Building the Network Module for a Didactic Game Engine

Luiz Emmerich, Dennis Tanaka, Rodrigo Petriche, Felipe Kamakura, João Bernardes

Escola Politécnica da Universidade de São Paulo,
Department of Computer Engineering and Digital Systems, Brazil

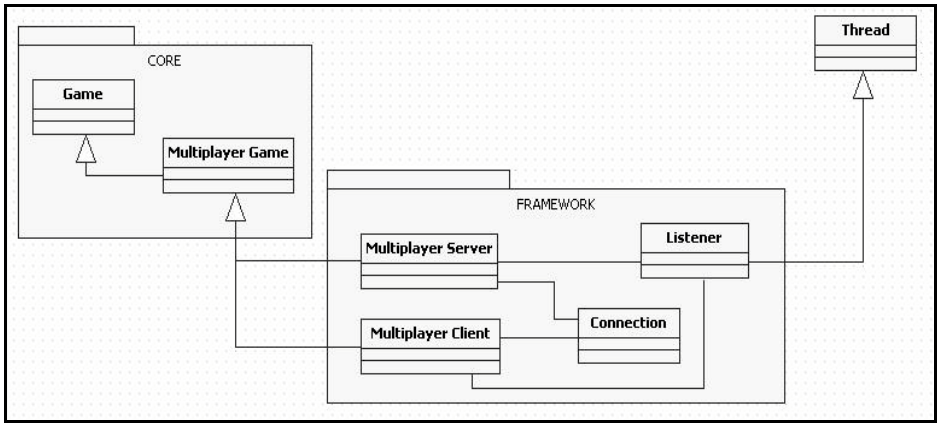

Figure 1 – The simple software architecture of the proposed module.

Abstract

This short paper presents the architecture of a network module being implemented for enJine, and aspects about its use in the development of massively multiplayer games (MMG). EnJine is a didactic game engine implemented in Java. The paper presents some partial results based on the implementation of a prototype architecture and application.

Keywords: networking, client-server, MMG, game engine, NIO

Authors' contact:
{luiz.emmerich,dennis.tanaka,rodrigo.petriche,felipe.kamakura,joao.bernardes}@poli.usp.br

1. Introduction

The main goal of the project described in this paper is the development of a network module for use in Multiplayer Online Games with future support to Massively Multiplayer Online Games. This component will be part of enJine, an open-source game engine developed by the Interactive Technologies Laboratory (INTERLAB) at University of Sao Paulo. This engine was written in the Java language and supports the creation of complete single player 3D games (using Java 3D) through a set of classes that abstract high level entities such as game stages and objects. EnJine and its use in teaching Computer Graphics are described in more detail by Tori et al. [2006].

Like most of enJine's implementation, the new network component must have low coupling with the implementation technology and offer appropriate resources for its use as a didactic tool.

Moreover it has some other functional requirements, such as:

- Support for multiple players and servers
- Game state synchronization
- Division of players in logical groups
- Cheat-proof architecture
- Support to different kinds of messages

2. Related Work

There are several published works discussing network architectures, if not for MMGs, for applications with similar requirements, such as collaborative virtual environments. Here only the works more closely related to this project are mentioned.

BERNARDES et al. [2003] show how networking is one of the most critical technical issues in a MMG. That is accomplished by discussing many related problems, such as the number of users, security and world persistence. Different network architectures are also presented, showing they are usually client-server, peer-to-peer (p2p) or a hybrid solution combining those two.

CALTAGIRONE [2002] presents an architecture for a MMORPG engine based in the client-server model that attempts to achieve six goals: security, maintainability, scalability, low network traffic, client application performance, and load balancing.

CECIN [2004] proposes a hybrid solution using both client-server and p2p models, and focusing on the security, scalability and fail recovery aspects.

In SMED [2002] many important techniques for improving networking are presented such as dead reckoning and interest management.

KO [2004] and LEE [2004] discuss the use of more than one server for large scale online games to assure the quality and the real time interaction between users. Adaptive server selection and server replication are shown as possible techniques to be used with these many servers.

3. Network Architecture

The choice of the network architecture on which the development of a MMG will be made is a fundamental step to the project of any game of this genre. It is this architecture that will define the effective maximum number of users connected to the game, define the security against "cheaters" and permit scalability to the game.

To choose one among the numerous existing architectures, some aspects have been taken in consideration:

- Maintenance, implementation cost and system performance (hardware and bandwidth basically);
- Security, especially against dishonest players (or cheaters);
- Scalability;
- Compatibility to the development of a wide range of multiplayer games, MMGs in special. A MMG requires an architecture capable of providing a way to host a large number of players playing at the same time and persistent worlds (a game environment where a player can leave at a time and when he comes back later, he will encounter the same world with only some changes made by the players or the game management). Persistent worlds are mainly related to the database architecture (which was not the focus of this project), but the database architecture is highly influenced by the network architecture, as the absence of a centralized server requires special methods of data persistence;
- Viability: in this case, how would be the integration with the enJine, which has its own requirements such as simplicity of use and implementation;
- Reliability and fault tolerance.

After consulting similar works, discussed in the last section, three possible network architectures were chosen for further study: peer-to-peer and client-server or a hybrid implementation, using both.

P2P offered the best aspects of cost for hardware, low consumption of bandwidth and scalability due to its distributed architecture. It presents an architecture capable of hosting millions of players simultaneously by transferring most of the processing role and bandwidth consumption to each one of the users. But there are some drawbacks on the utilization of this architecture to a MMG. Because each player's client has so much responsibility for the game data (about his own character and actions) that is produced and transmitted through the network, malicious can manipulate this data more easily, making such games potentially more vulnerable. Some research is being made to try to overcome this issue, but most of these works are far from trivial, failing the simplicity requisite.

The client-server model provides most of what was required previously such as security, simplicity and tolerance against cheating by being a centralized architecture. Its drawbacks were related to scalability and hardware and bandwidth requirements. Most of the commercial MMG are implemented using this architecture.

At this point, a hybrid solution seemed to be best suited to the problem. But there is a big drawback in this, which is its high complexity of implementation. By adding user manipulation of data to permit scalability and yet trying to keep redundancy to a reliable centralized server, a large amount of added synchronization and logic are required to make it a practical solution. Figure 2 shows a sample hybrid architecture proposed by Cecin [2004].

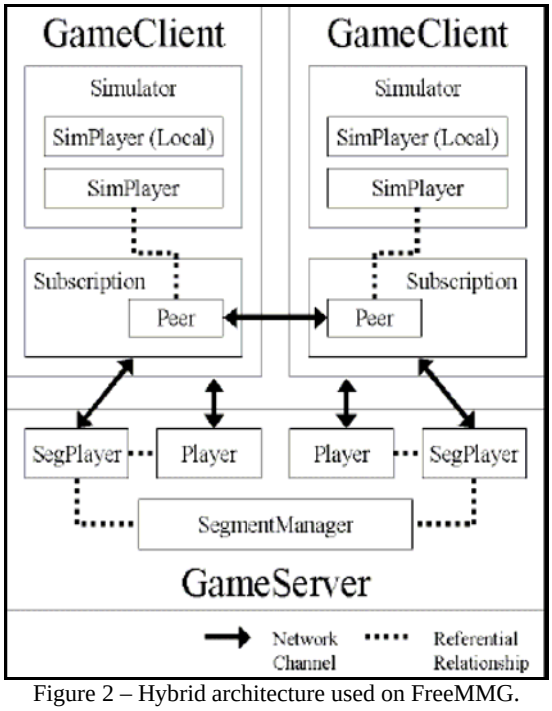

Figure 2 – Hybrid architecture used on FreeMMG.
Source: CECIN [2004].

After analyzing the advantages and disadvantages of each solution, the client-server model was chosen,

based on the analysis shown in table 1. That table expresses the criteria that were used to choose the best architecture for the project and the weight given to each one. Grades and weights were chosen based on the analysis of the related work discussed on section 2 of this paper, in a subjective way. Grades vary from 1 to 5 and a higher grade indicates a better option.

Table 1: Network architecture selection.

Criteria	Weight	P2P	Client / Server	Hybrid
Cost: Implementation, maintenance	**5**	5	2	3
Performance: Bandwidth consumption	**5**	5	2	4
Tolerance against unfair players:	**4**	1	5	3
Scalability:	**3**	5	3	4
Persistent worlds:	**5**	1	5	3
Viability:	**5**	2	5	2
Complexity:	**5**	3	4	1
Reliability and fault tolerance:	**3**	5	2	4
Total:		114	**125**	101

4. Software Architecture

By choosing the network architecture, some considerations about the software implementation and game logic were taken in account.

To achieve scalability, it was decided to keep each game area associated to one server at a time, due to the heavy utilization of game stages or areas (also known as dungeons in RPGs) on non-MMGs (the enJine network implementation, should be suited to the creation of games like that as well), and the possibility to create MMGs based on that concept as well. To let many servers take care of a single game area, additional logic would have to be implemented. Software complexity to synchronize data on servers would then grow to a point not desired by the developers and unfit for the simplicity required by games made with enJine.

The software architecture developed in this project and shown in Figure 1 consists basically of extensions from the existing `Game` class already present in enJine. The first one is the `MultiplayerGame` class. This class contains some functions related to the connection phase and data transmission that has to be implemented in the both client and server.

Aside from that, there are the `MultiplayerServer` and `MultiplayerClient` classes in the framework package, which inherit from `MultiplayerGame`. The client class encapsulates the functionality related to rendering, like the already implemented `SinglePlayerGame` class. This class treats the user input and sends it to the server as messages, and processes the server message to update the client state.

The `MultiplayerServer` class is responsible for the logical aspects of the game, such as the validation of user actions and collision detection. It has some server-related network functions like connection acceptance and message broadcasting to update the client games.

The `Connection` class represents the link between server and client, and stores the information of each point of the connection. It also has two-buffers to store messages that will be sent and the ones that arrive.

`Listener` is a thread that keeps going through all connections to check the arrival of new messages. When one is found, it is put in the correspondent connection's buffer to be treated in the future.

`Message` is an abstract class that currently must be implemented by the user. It contains the fields and methods necessary to mount and dismount messages for network communication.

All these classes have been included in the enJine framework package to free users from the need to consider several of the implementation aspects discussed above.

5. Development Tools

An important software tool used in the project is a Java package called New Input and Output (NIO). It consists of a number of classes designed specifically to provide high-performance I/O system. The package includes improvements in buffer management, scalable network and others. Most of all, it boosts performance and speed, greatly improving the efficiency of the java code [Hitchens 2002].

6. Results and Conclusions

As the project is still in development there are no final results yet. Until now a functional prototype has been built and the project's main decisions have already been made and tested to an extent.

The prototype is basically a game where the players move 3d models around the screen and all players sees it. It is a very simple game used only to assure that the requirements are accomplishable. The tests with it

were not yet finished but it has already proved useful, helping to make some of the important choices previously discussed in this document.

In the prototype the player uses the keyboard arrows to move a 3d model around the screen. Each action produced by the player generates a message to the game server which in its turn, processes the message and updates the state of the game, forwarding the results to all players. This way everyone sees the movement of all the models on the screen.

The only class not completely implemented in this prototype is the Message class. The messages in it are simple strings with fields such as type of message, destiny and data. These fields are separated by semicolons.

The next steps of the project are the tests regarding bandwidth consumption by the messages and processing time, and the implementation of a multiplayer game using the developed module to test its performance in real world applications.

Acknowledgements

Thanks to Romero Tori for the opportunity to work on this project and to Ricardo Nakamura for the valuable assistance on technical issues.

References

BERNARDES, J.L., TORI, R., JACOBER, E., NAKAMURA, R., BIANCHINI, R., 2003. A Survey on Networking for Massively Multiplayer Online Games. *In: Wjogos 2003.*

CALTAGIRONE, S., KEYS, M., SCHLIEF, B. AND WILLSHIRE, M.J., 2002. Architecture for a Massively Multiplayer Online Game Engine. *In: Proceedings of the Consortium for Computing Sciences in Colleges, December 2002.*

CECIN, F.R., MARTINS, M.G. AND BARBOSA, J.L.V., 2004. FreeMMG: A Hybrid Peer-to-Peer and Client-Server Model for Massively Multiplayer Games. *In: Proceedings of the SIGCOMM'04 Workshops, August 2004.*

LEE, K., KO, B. AND CALO, S. 2004. Adaptive Server Selection for Large Scale Interactive Online Games. *In: Proceedings of the NOSSDAV'04 Workshops, June 2004.*

KO, B., RUBENSTEIN, D., 2004. Distributed Server Replication in Large Scale Networks. *In: Proceedings of the NOSSDAV'04 Workshops, June 2004.*

Hitchens, R., 2002. Java NIO. O'Reilly.

SMED, J., KAUKORANTA, T. AND HAKONEN, H., 2002. A Review on Networking and Multiplayer Computer Games. Technical Report 454, Turku Centre for Computer Science.

TORI, R., BERNARDES JR., J. L. AND NAKAMURA, R., 2006. Teaching Introductory Computer Graphics Using Java 3D, Games and Customized Software: a Brazilian Experience, 2006. *In: Proceedings of Siggraph 2006 Educators Program, 30 July – 3 August 2006 Boston*. New York: ACM Press.