientHasNewMaster(String, KeepAlive, ClientTimeout(String), Backup(Array),
ManageZone(Integer), ZoneInfo(Array), NewAdjacentZoneMaster(Integer,
Address), StopManagingZone, ChangeState(Integer, String, String),
CallProcedure(Integer, String, String), ObjectAdd(Object), AOIUpdate(String),
ZMConnectGS(), AOIUpdate(Array), AreYouAlive, CannotConnectToZM, IAmAlive,
GetZoneInfo(Integer), GetObject(Integer), ZMTimeout, ClientConnected(Integer);
SIGNALLIST Gin = ChangeState, CallProcedure, LogIn, ConnectGS;
SIGNALLIST Gout = ObjectAdd, ChangeState, CallProcedure, LoginOK,
LoginAccessDenied, ConnectZMFail, AOIUpdate;
SIGNALLIST ZM-C = NewZoneMaster, ChangeState, CallProcedure, ObjectAdd,
AOIUpdate;
SIGNALLIST C-ZM = ConnectZM, KeepAlive, ChangeState, CallProcedure;
SIGNALLIST GS-C = LoginOK, LoginAccessDenied, ManageZone, NewZoneMaster,
ConnectGSOK, ConnectGSFail, AreYouAlive;
SIGNALLIST C-GS = LogIn, ConnectGS, IAmAlive;
SIGNALLIST ZM-GS = ClientHasNewMaster, ClientTimeout, Backup, GetObject,
ZMConnectGS, ClientConnected;
SIGNALLIST GS-ZM = ZoneInfo, NewAdjacentZoneMaster,
StopManagingZone, ChallengeAnswer, ObjectAdd, ClientConnected;
SIGNALLIST ZMout = ProxyUpdate;
SIGNALLIST ZMin = CreateProxy, UpdateProxy;

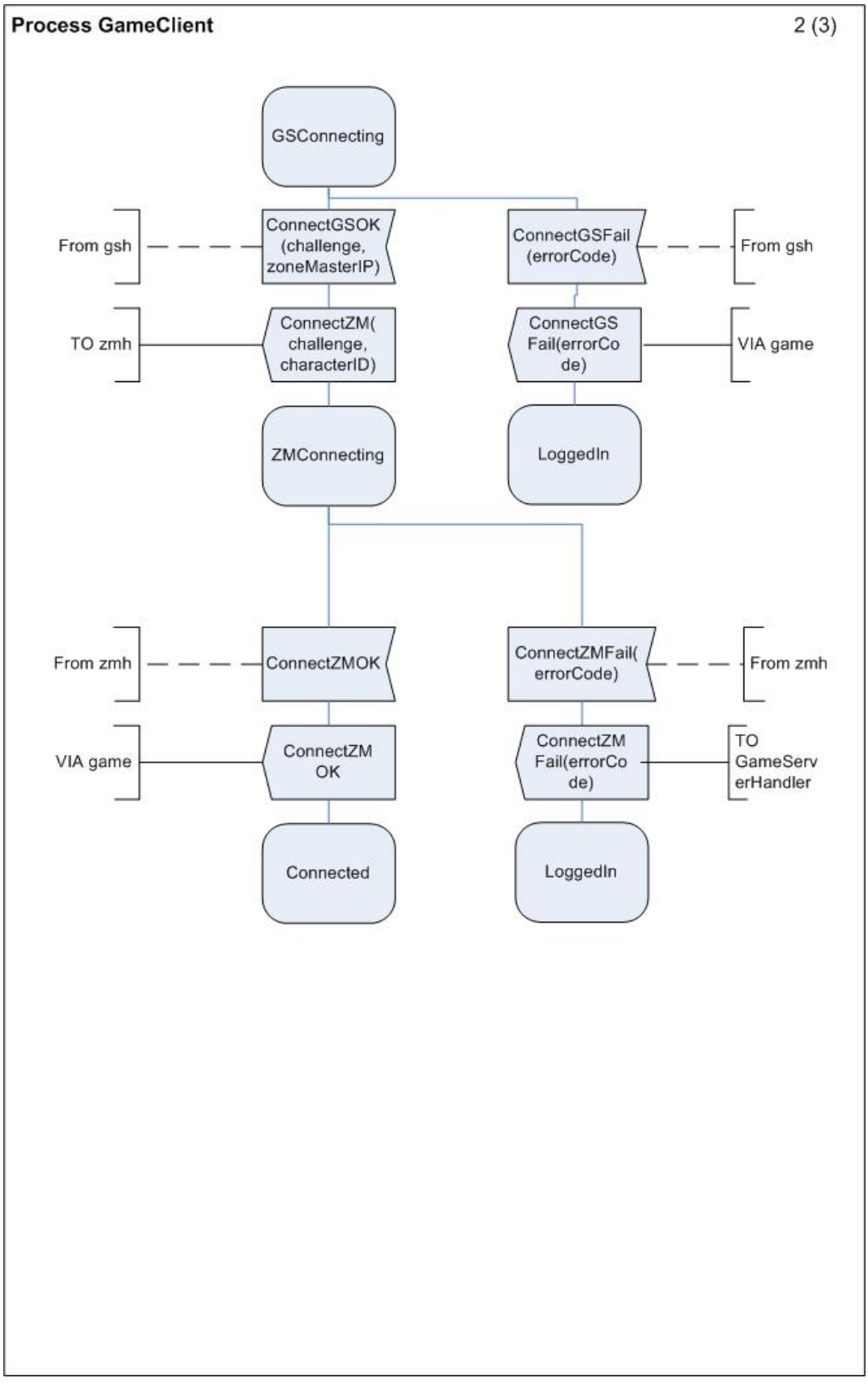

BLOCKTYPE Client
1 (2)
zmh:ZoneMasterHandler
[(ZM-C)]
zoneconnection
[(C-ZM)]
[(ZM-C)]
[(C-ZM)]
zc
[CallProcedure, ChangeState NewZoneMaster, ConnectZM]
validate
[ObjectAdd, CallProcedure, ChangeState ConnectZMOK, ConnectZMFailed, AOIUpdate]
GameClient
[(Gin)]
game
[(Gout)]
[(Gin)]
[(Gout)]
g
[LoginOK, LoginAccessDenied, ConnectGSOK, ConnectGSFail, NewZoneMaster]
gameconnection
[LogIn, ConnectGS]
To Page 2
gsh:GameServerHandler
[(GS-C)]
gameserver
[(C-GS)]
[(GS-C)]
[(C-GS)]
gs

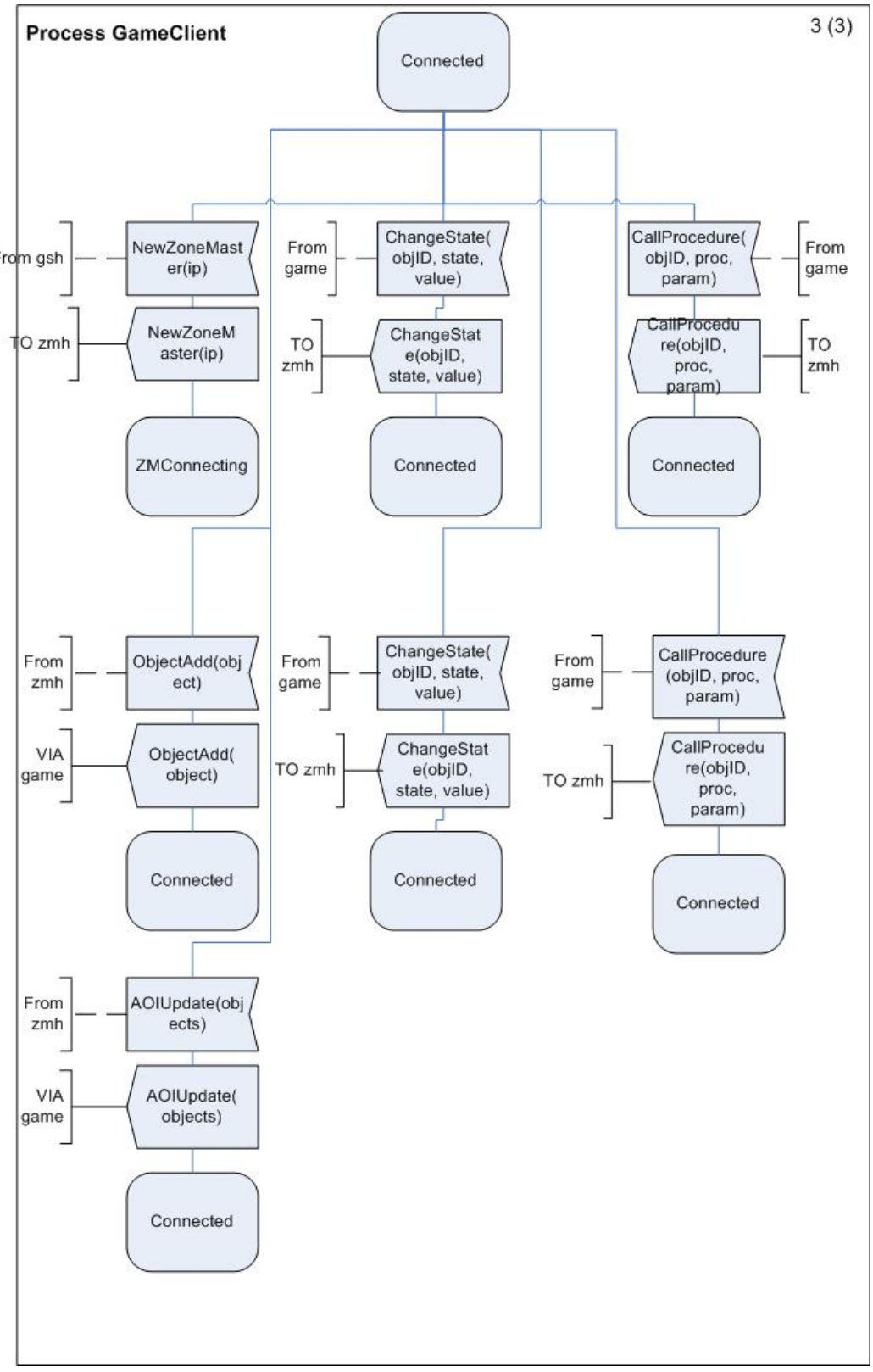

BLOCK Client
2 (2)
[(ZMout)] [(ZMin)] [(ZMout)]
zm
proxyout
[(ZMin)] [(ZMout)] [(ZMin)]
zm
proxyin
3d:3DUnit
[ZoneInfo, ObjectAdd]
[ChangeState, CallProcedure]
dbupdate
validate
[(ZM-C)] [(C-ZM)] [(ZM-C)]
cc
clientconnection
[(C-ZM)]
zmcc:ZMClientConnection
[ObjectAdd, CallProcedure, ChangeState, AOIUpdate, NewZoneMaster]
[CallProcedure, ChangeState]
[(ZMin)]
[(ZMout)]
[ChallengeAnswer]
update
[ClientTimeout, ClientConnected, ClientDisconnect]
zm:ZoneModel
challenge
[ClientConnected, ClientTimeout, ClientHasNewMaster]
backup
[ZoneInfo, NewAdjacentZoneMaster, CharacterObject, ObjectAdd]
[Backup]
[(ZM-GS)] [(GS-ZM)] [(ZM-GS)]
gs
gameserver
[(GS-ZM)]
GameServerConnection
From Page 1

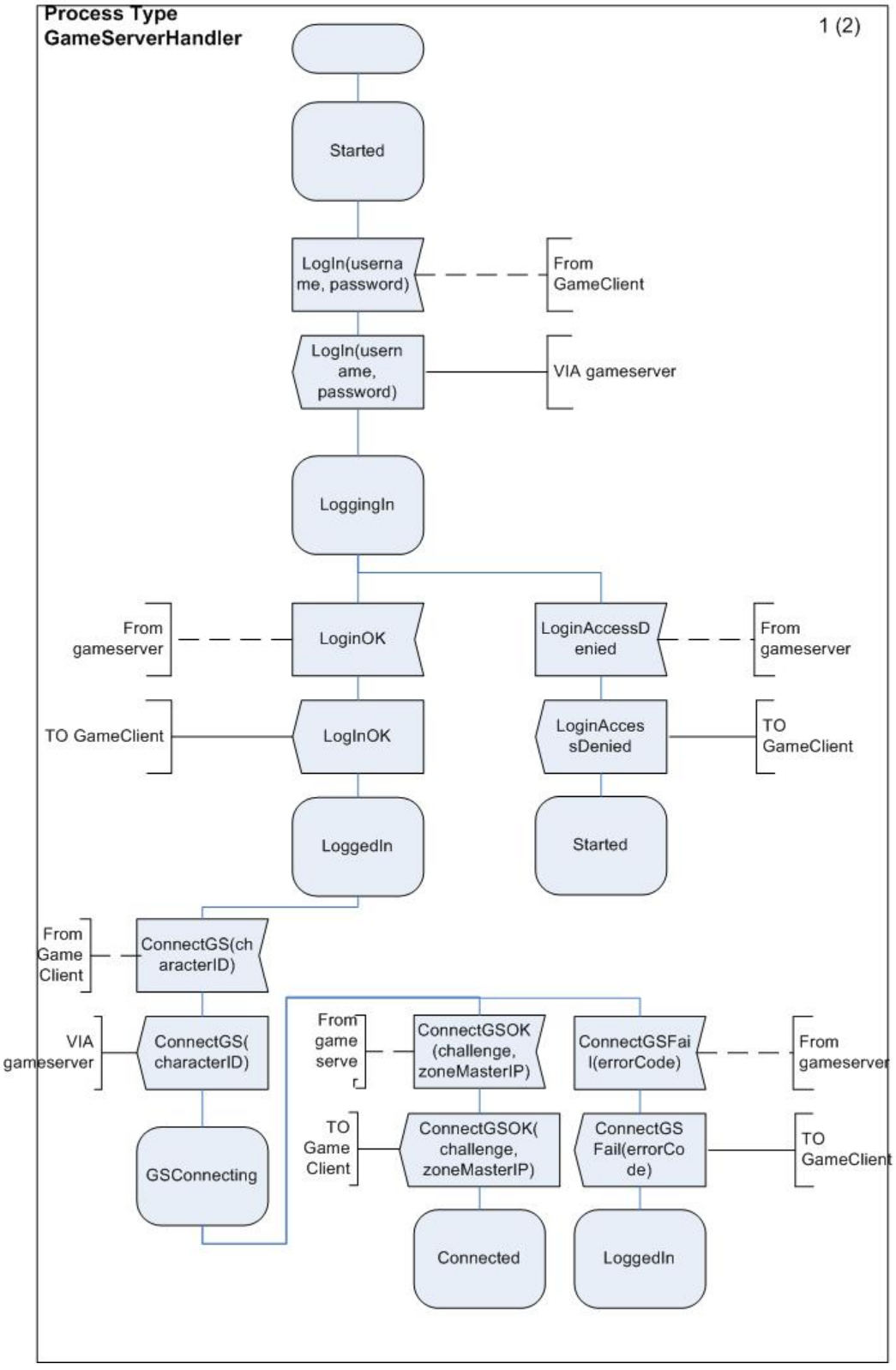

BLOCK Game Server
1 (2)
[(GS-ZM)] [(ZM-GS)]
zm
[(GS-ZM)]
zonemaster
[(ZM-GS)]
zm:ZoneMasterConnection
[ZoneInfo, NewAdjacentZoneMaster, StopManagingZone, ChallengeAnswer, ObjectAdd, ClientDisconnected]
To Page 2
manage
[ClientHasNewMaster, ClientTimeout, Backup, GetZoneInfo, GetObject]
ZoneManager
GameEngine
[ConnectGS, CannotConnectTo ZM, Disconnect]
[ObjectAdd]
broadcast
[ObjectAdd, ZoneInfo, ConnectGSOK, ConnectGSFail]
read/write
[Backup, GetObject, GetZoneInfo, ConnectGS]
store
[ObjectAdd]
Database
[LogIn]
update
[ConnectGSFail, ConnectGSOK, ManageZone, NewZoneMaster, ClientTimeout]
authenticate
[(GS-C)] [(C-GS)]
cc
[(GS-C)]
client
[(C-GS)]
cc:ClientConnection
[LoginOK, LoginAccess Denied]

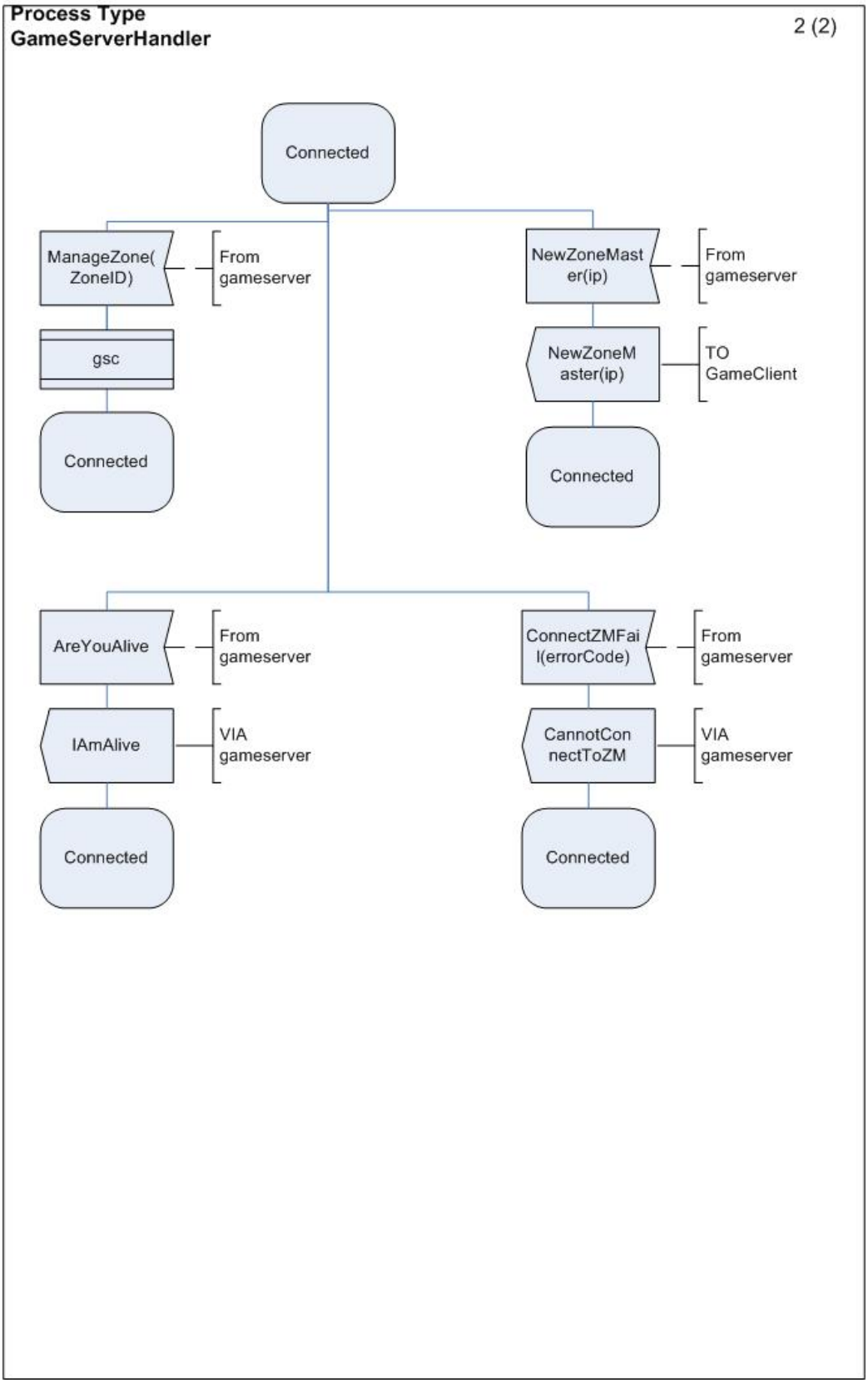

BLOCK Game Server
2 (2)
[(ZMout)] [(ZMin)] [(ZMout)]
zm
proxyout
[(ZMin)] [(ZMout)] [(ZMin)]
zm
proxyin
3d:3DUnit
[ZoneInfo, ObjectAdd]
[ChangeState, CallProcedure]
dbupdate
validate
[(ZM-C)] [(C-ZM)] [(ZM-C)]
cc
clientconnection
[(C-ZM)]
zmcc:ZMClientConnection
[ObjectAdd, CallProcedure, ChangeState, AOIUpdate, NewZoneMaster]
[CallProcedure, ChangeState]
[(ZMin)]
[(ZMout)]
[ChallengeAnswer]
update
[ClientTimeout, ClientConnected, ClientDisconnect]
zm:ZoneModel
challenge
[ClientConnected, ClientTimeout, ClientHasNewMaster]
backup
[ZoneInfo, NewAdjacentZoneMaster, CharacterObject, ObjectAdd]
[Backup]
[(ZM-GS)] [(GS-ZM)] [(ZM-GS)]
gs
gameserver
[(GS-ZM)]
GameServerConnection
From Page 1

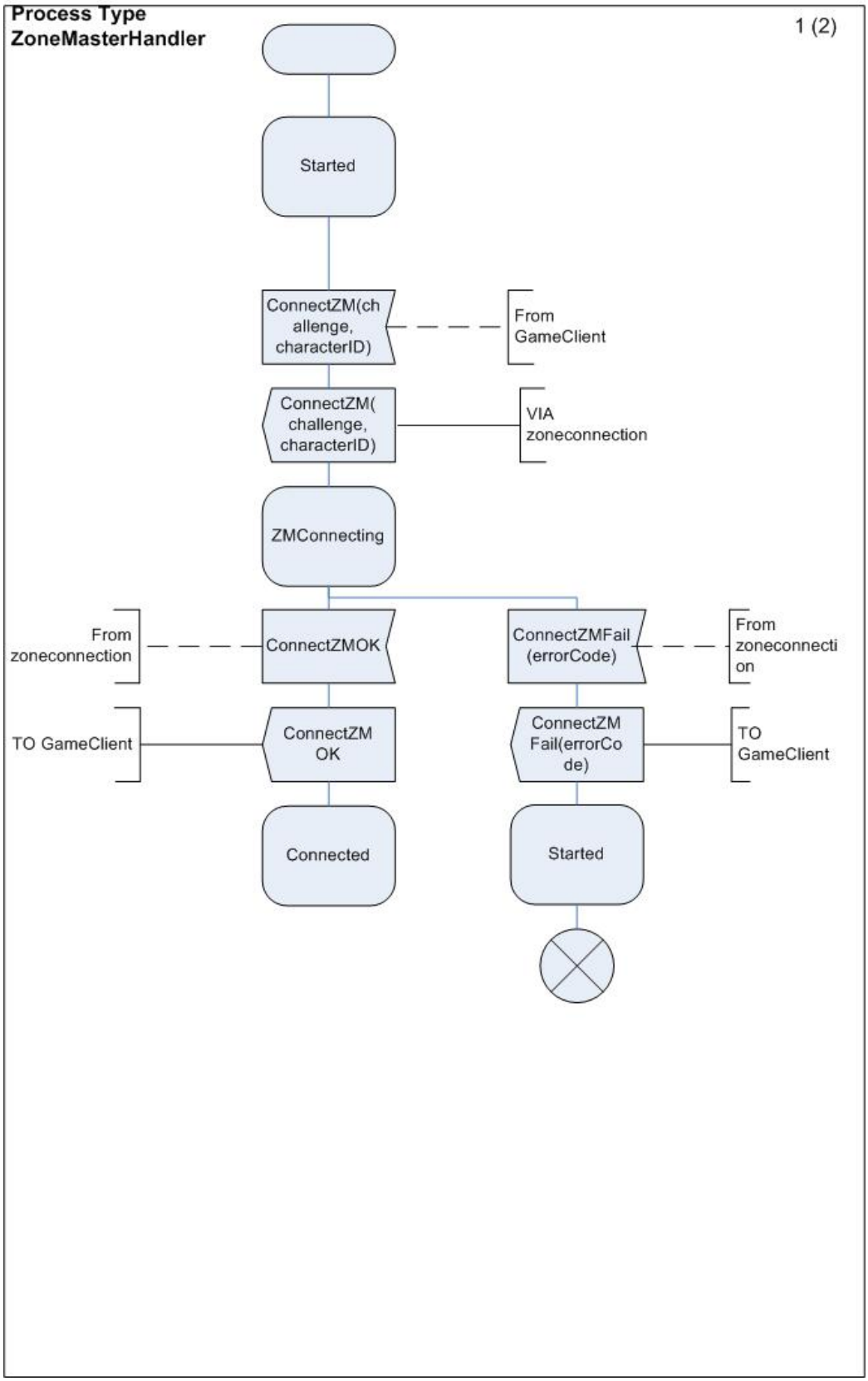

Process GameClient
1 (3)
gsh
Started
LogIn(username, password)
From game
LogIn(username, password)
TO gsh
LoggingIn
From gsh
LoginOK
LoginAccessDenied
From gsh
VIA game
LogInOK
LoginAccessDenied
VIA game
LoggedIn
Started
From game
ConnectGS(characterID)
TO gsh
ConnectGS(characterID)
NewZoneMaster(ip)
From gsh
zmh
NewZoneMaster(ip)
TO zmh
GSConnecting
ZMConnecting

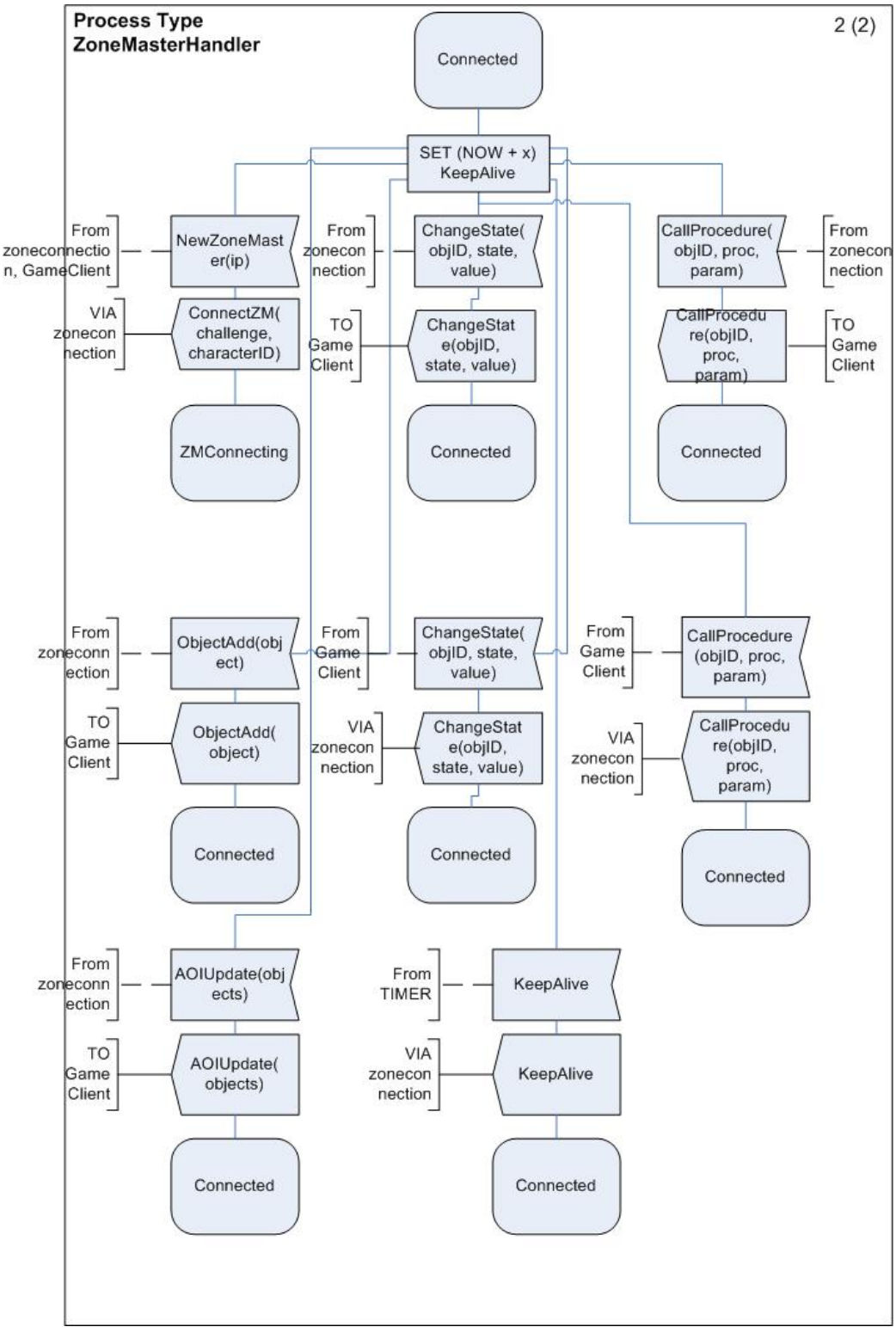

Process GameClient
2 (3)
GSConnecting
From gsh
ConnectGSOK (challenge, zoneMasterIP)
ConnectGSFail (errorCode)
From gsh
TO zmh
ConnectZM(challenge, characterID)
ConnectGS Fail(errorCo de)
VIA game
ZMConnecting
LoggedIn
From zmh
ConnectZMOK
ConnectZMFail(errorCode)
From zmh
VIA game
ConnectZM OK
ConnectZM Fail(errorCo de)
TO GameServ erHandler
Connected
LoggedIn

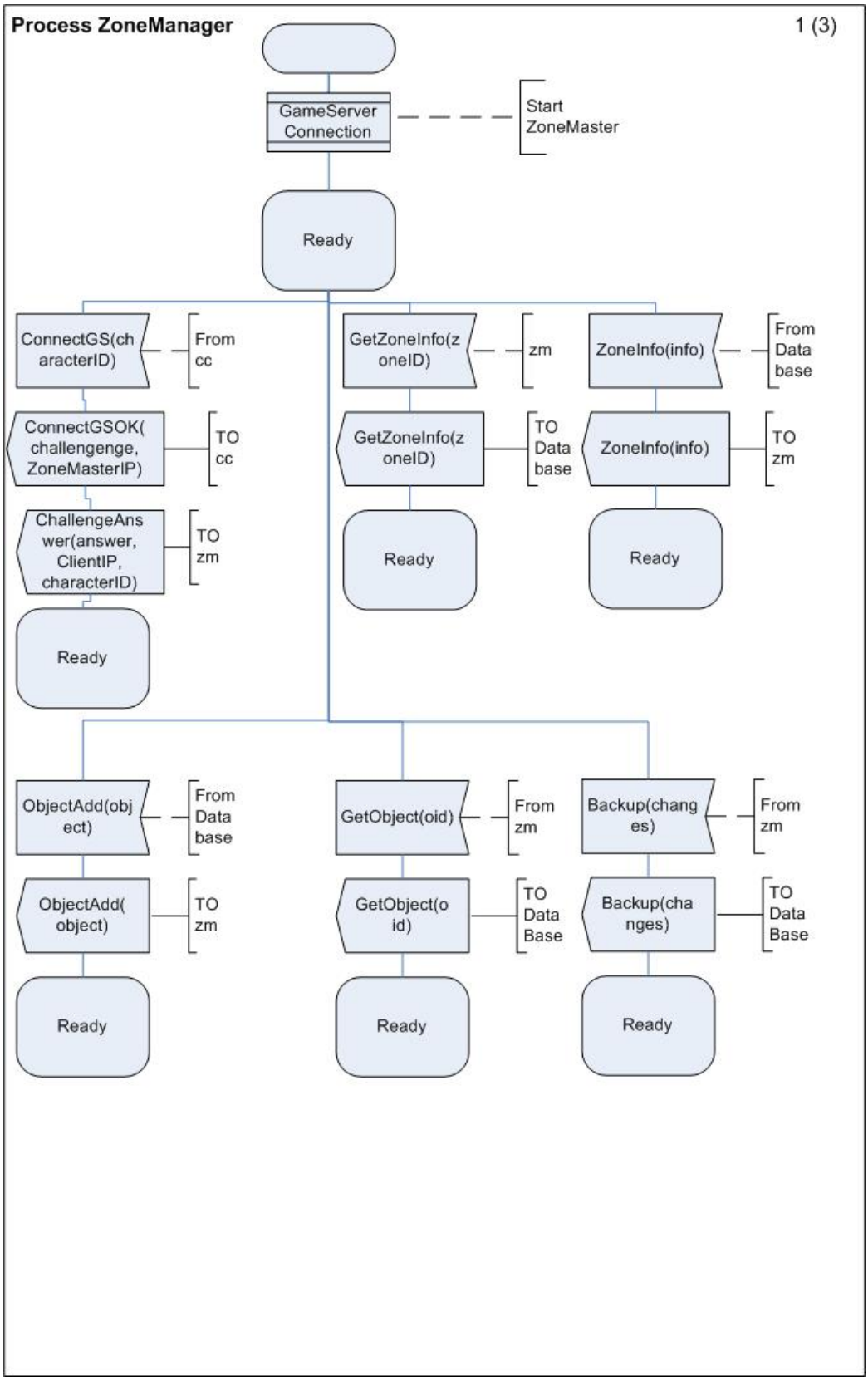

Process GameClient
3 (3)
Connected
From gsh
NewZoneMaster(ip)
TO zmh
NewZoneMaster(ip)
ZMConnecting
From game
ChangeState(objID, state, value)
TO zmh
ChangeState(objID, state, value)
Connected
CallProcedure(objID, proc, param)
From game
CallProcedure(objID, proc, param)
TO zmh
Connected
From zmh
ObjectAdd(object)
VIA game
ObjectAdd(object)
Connected
From game
ChangeState(objID, state, value)
TO zmh
ChangeState(objID, state, value)
Connected
From game
CallProcedure(objID, proc, param)
TO zmh
CallProcedure(objID, proc, param)
Connected
From zmh
AOIUpdate(objects)
VIA game
AOIUpdate(objects)
Connected

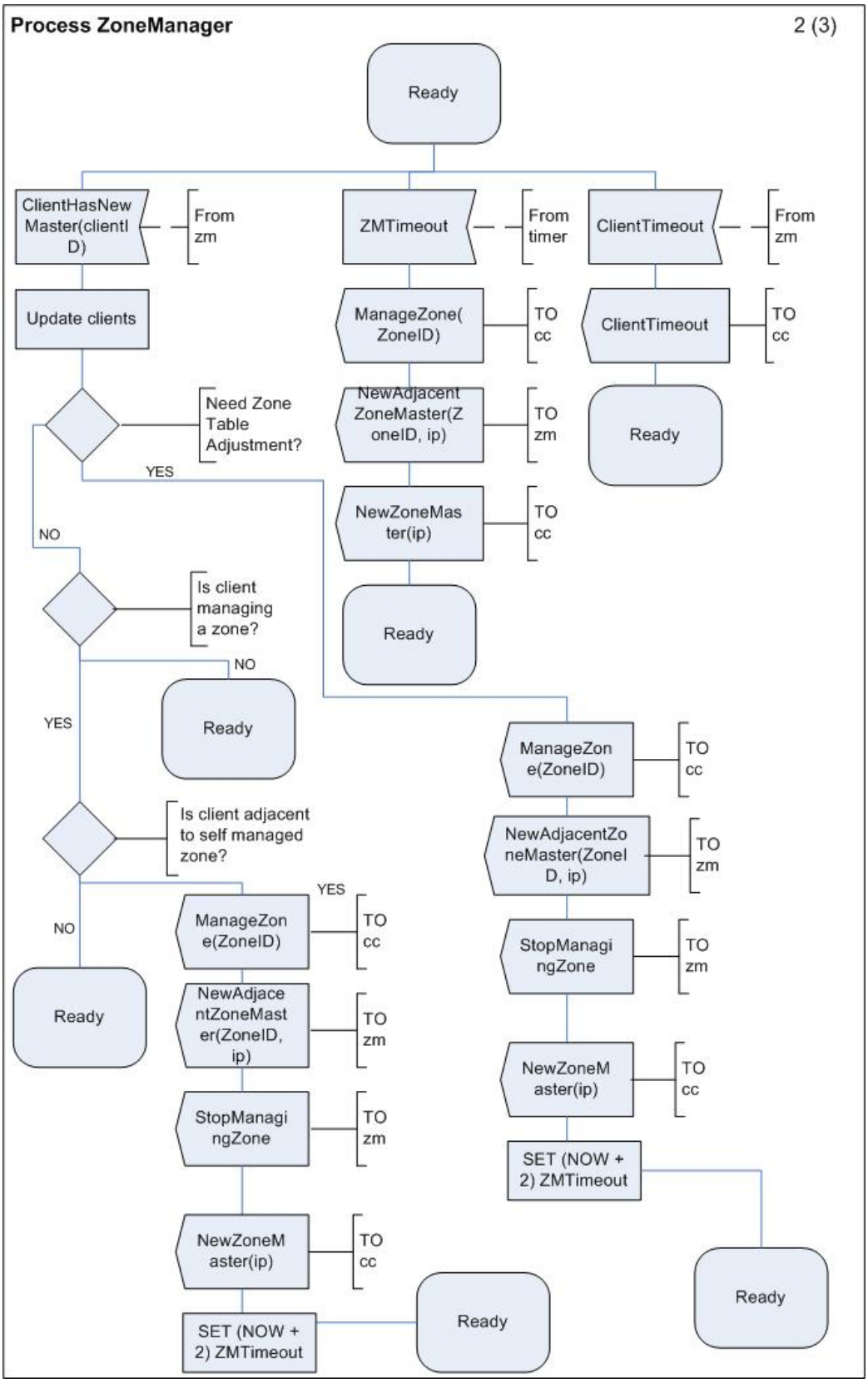

Process Type
GameServerHandler
1 (2)
Started
LogIn(username, password)
From GameClient
LogIn(username, password)
VIA gameserver
LoggingIn
From gameserver
LoginOK
LoginAccessDenied
From gameserver
TO GameClient
LogInOK
LoginAccessDenied
TO GameClient
LoggedIn
Started
From Game Client
ConnectGS(characterID)
VIA gameserver
ConnectGS(characterID)
From game serve
ConnectGSOK(challenge, zoneMasterIP)
ConnectGSFail(errorCode)
From gameserver
GSConnecting
TO Game Client
ConnectGSOK(challenge, zoneMasterIP)
ConnectGSFail(errorCode)
TO GameClient
Connected
LoggedIn

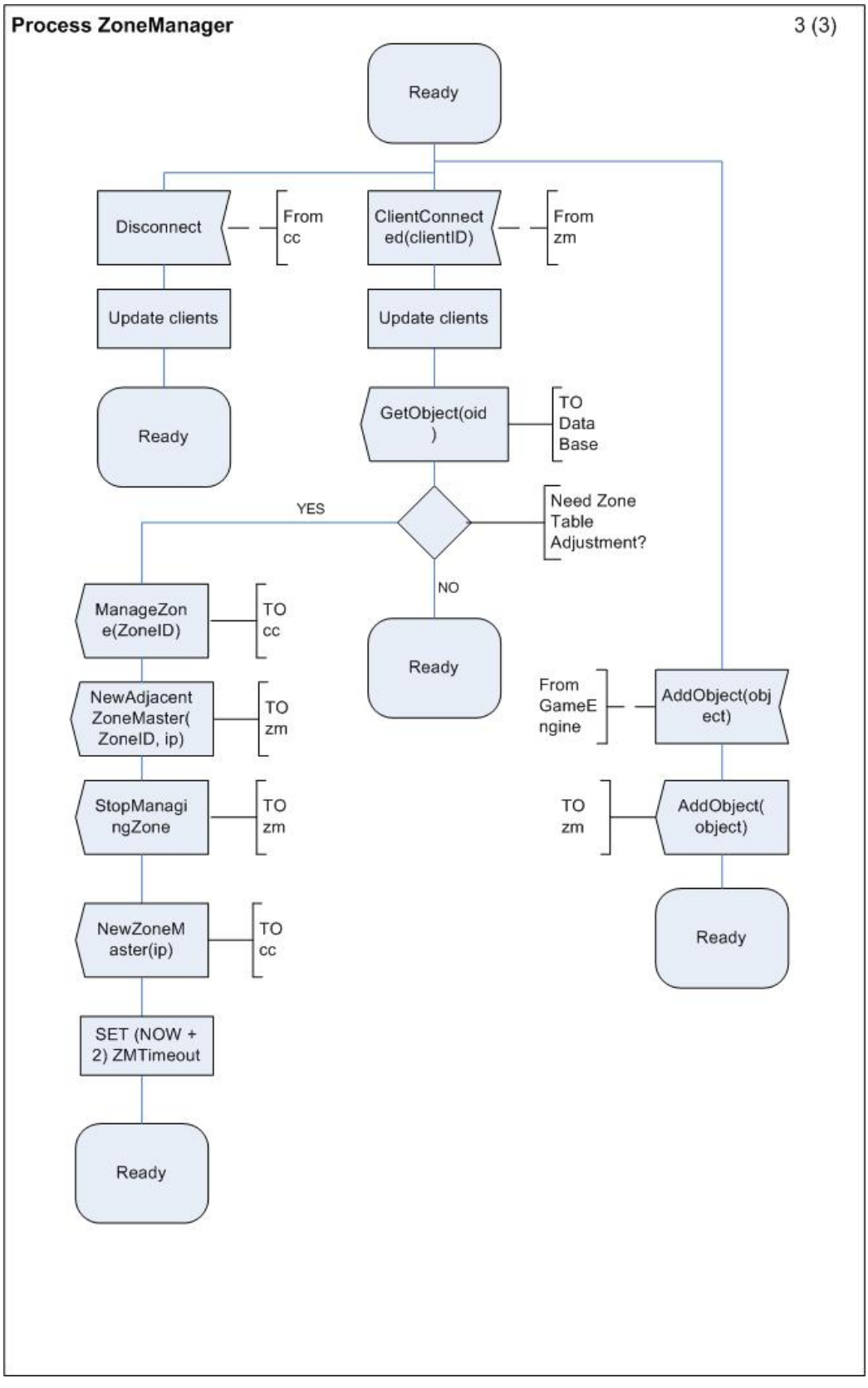

Process Type
GameServerHandler
2 (2)
Connected
ManageZone(ZoneID)
From gameserver
gsc
Connected
NewZoneMast er(ip)
From gameserver
NewZoneM aster(ip)
TO GameClient
Connected
AreYouAlive
From gameserver
IAmAlive
VIA gameserver
Connected
ConnectZMFai l(errorCode)
From gameserver
CannotCon nectToZM
VIA gameserver
Connected

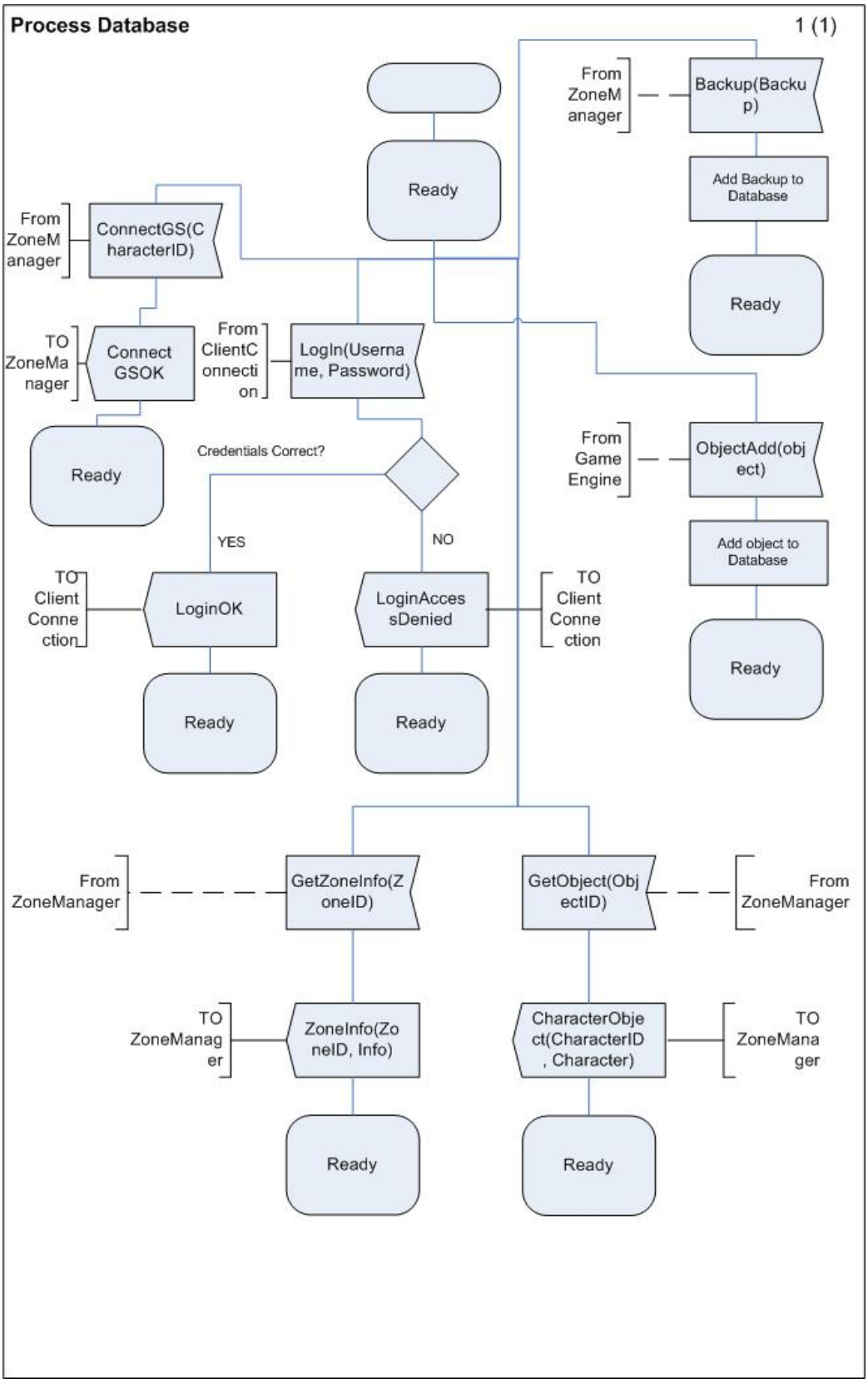

Process Type
ZoneMasterHandler
1 (2)
Started
ConnectZM(challenge, characterID)
From GameClient
ConnectZM(challenge, characterID)
VIA zoneconnection
ZMConnecting
From zoneconnection
ConnectZMOK
ConnectZMFail (errorCode)
From zoneconnection
TO GameClient
ConnectZM OK
ConnectZM Fail(errorCode)
TO GameClient
Connected
Started

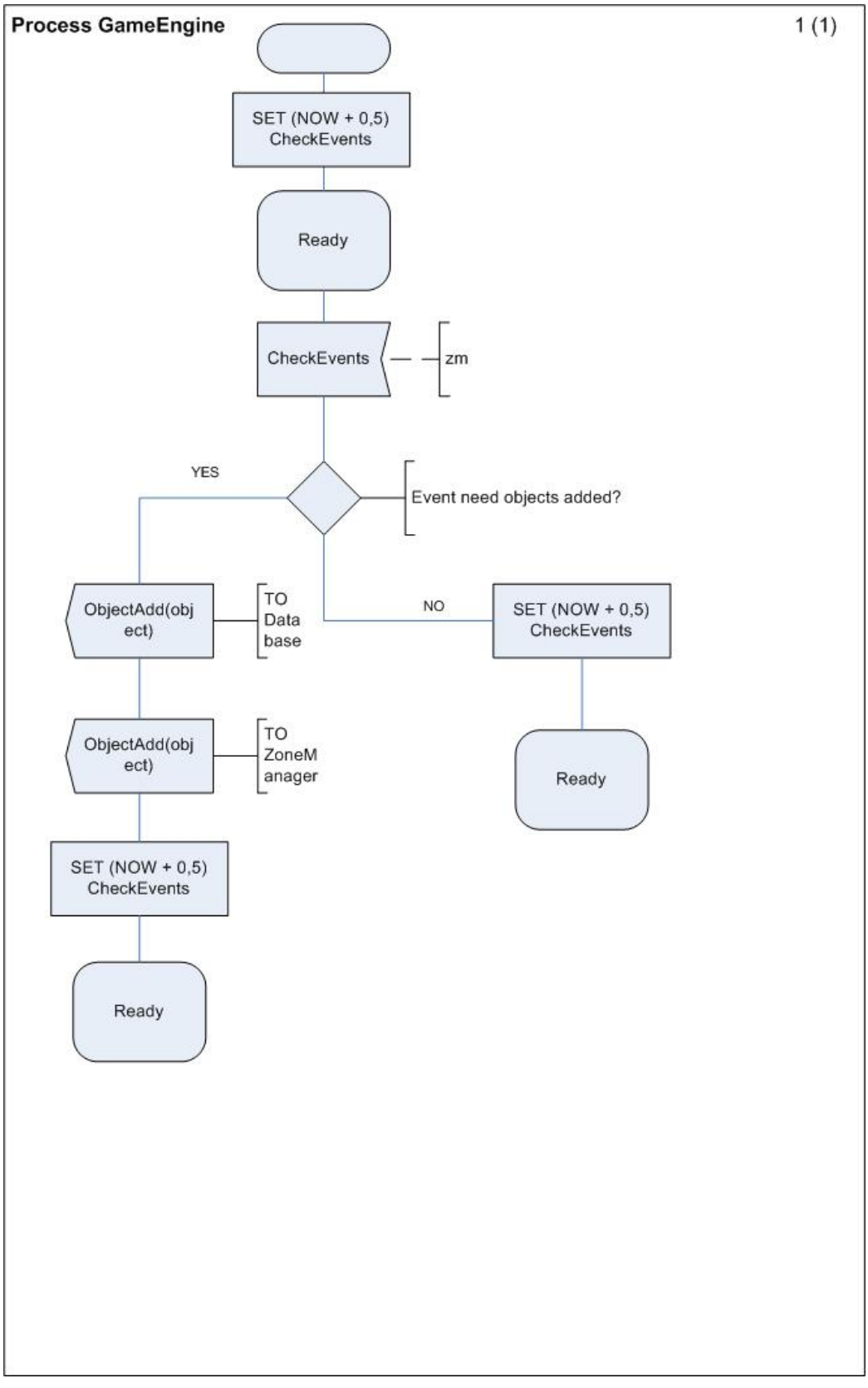

Process Type
ZoneMasterHandler
2 (2)
Connected
SET (NOW + x)
KeepAlive
From zoneconnection, GameClient
NewZoneMaster(ip)
From zonecon nection
ChangeState(objID, state, value)
CallProcedure(objID, proc, param)
From zonecon nection
VIA zonecon nection
ConnectZM(challenge, characterID)
TO Game Client
ChangeStat e(objID, state, value)
CallProcedu re(objID, proc, param)
TO Game Client
ZMConnecting
Connected
Connected
From zoneconn ection
ObjectAdd(obj ect)
From Game Client
ChangeState(objID, state, value)
From Game Client
CallProcedure (objID, proc, param)
TO Game Client
ObjectAdd(object)
VIA zonecon nection
ChangeStat e(objID, state, value)
VIA zonecon nection
CallProcedu re(objID, proc, param)
Connected
Connected
Connected
From zoneconn ection
AOIUpdate(obj ects)
From TIMER
KeepAlive
TO Game Client
AOIUpdate(objects)
VIA zonecon nection
KeepAlive
Connected
Connected

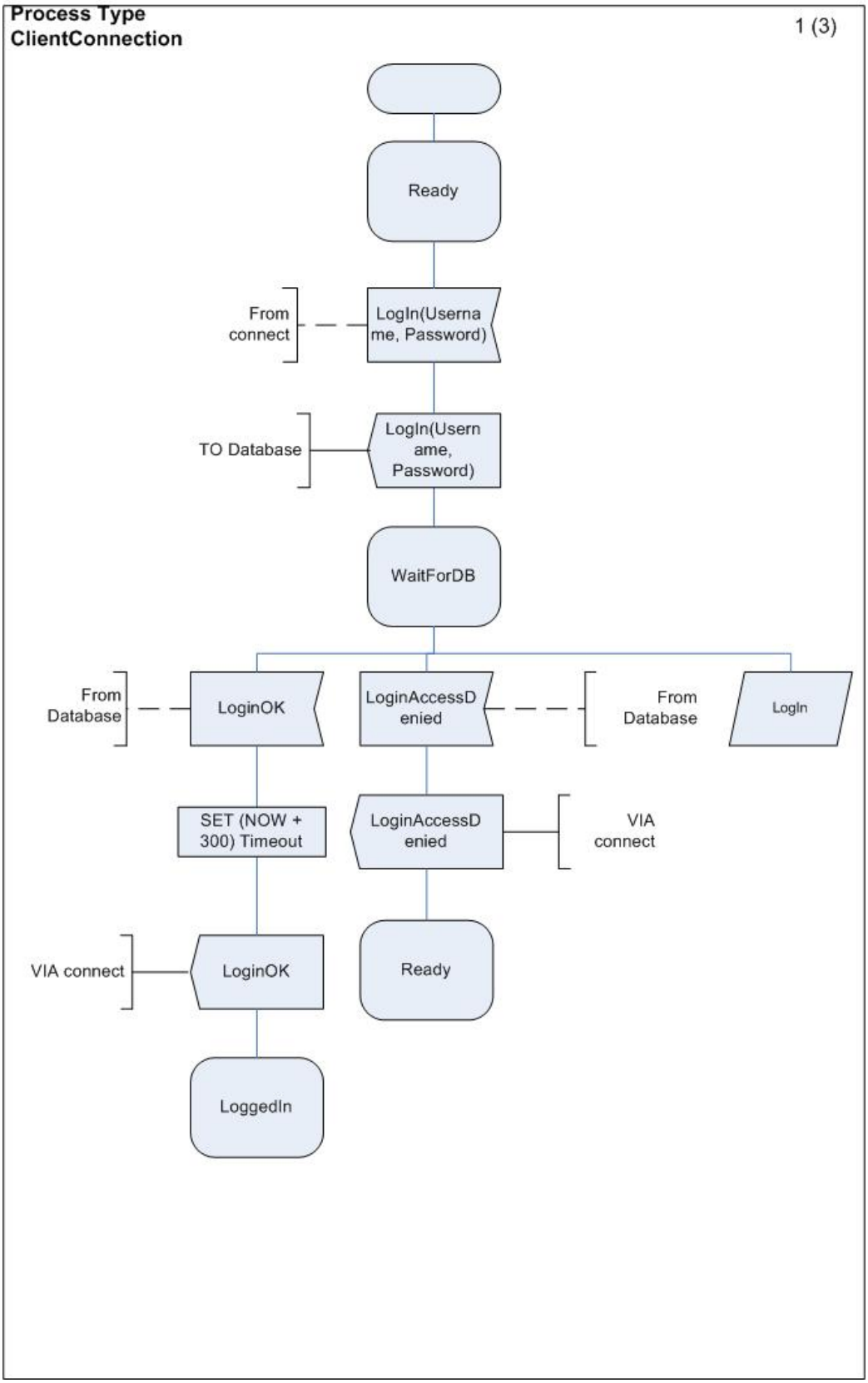

Process ZoneManager
1 (3)
GameServer Connection
Start ZoneMaster
Ready
ConnectGS(characterID)
From cc
ConnectGSOK(challengenge, ZoneMasterIP)
TO cc
ChallengeAnswer(answer, ClientIP, characterID)
TO zm
Ready
GetZoneInfo(zoneID)
zm
GetZoneInfo(zoneID)
TO Data base
Ready
ZoneInfo(info)
From Data base
ZoneInfo(info)
TO zm
Ready
ObjectAdd(object)
From Data base
ObjectAdd(object)
TO zm
Ready
GetObject(oid)
From zm
GetObject(oid)
TO Data Base
Ready
Backup(changes)
From zm
Backup(changes)
TO Data Base
Ready

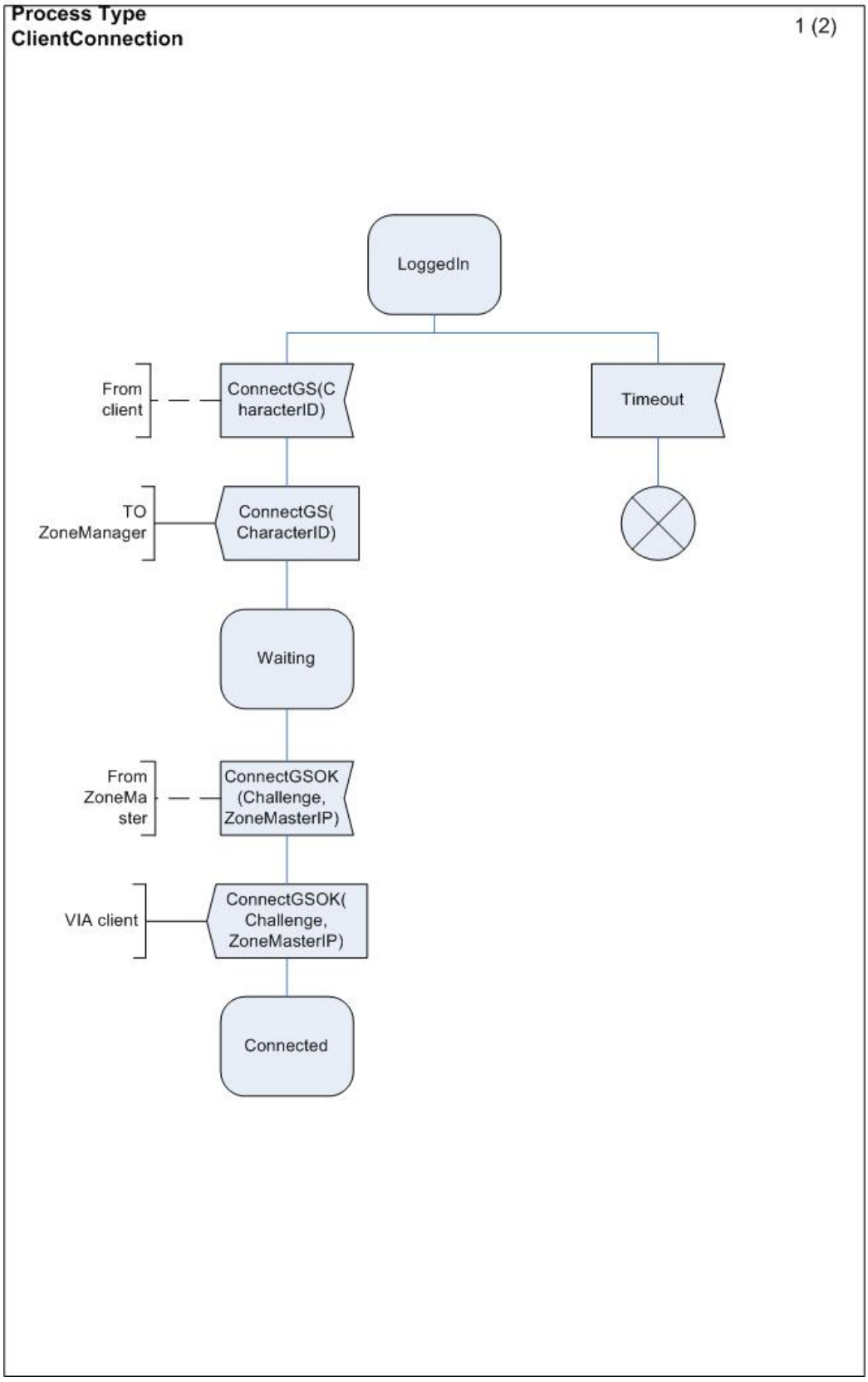

Process ZoneManager
2 (3)
Ready
ClientHasNewMaster(clientID)
From zm
ZMTimeout
From timer
ClientTimeout
From zm
Update clients
ManageZone(ZoneID)
TO cc
ClientTimeout
TO cc
Need Zone Table Adjustment?
NewAdjacentZoneMaster(ZoneID, ip)
TO zm
Ready
YES
NewZoneMaster(ip)
TO cc
NO
Is client managing a zone?
Ready
NO
YES
Ready
ManageZone(ZoneID)
TO cc
Is client adjacent to self managed zone?
NewAdjacentZoneMaster(ZoneID, ip)
TO zm
YES
NO
ManageZone(ZoneID)
TO cc
StopManagingZone
TO zm
Ready
NewAdjacentZoneMaster(ZoneID, ip)
TO zm
NewZoneMaster(ip)
TO cc
StopManagingZone
TO zm
SET (NOW + 2) ZMTimeout
NewZoneMaster(ip)
TO cc
Ready
SET (NOW + 2) ZMTimeout
Ready

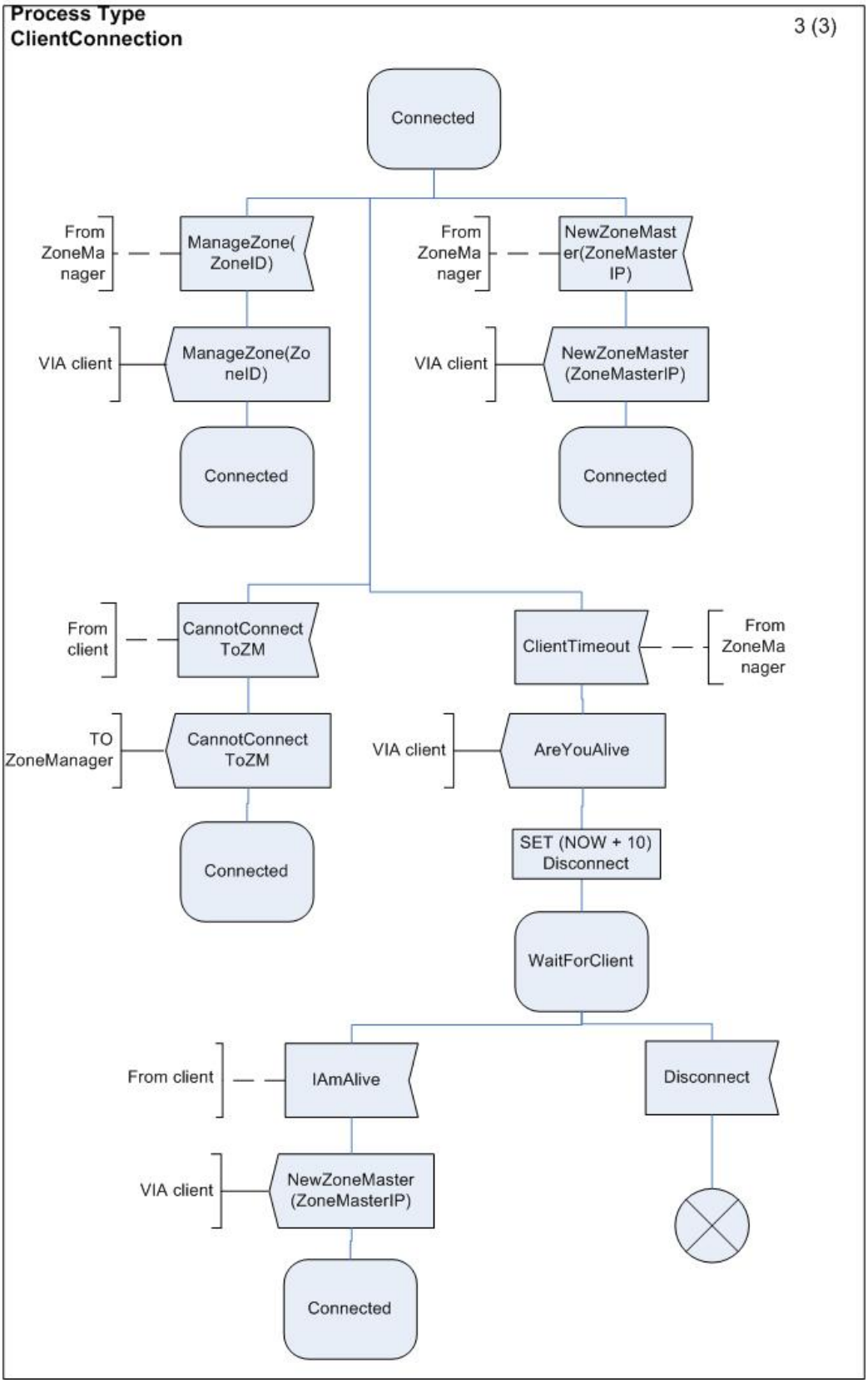

Process ZoneManager
3 (3)
Ready
Disconnect
From cc
ClientConnected(clientID)
From zm
Update clients
Update clients
Ready
GetObject(oid)
TO Data Base
YES
Need Zone Table Adjustment?
NO
ManageZone(ZoneID)
TO cc
Ready
NewAdjacent ZoneMaster(ZoneID, ip)
TO zm
From GameEngine
AddObject(object)
StopManagingZone
TO zm
TO zm
AddObject(object)
NewZoneMaster(ip)
TO cc
Ready
SET (NOW + 2) ZMTimeout
Ready

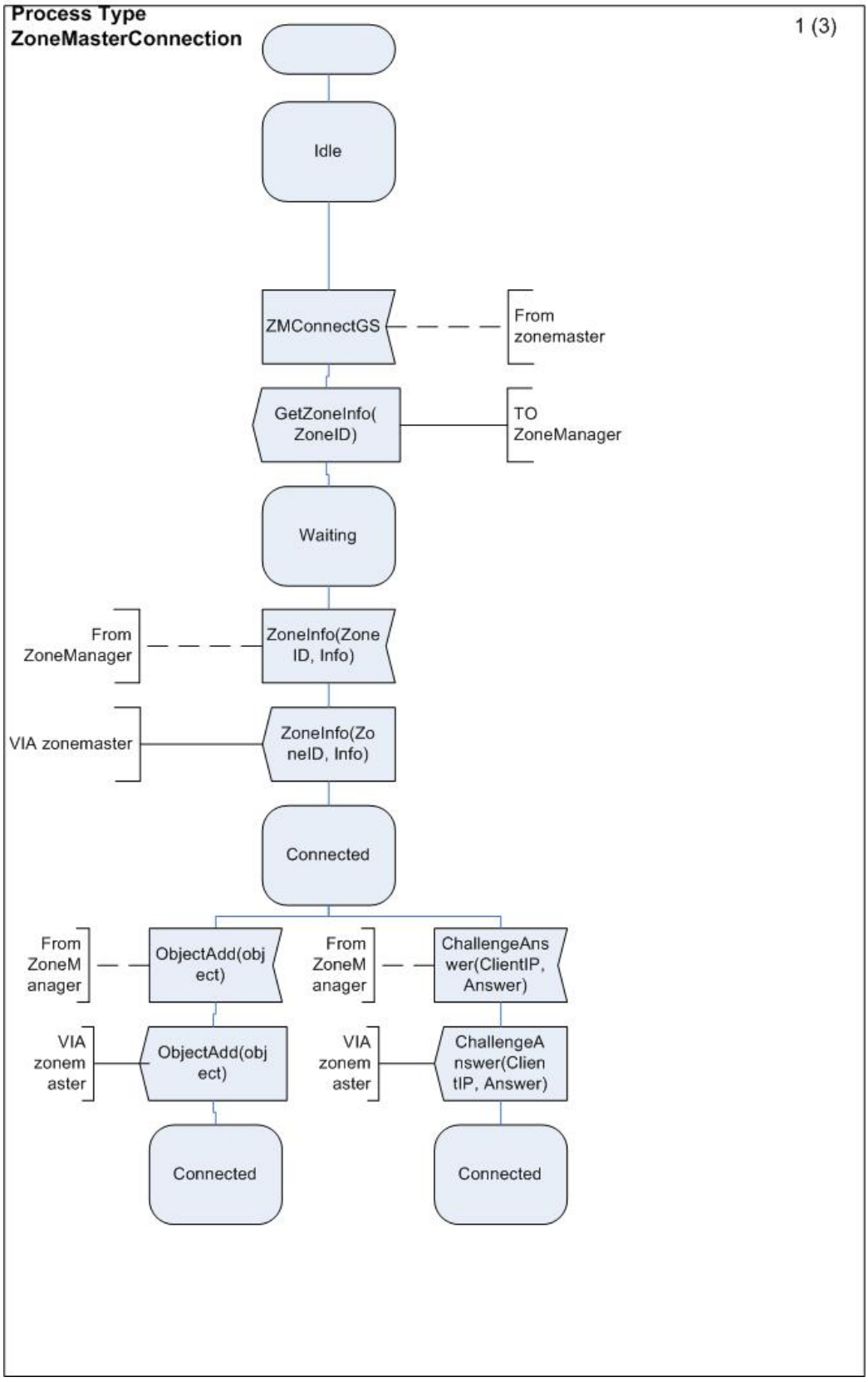

Process Database
1 (1)
Ready
From ZoneManager
Backup(Backup)
Add Backup to Database
Ready
From ZoneManager
ConnectGS(CharacterID)
TO ZoneManager
Connect GSOK
From ClientConnection
LogIn(Username, Password)
Ready
Credentials Correct?
YES
NO
From Game Engine
ObjectAdd(object)
Add object to Database
Ready
TO Client Connection
LoginOK
LoginAccessDenied
TO Client Connection
Ready
Ready
From ZoneManager
GetZoneInfo(ZoneID)
GetObject(ObjectID)
From ZoneManager
TO ZoneManager
ZoneInfo(ZoneID, Info)
CharacterObject(CharacterID, Character)
TO ZoneManager
Ready
Ready

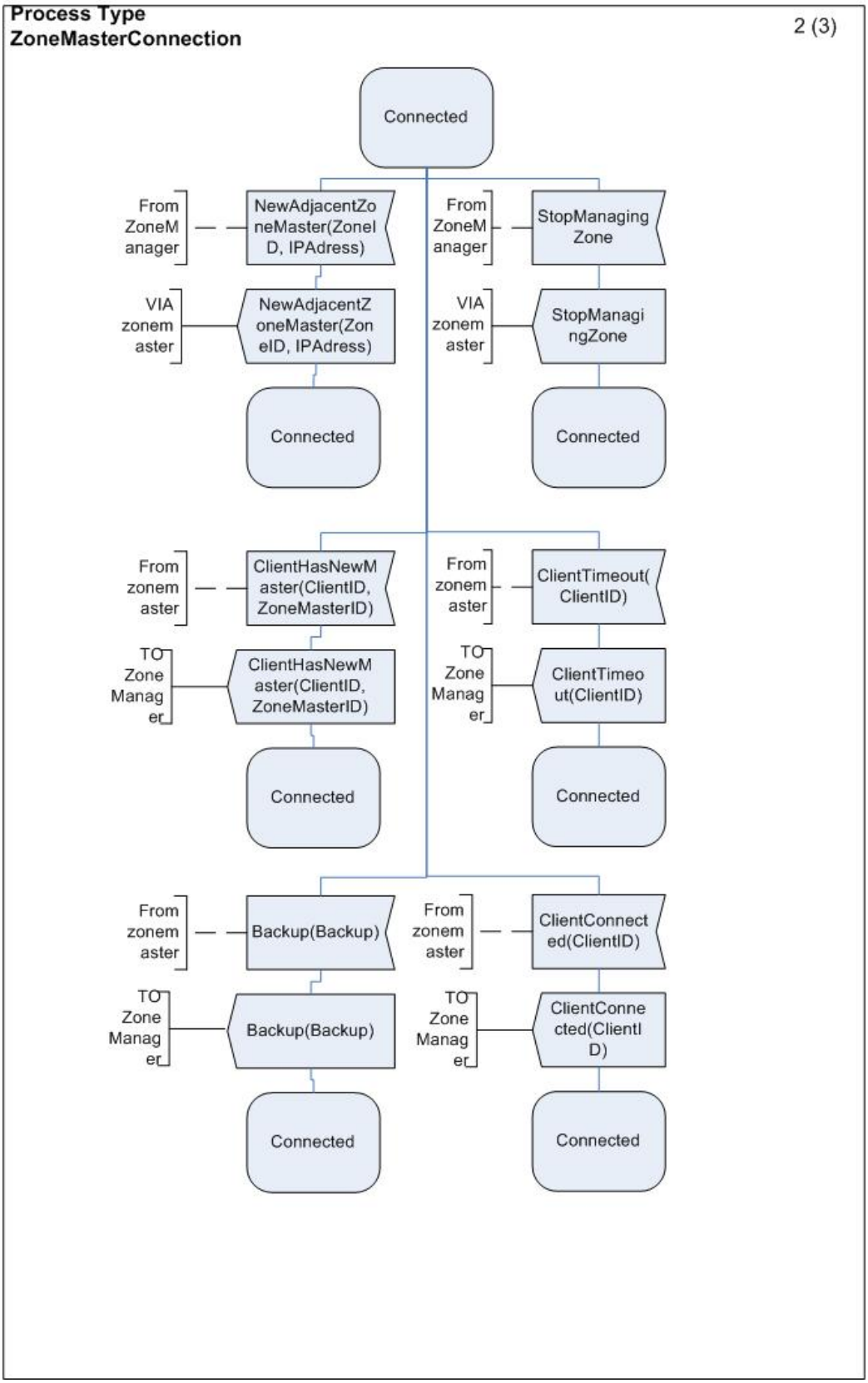

Process GameEngine
1 (1)
SET (NOW + 0,5)
CheckEvents
Ready
CheckEvents
zm
YES
Event need objects added?
ObjectAdd(obj
ect)
TO
Data
base
NO
SET (NOW + 0,5)
CheckEvents
ObjectAdd(obj
ect)
TO
ZoneM
anager
Ready
SET (NOW + 0,5)
CheckEvents
Ready


Process Type
ClientConnection
1 (3)
Ready
From connect
LogIn(Username, Password)
TO Database
LogIn(Username, Password)
WaitForDB
From Database
LoginOK
LoginAccessDenied
From Database
LogIn
SET (NOW + 300) Timeout
LoginAccessDenied
VIA connect
VIA connect
LoginOK
Ready
LoggedIn


Process Type
ClientConnection
1 (2)
LoggedIn
From client
ConnectGS(CharacterID)
Timeout
TO ZoneManager
ConnectGS(CharacterID)
Waiting
From ZoneMaster
ConnectGSOK (Challenge, ZoneMasterIP)
VIA client
ConnectGSOK(Challenge, ZoneMasterIP)
Connected

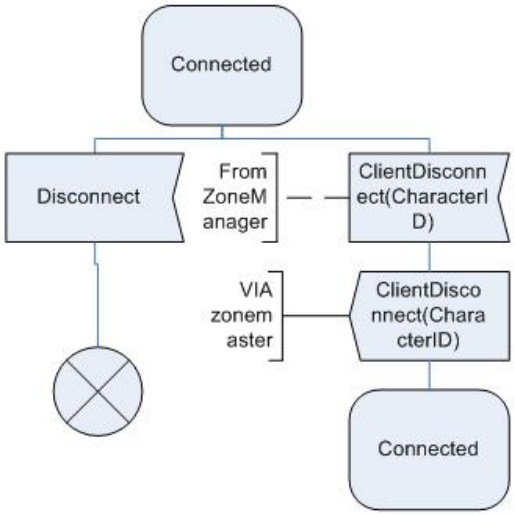

Process Type
ClientConnection
3 (3)
Connected
From ZoneManager
ManageZone(ZoneID)
From ZoneManager
NewZoneMaster(ZoneMaster IP)
VIA client
ManageZone(ZoneID)
VIA client
NewZoneMaster (ZoneMasterIP)
Connected
Connected
From client
CannotConnect ToZM
ClientTimeout
From ZoneManager
TO ZoneManager
CannotConnect ToZM
VIA client
AreYouAlive
Connected
SET (NOW + 10) Disconnect
WaitForClient
From client
IAmAlive
Disconnect
VIA client
NewZoneMaster (ZoneMasterIP)
Connected

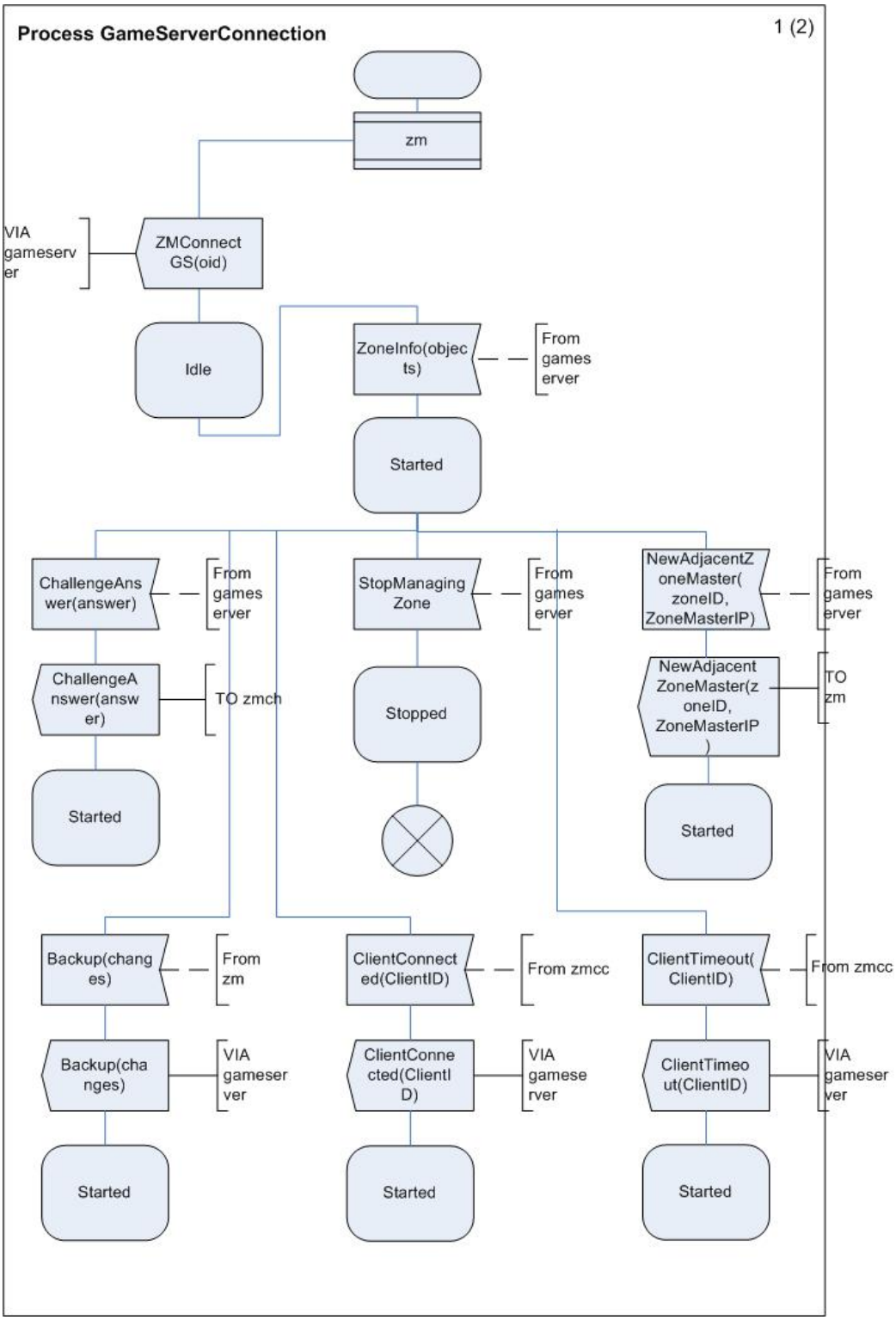

Process Type
ZoneMasterConnection
1 (3)
Idle
ZMConnectGS
From zonemaster
GetZoneInfo(ZoneID)
TO ZoneManager
Waiting
From ZoneManager
ZoneInfo(Zone ID, Info)
VIA zonemaster
ZoneInfo(Zo neID, Info)
Connected
From ZoneM anager
ObjectAdd(obj ect)
From ZoneM anager
ChallengeAns wer(ClientIP, Answer)
VIA zonem aster
ObjectAdd(obj ect)
VIA zonem aster
ChallengeA nswer(Clien tIP, Answer)
Connected
Connected

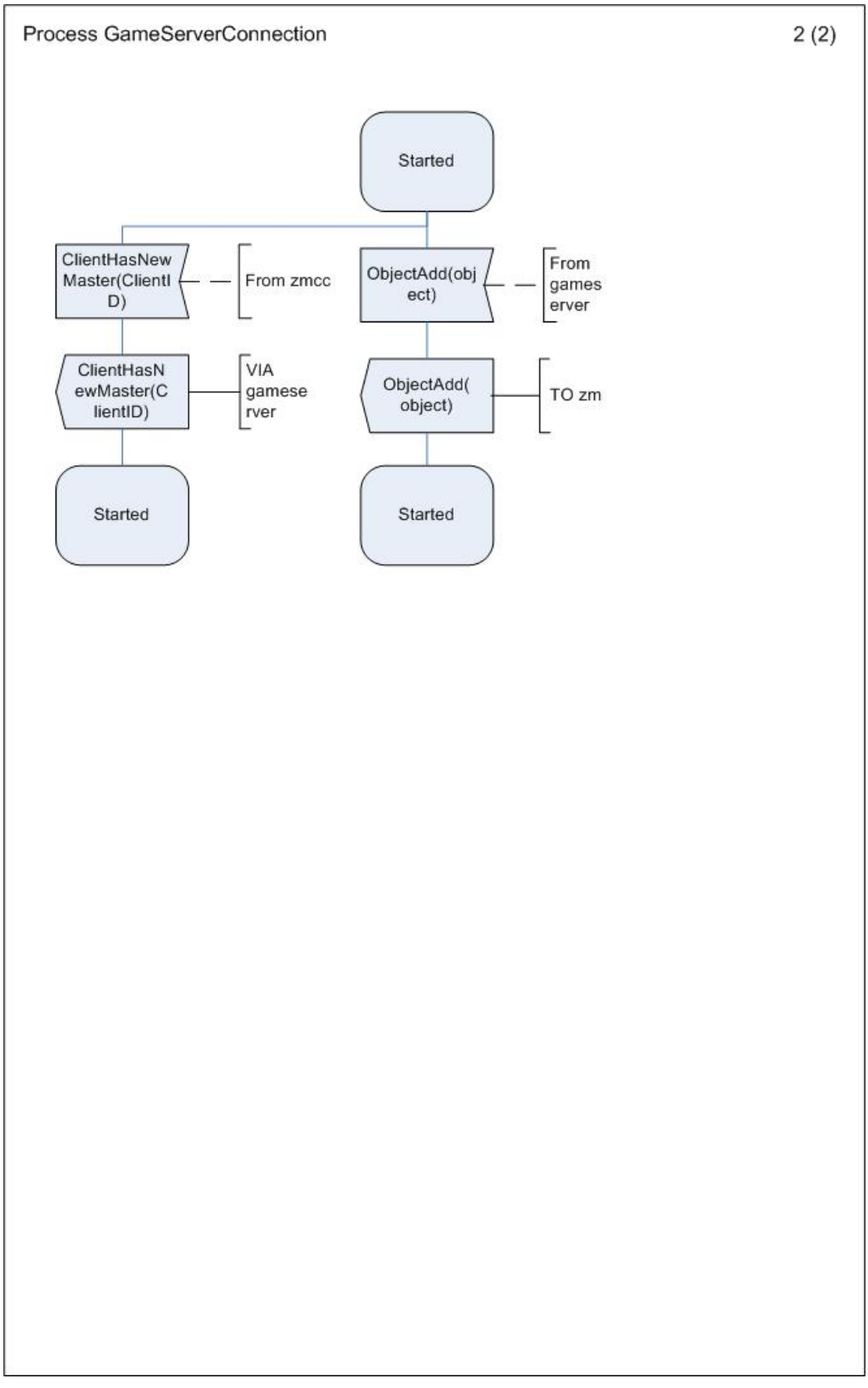

Process Type
ZoneMasterConnection
2 (3)
Connected
From ZoneManager
NewAdjacentZoneMaster(ZoneID, IPAdress)
VIA zonemaster
NewAdjacentZoneMaster(ZoneID, IPAdress)
Connected
From ZoneManager
StopManagingZone
VIA zonemaster
StopManagingZone
Connected
From zonemaster
ClientHasNewMaster(ClientID, ZoneMasterID)
TO ZoneManager
ClientHasNewMaster(ClientID, ZoneMasterID)
Connected
From zonemaster
ClientTimeout(ClientID)
TO ZoneManager
ClientTimeout(ClientID)
Connected
From zonemaster
Backup(Backup)
TO ZoneManager
Backup(Backup)
Connected
From zonemaster
ClientConnected(ClientID)
TO ZoneManager
ClientConnected(ClientID)
Connected

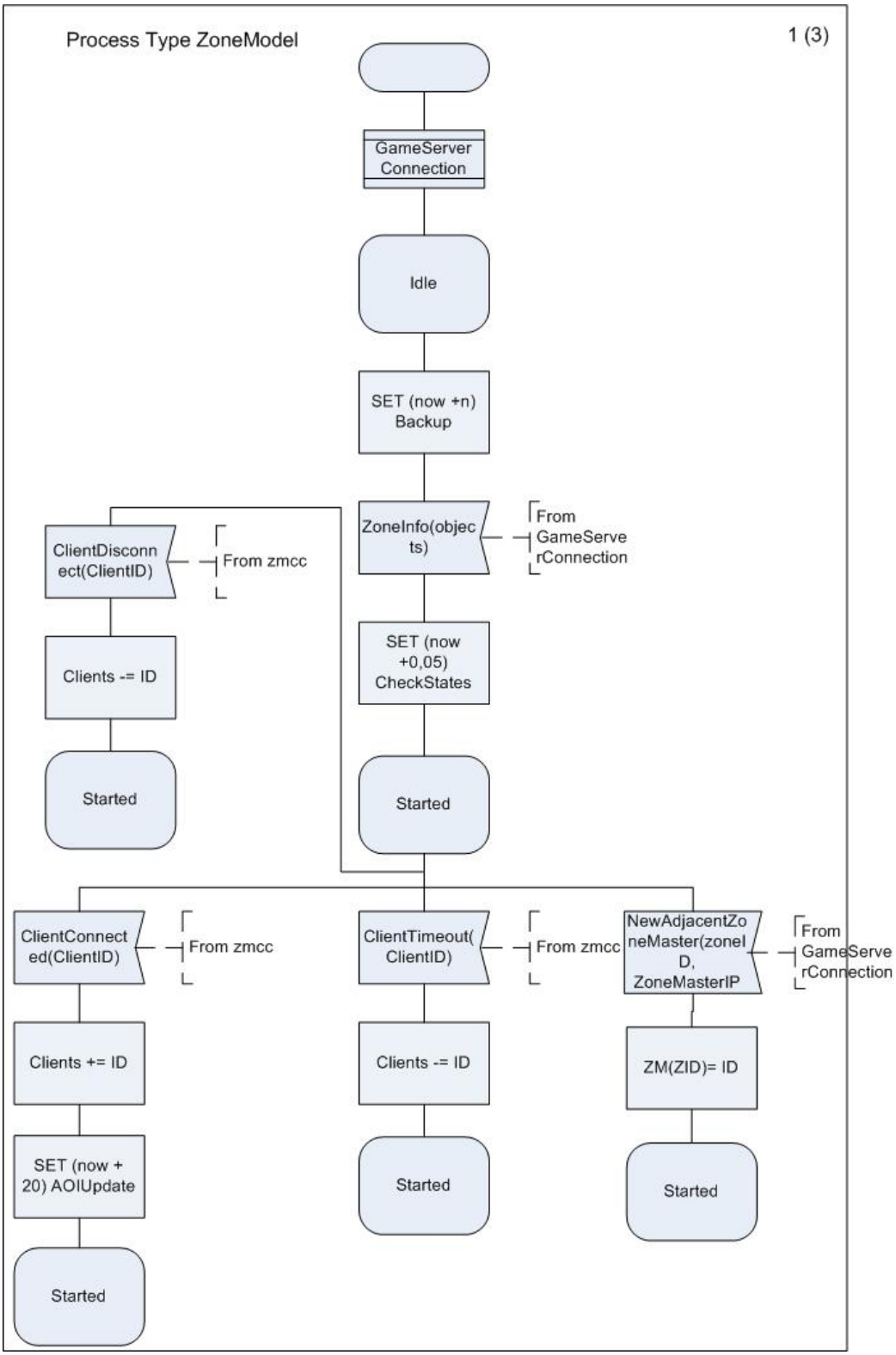
Process Type
ZoneMasterConnection

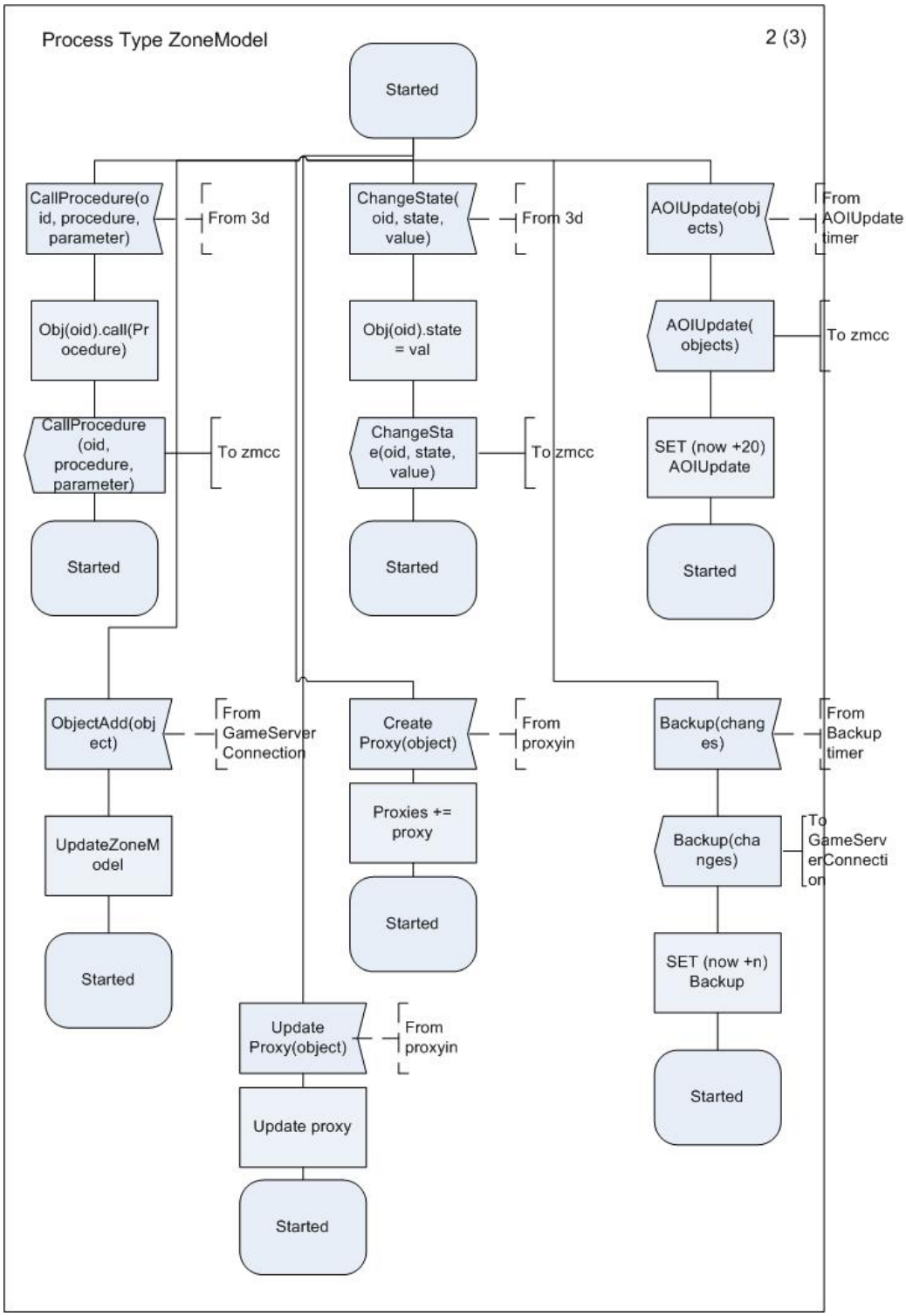
3 (3)

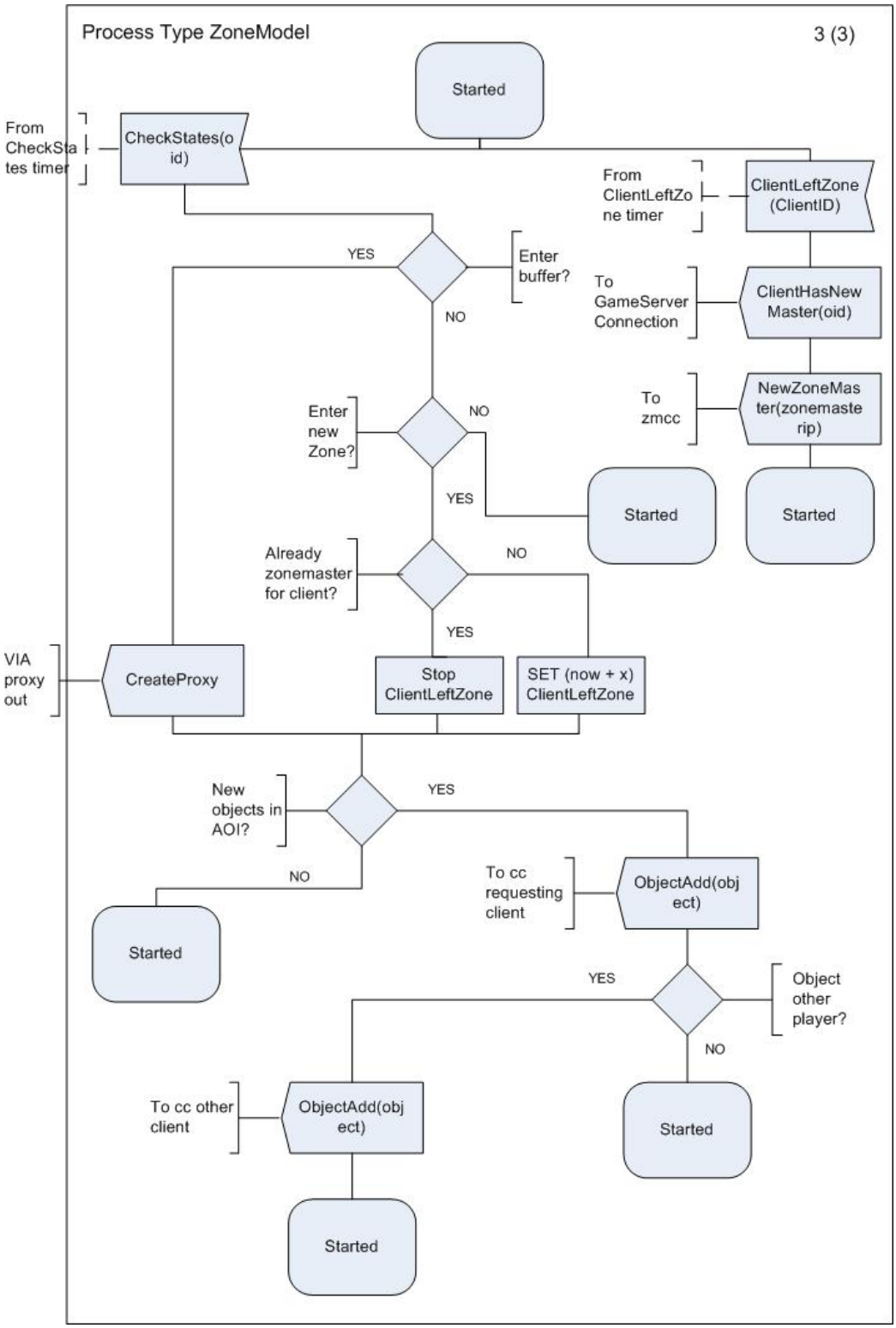


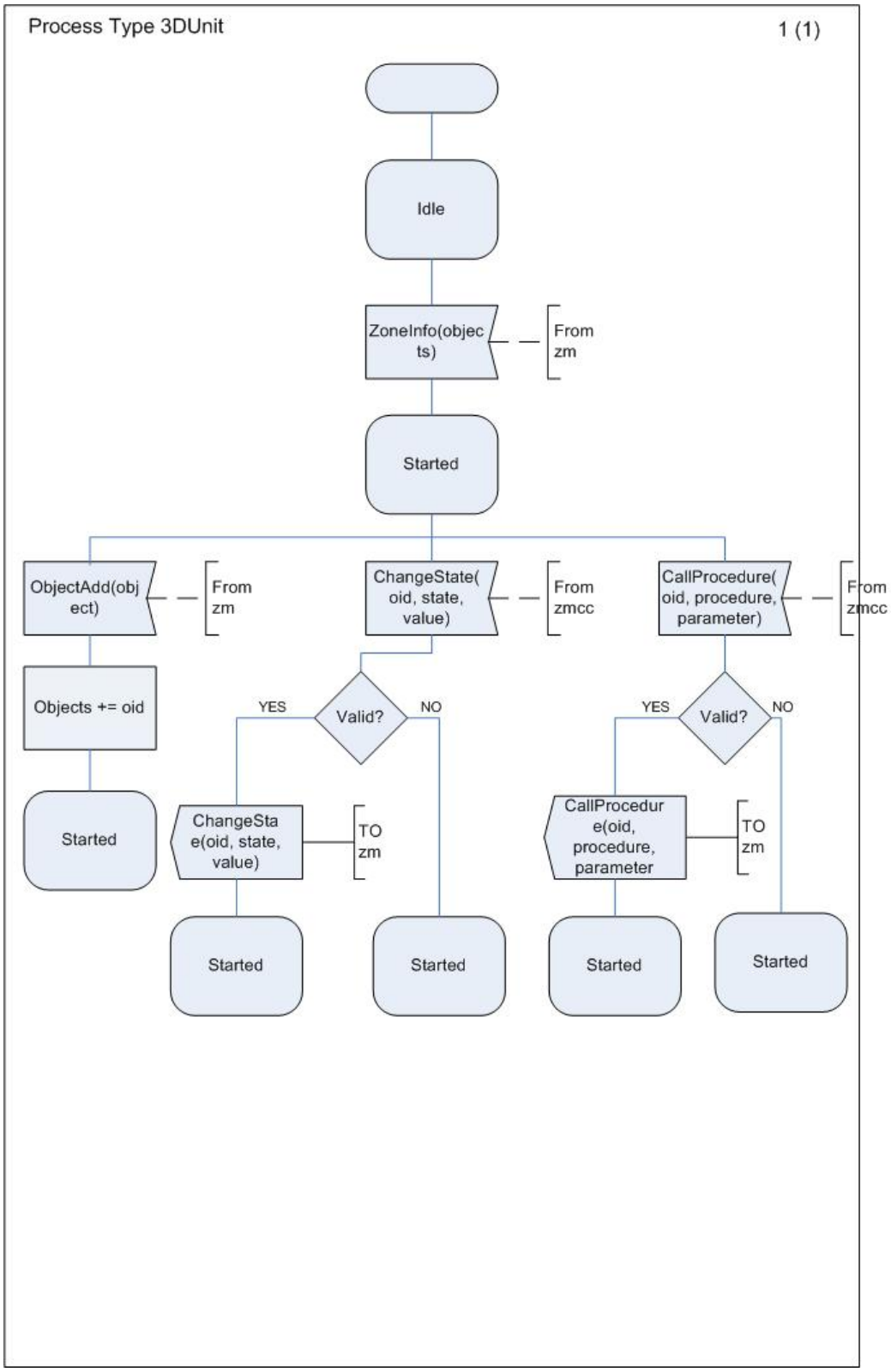

Process GameServerConnection
1 (2)
zm
VIA gameserver
ZMConnectGS(oid)
Idle
ZoneInfo(objects)
From gameserver
Started
ChallengeAnswer(answer)
From gameserver
ChallengeAnswer(answer)
TO zmch
Started
StopManagingZone
From gameserver
Stopped
NewAdjacentZoneMaster(zoneID, ZoneMasterIP)
From gameserver
NewAdjacentZoneMaster(zoneID, ZoneMasterIP)
TO zm
Started
Backup(changes)
From zm
Backup(changes)
VIA gameserver
Started
ClientConnected(ClientID)
From zmcc
ClientConnected(ClientID)
VIA gameserver
Started
ClientTimeout(ClientID)
From zmcc
ClientTimeout(ClientID)
VIA gameserver
Started

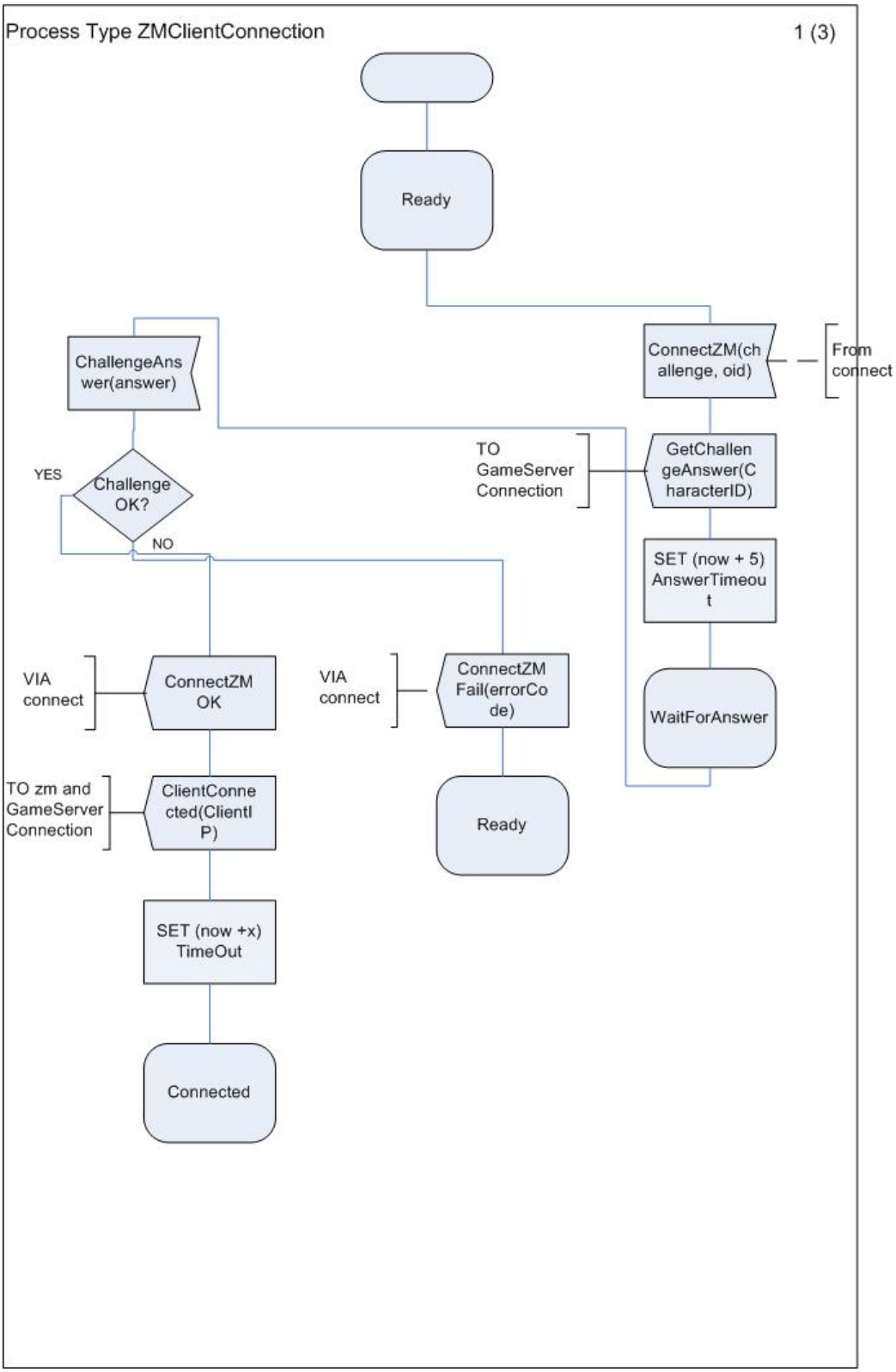

Process GameServerConnection
2 (2)
Started
ClientHasNew Master(ClientI D)
From zmcc
ObjectAdd(obj ect)
From games erver
ClientHasN ewMaster(C lientID)
VIA gamese rver
ObjectAdd(object)
TO zm
Started
Started

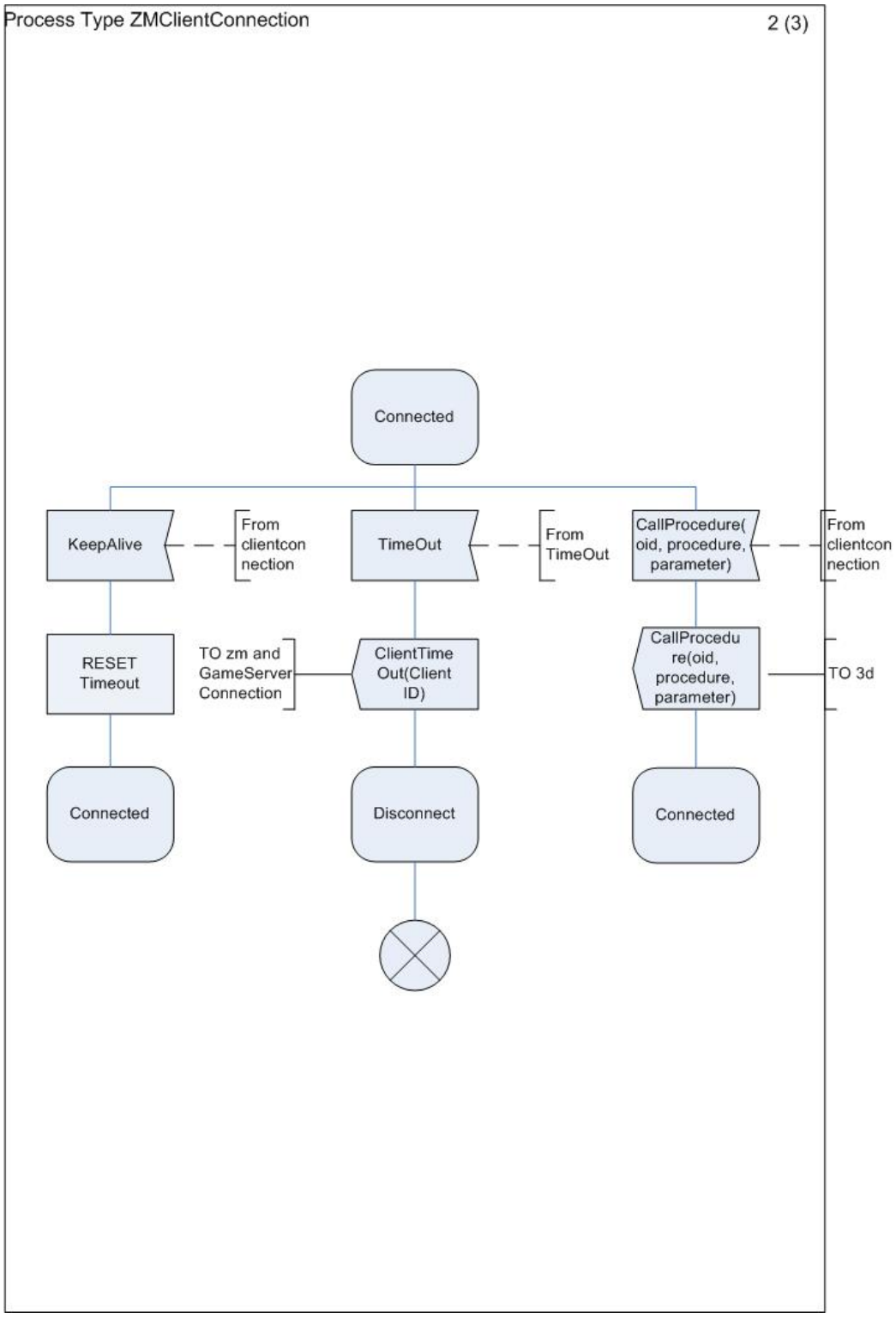

Process Type ZoneModel
1 (3)
GameServer Connection
Idle
SET (now +n) Backup
ZoneInfo(objects)
From GameServerConnection
ClientDisconnect(ClientID)
From zmcc
Clients -= ID
Started
SET (now +0,05) CheckStates
Started
ClientConnected(ClientID)
From zmcc
Clients += ID
SET (now + 20) AOIUpdate
Started
ClientTimeout(ClientID)
From zmcc
Clients -= ID
Started
NewAdjacentZoneMaster(zoneID, ZoneMasterIP
From GameServerConnection
ZM(ZID)= ID
Started

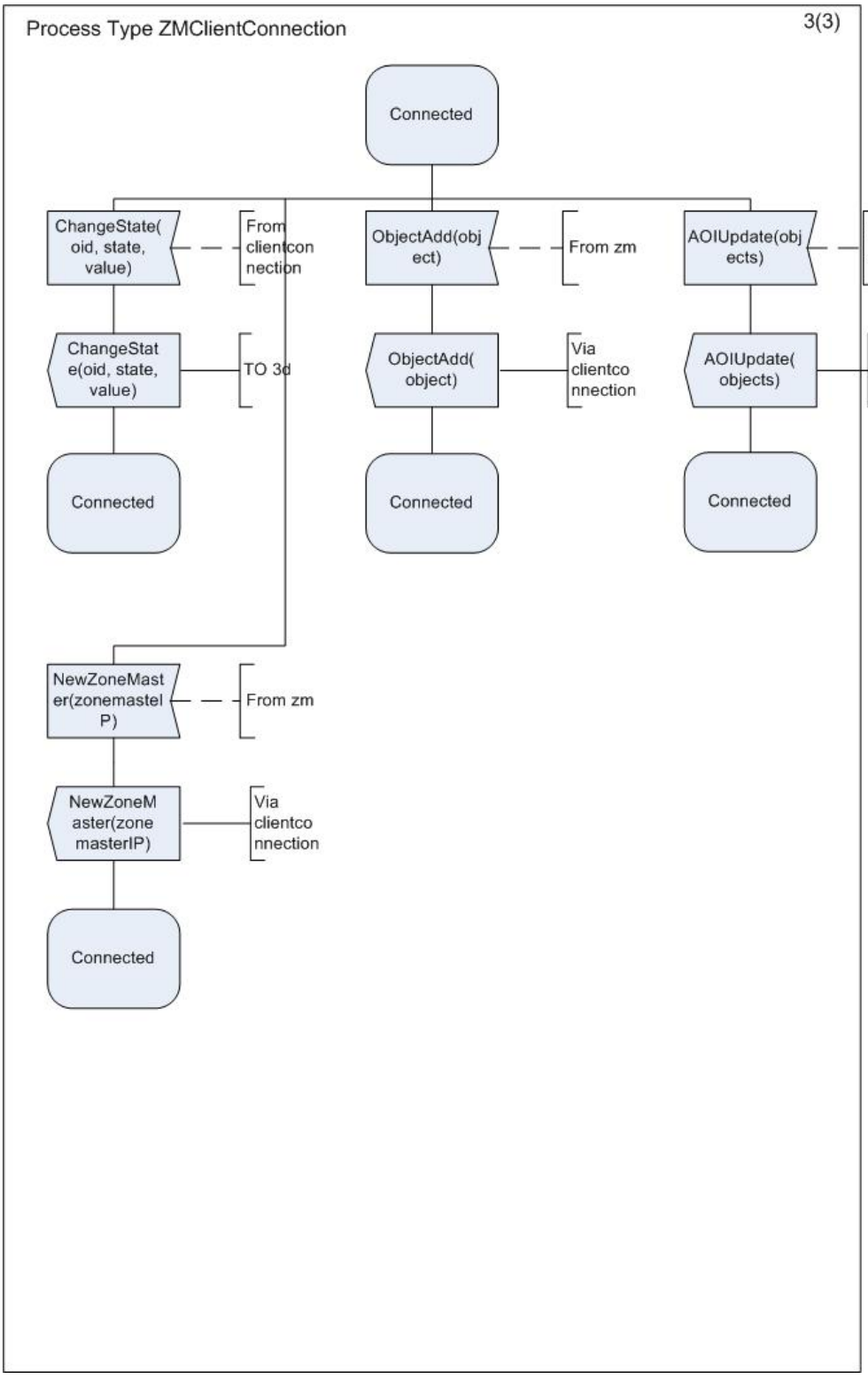

Process Type ZoneModel
2 (3)
Started
CallProcedure(oid, procedure, parameter)
From 3d
ChangeState(oid, state, value)
From 3d
AOIUpdate(objects)
From AOIUpdate timer
Obj(oid).call(Procedure)
Obj(oid).state = val
AOIUpdate(objects)
To zmcc
CallProcedure(oid, procedure, parameter)
To zmcc
ChangeState(oid, state, value)
To zmcc
SET (now +20) AOIUpdate
Started
Started
Started
ObjectAdd(object)
From GameServer Connection
Create Proxy(object)
From proxyin
Backup(changes)
From Backup timer
UpdateZoneModel
Proxies += proxy
Backup(changes)
To GameServerConnection
Started
Started
SET (now +n) Backup
Update Proxy(object)
From proxyin
Started
Update proxy
Started


Process Type ZoneModel
3 (3)
Started
From CheckStates timer
CheckStates(oid)
From ClientLeftZone timer
ClientLeftZone (ClientID)
YES
Enter buffer?
NO
To GameServer Connection
ClientHasNewMaster(oid)
Enter new Zone?
NO
To zmcc
NewZoneMaster(zonemasterip)
YES
Started
Started
Already zonemaster for client?
NO
YES
VIA proxy out
CreateProxy
Stop ClientLeftZone
SET (now + x) ClientLeftZone
New objects in AOI?
YES
NO
To cc requesting client
ObjectAdd(object)
Started
YES
Object other player?
NO
To cc other client
ObjectAdd(object)
Started
Started


Process Type 3DUnit
1 (1)
Idle
ZoneInfo(objects)
From zm
Started
ObjectAdd(object)
From zm
ChangeState(oid, state, value)
From zmcc
CallProcedure(oid, procedure, parameter)
From zmcc
Objects += oid
YES
Valid?
NO
YES
Valid?
NO
Started
ChangeState(oid, state, value)
TO zm
CallProcedure(oid, procedure, parameter
TO zm
Started
Started
Started
Started


Process Type ZMClientConnection
1 (3)
Ready
ChallengeAns wer(answer)
ConnectZM(ch allenge, oid)
From connect
YES
Challenge OK?
NO
TO GameServer Connection
GetChallen geAnswer(C haracterID)
SET (now + 5) AnswerTimeou t
VIA connect
ConnectZM OK
VIA connect
ConnectZM Fail(errorCo de)
WaitForAnswer
TO zm and GameServer Connection
ClientConne cted(ClientI P)
Ready
SET (now +x) TimeOut
Connected


Process Type ZMClientConnection
2 (3)
Connected
KeepAlive
From clientcon nection
TimeOut
From TimeOut
CallProcedure(oid, procedure, parameter)
From clientcon nection
RESET Timeout
TO zm and GameServer Connection
ClientTime Out(Client ID)
CallProcedu re(oid, procedure, parameter)
TO 3d
Connected
Disconnect
Connected


Process Type ZMClientConnection
3(3)
Connected
ChangeState(oid, state, value)
From clientcon nection
ObjectAdd(obj ect)
From zm
AOIUpdate(obj ects)
From zm
ChangeStat e(oid, state, value)
TO 3d
ObjectAdd(object)
Via clientco nnection
AOIUpdate(objects)
Via client conne ction
Connected
Connected
Connected
NewZoneMast er(zonemastel P)
From zm
NewZoneM aster(zone masterIP)
Via clientco nnection
Connected

Appendix B

Recommendation for Ablemagic

Through our work in this thesis we have provided a design and an implementation of an MMORPG network engine. This can be utilized either as a hybrid P2P solution for cost saving, or as distributed architecture. The main problems with the P2P today is that users gain possibilities of cheating if the client software is modified and that their identities (IP addresses) are disclosed. In our opinion this is a too high risk to take when launching a new game. We believe that through research in the areas of network security and topologies solutions can be developed that solves these issues, but today we see no suitable solutions.

Our recommendation, based on Ablemagic being a newly started small game developer is to either use the solution developed in a distributed server architecture, or to utilize existing MMORPG networking software. To research the required solutions to enable P2P will likely demand too much resources to be a viable approach.

Of the existing software, Netdog and Project Darkstar appear to be the best suitable solutions for 'tisu. However as of today no game titles have been released using either of the engines, and neither have been tested during this thesis. This recommendation is based on the features of the engines, not on thorough analysis.
It should also be noted that the Java implementation of the design performs poorly, and a final server based on the design should be developed in another language like C++ which generally provides better performance.

We wish Ablemagic the best of luck with the further development, and the release of 'tisu. We hope our work has been helpful for both decision-making and as a base for development.