UDC 519.688

DESIGNING A SERVER-SIDE SYSTEM FOR MULTIPLAYER GAMES USING THE ENTITY-COMPONENT-SYSTEM PATTERN

D.V. Olshevsky
4rd year undergraduate student in
02.03.03 Mathematical support
and administering information systems
Kursk State University
e-mail: d.olshevskii@mail.ru

Scientific supervisor:
E.A. Zakharchuk
PhD in Linguistics, Docent,
Associate Professor
at the Department of Foreign Languages
and Professional Communication
Kursk State University
e-mail: elena_nezhura@mail.ru

The author of the article studies the development of a server-side system for a multiplayer strategy game using the Entity-Component-System (ECS) architectural pattern. This project aims to improve the performance and reliability of the system by ensuring efficient synchronization of client states, data processing, and game session management.

Key words: *multiplayer game, server-side system, Entity-Component-System, synchronization, game session management.*

Introduction

Currently, most multiplayer games use a centralized server that processes all client requests and synchronizes game data. This approach is not always optimal, as server optimization and flexibility in configuration remain at a low level. To improve system performance and reliability, this project proposes the use of the Entity-Component-System (ECS) architectural pattern, which allows for efficient management of game object states and logic [Booch et al. 2008]. The ECS pattern has been widely adopted in game development due to its ability to handle complex systems with high performance [Skypjack *http*].

The goal of this project is to develop the server-side of a strategy game that ensures reliable system operation, including client state synchronization, data processing, and game session management [Go Programming Language Documentation *http*].

In accordance with the goal, the following tasks were identified:

– to analyze existing solutions for the server-side of multiplayer games, consider architectural approaches, and formulate functional requirements [Vendrov 2003].

– to develop a conceptual model of the subject area, including class diagrams, sequence diagrams, state diagrams, and activity diagrams [Booch et al. 2008].

– to design component and deployment diagrams for the system, and implement the client interaction interface [Visual Paradigm Tutorials *http*].

Analysis of Information System Requirements

One of the main tasks in developing online game applications is creating a stable and high-performance server component that ensures efficient data synchronization between game clients and the server. In modern multiplayer games, it is important not only to support player interaction but also to guarantee the safety of user data and statistics, as well as ensure system security [Vendrov 2003]. The use of databases and secure communication protocols is essential for maintaining data integrity and preventing unauthorized access [Olypher V., Olypher N. 2016].

The development of the server-side for a multiplayer game requires consideration of many factors such as high performance, security, and system scalability. In this project, the server will ensure game state synchronization, user management, registration, and authentication, as well as the storage of game statistics and results. The implementation of these features is supported by the use of advanced algorithms and data structures, which are described in detail in [UML Specification *http*].

To ensure system security, user data protection and logging of user actions during the game are provided. The most important task is data synchronization between clients, which avoids discrepancies in game states and ensures the fairness of the game process [Olypher V., Olypher N. 2016]. The synchronization mechanism is implemented using efficient algorithms that minimize latency and ensure real-time updates [UML Specification *http*].

Additionally, the system will implement a mechanism for finding opponents for game sessions, allowing players to find one another based on criteria such as game level, session availability, and preferences. All user and statistical information will be stored in a database, ensuring security and scalability of the solution [Skypjack *http*]. The database design and management are based on best practices described in [Vendrov 2003].

Thus, the project is focused on designing an information system that not only ensures data synchronization and game session management but also supports reliable user interaction and a high level of optimization.

Functional Requirements

The game client information system must meet the following functional requirements:

1. Synchronization of game client states and the server.
2. Providing the necessary data to the client.
3. Finding an opponent for a game session.
4. Saving user statistics.
5. User registration and authentication.

The actors of the information system are presented in *Table 1*.

Table 1 – Actors of the Information System

Term	Meaning
Administrator	A user of the server application who manages the server, including its startup and initialization
Client	An external system of the client application interacting with the server
DBMS	Used for storing and processing user data, statistics, and game resources

The use case diagram is presented in *Figure 1*.

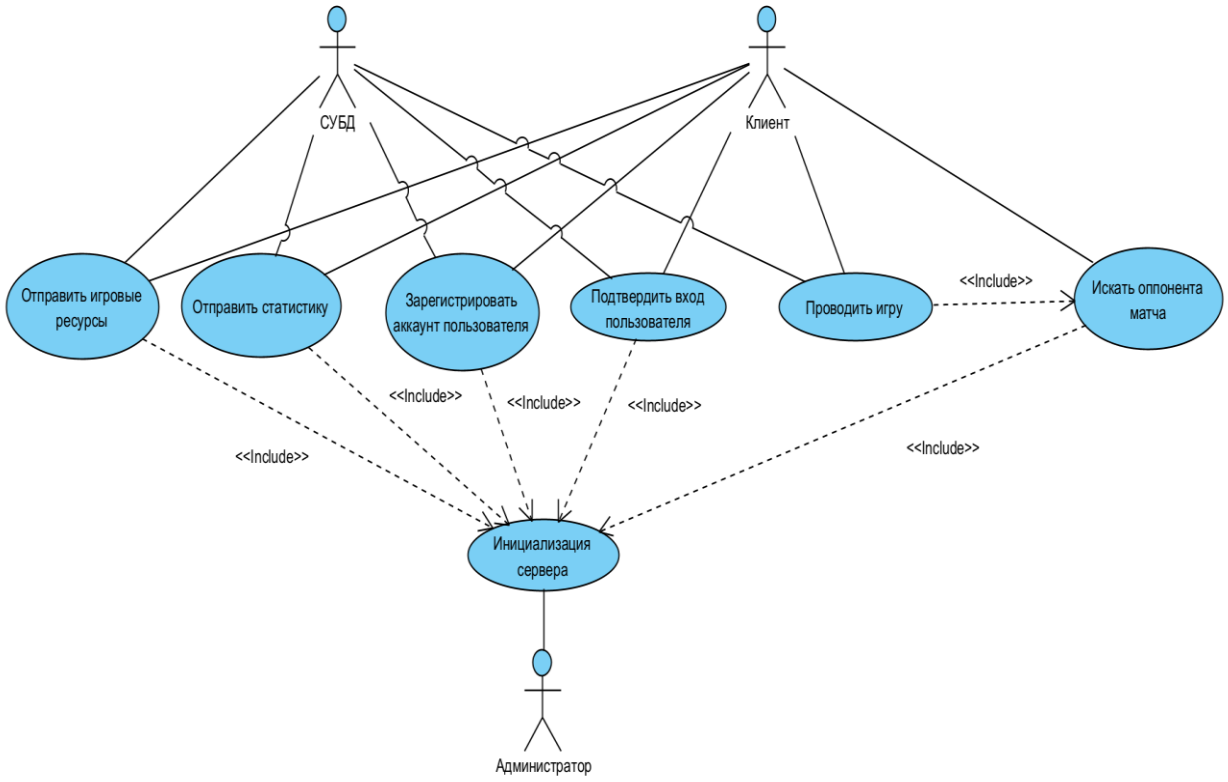

Figure 1 – Use Case Diagram

Textual Scenarios

Use Case «Initialize Server»

The process of preparing the server system for operation, including loading all necessary modules and data.

Regular Flow

Actor Actions	*System Actions*
1. Administrator starts the system.	2. The system loads the server configuration and necessary modules.
*Error Flow E1. Error when loading modules**	
	3. The system reports that the server is ready for operation.
6. The use case ends.	

****Error Flow E1. Error when loading modules***

Actor Actions	*System Actions*
	1. Display an error message about module loading failure.
2. Administrator debugs the module operation.	
3. The use case ends.	

Use Case «Register User Account»

This allows the client to create a new account in the system using an email and password.

Regular Flow

Actor Actions	*System Actions*
1. Client sends registration data (email, password) to the server.	2. The system checks the registration data in the database.
*Error Flow E1. Invalid registration data**	
	3. The system creates a new account and saves the data in the database.
	4. The system sends a successful registration result.
5. Client receives a response from the system about successful authorization.	
6. The use case ends.	

****Error Flow E1. False registration data***

Actor Actions	*System Actions*
	1. The system sends an error message about registration failure.
2. The application receives a response	

from the server about the error.
3. The use case ends.

Use Case «Authenticate User»

This allows the client to authenticate in the system using a login and password.

Regular Flow

Actor Actions	*System Actions*
1. Client sends authentication data to the server.	2. The system checks the authentication data in the database.
*Error Flow E1. Invalid authentication data**	
	3. The system returns a positive verification result and authenticates the user in the system.
4. The use case ends.	

****Error Flow E1. Invalid authentication data***

Actor Actions	*System Actions*
	1. The system sends an error message about authentication failure.
2. Client receives a response from the server about the error.	
3. The use case ends.	

Use Case «Find Match Opponent»

This allows the client to find opponents for a game session.

Regular Flow

Actor Actions	*System Actions*
1. Client sends a request to the server to find an opponent for a game session.	2. The system accepts the request and starts the process of finding an opponent for the game session.
*Alternative Flow A1. Error in the process of finding an opponent for the game session**	
	3. Activation of game systems and synchronization.
	4. Start of the game session.
5. The use case ends.	

****Alternative Flow A1. Could not find an opponent for the game session***

Actor Actions	*System Actions*
	1. The system sends an error message to the client.

2. The application receives a response from the server about the error.
3. The use case ends.

Use Case «Conduct Game»

This allows the client to conduct the game process for users.

Regular Flow

Actor Actions	*System Actions*
1. Client sends changes in the game session in the form of a special container.	2. Processing of game events in real-time.
	3. Checking the conditions for ending the game session

*Alternative Flow A1. The game session is not finished**
*Alternative Flow A2. The game session is finished***
4. The use case ends.

Alternative Flow A1. The game session is not finished

Actor Actions	*System Actions*
	1. The system sends the result of processing game events.
2. Client receives a response from the server about the game session state.	
3. The use case ends.	

****Alternative Flow A2. The game session is finished***

Actor Actions	*System Actions*
	1. The system ends the game session.
	2. The system updates user data in the database.
	2. The system sends information about the end of the game session to the client.
4. Client receives a response from the server about the game session state.	
5. The use case ends.	

Use Case «Send Statistics»

This allows the client to receive data about user statistics and progress.

Regular Flow

Actor Actions	*System Actions*
1. Client sends a request to receive	2. The system processes the request to

statistics.	receive statistics.
*Alternative Flow A1. Error in the process of sending statistics**	
	3. The system sends the statistics data to the client.
4. Client receives a response from the server.	
5. The use case ends.	

Alternative Flow A1. Error in the process of sending statistics

Actor Actions	*System Actions*
	1. The system sends an error message about the failure to execute the request.
2. Client receives a response from the server.	
3. The use case ends.	

Use Case «Send Game Resources»

This allows the client to receive game resources.

Regular Flow

Actor Actions	*System Actions*
1. Client sends a request to receive game resources.	2. The system processes the request to receive game resources.
*Alternative Flow A1. Error in the process of sending game resources**	
	3. The system sends the packed game resources to the client.
4. Client receives a response from the server.	
5. The use case ends.	

Alternative Flow A1. Error in the process of sending game resources

Actor Actions	*System Actions*
	1. The system sends an error message about the failure to execute the request.
2. Client receives a response from the server.	
3. The use case ends.	

The main objects of the system are presented in Table 2 below.

Table 1 – The main objects of the system

Term	Meaning
	A module that implements game logic and controls the process

Game session emulator	of conducting game sessions. It processes the actions of the players, checks the correctness of game moves, updates the game state and transmits the results of the game session to the server for saving.
Account	A container for storing user information during the registration process. It is required for transferring data to the database.
Player information	A container for storing information about the user during the game, such as the characteristics of the game character, etc. is necessary to verify the correctness of game actions by the server.
Game resources	A container for storing game resources and sending them to the client on request, such as a game card, etc.
Server configuration	A container for storing system startup parameters, necessary for server initialization

Completeness and Consistency Check

The correspondence of the use case diagram to the functional requirements is presented in Table 3.

Table 3 – Completeness Check

	Initialize Server	Send Game Resources	Send Statistics	Register User Account	Authenticate User	Conduct Game	Find Match Opponent
Synchronization	+					+	
Data Provision		+	+				
Opponent Search							+
Statistics Saving						+	
Registration & Authentication				+	+		

The consistency check of use cases is presented in Table 4.

Table 4 – Consistency Check

Use case	Objects				
	Account	Player Info	Game Session Emulator	Game Resources	Server Configuration
Initialize Server					1, 3
Send Game Resources				1, 3	
Send Statistics	1, 2, 3, 4				
Register User Account	1, 2, 3, 4				

Authenticate User	1, 2, 3, 4				
Conduct Game		2, 3, 4	1, 3, 4		
Find Match Opponent	2				

In Table 4, the following operations are denoted:

1 – Create;

2 – View;

3 – Update;

4 – Delete.

Design of the Information System

Based on the system description, the following relationships were identified:

– Server: The main object that contains tools for processing client requests, such as user registration, authentication, data sending, finding opponents for game sessions, and conducting game sessions.

– Client Connection: A set of tools for establishing and maintaining an internet connection between the Server and the client.

– Terminal: An interface that allows the administrator to make changes to the Server's operation.

– Game Session: Contains sets of components associated with Game Entities through Pools and Systems that modify components. This subsystem is necessary for implementing the game process emulation.

– Account: A container for transferring user data between modules.

– Player Info: A container for transferring the user's state in the game session.

Based on the technical specification, the attributes presented in Table 5 were identified.

Table 5 – Attributes of the Subject Area Objects

Class	Attributes
Server Configuration	System operation parameters
Account	User account data
Player Info	Information about user actions and characteristics during the game session
Game Resources	Container for game resources
Game Session Emulator	Information about the game session

Conclusion

During the project, the server side of the strategy game was designed using the Entity-Component-System (ECS) architectural template. The system provides state synchronization between game clients and the server, manages game sessions, stores user statistics, and provides user registration and

authentication functions. A conceptual domain model was developed to clearly structure and understand the logic of the system.

In conclusion, we can say that the project meets the stated functional requirements, reflects the relevance of the designed information system, including integration with the database, providing all the necessary features for the full functioning of the online game.

References

Booch, G., Rumbaugh, J., & Jacobson, I. The Unified Modeling Language User Guide. Moscow: DMK Press, 2008. ISBN 5-94074-334-X.

Go Programming Language Documentation [Website]. URL: https://go.dev/doc/ (accessed 17.02.2025).

Olypher, V., & Olypher, N. Computer Networks: Principles, Technologies, Protocols. St. Petersburg: Peter, 2016.

Skypjack. Entity-Component-System (ECS) Articles [Website]. – URL: https://skypjack.github.io/2019-02-14-ecs-baf-part-1/ (accessed 19.02.2025).

UML Specification. Official UML 2.5.1 Specification [Website]. – URL: https://www.omg.org/spec/UML/2.5.1/ (accessed 22.02.2025).

Vendrov, A. M. Designing Software for Economic Information Systems. – Moscow: Finance and Statistics, 2003. 347 p.

Visual Paradigm Tutorials [Website]. URL: https://www.visual-paradigm.com/tutorials/ (accessed 17.02.2025).

УДК 519.688

РАЗРАБОТКА СЕРВЕРНОЙ СИСТЕМЫ ДЛЯ МНОГОПОЛЬЗОВАТЕЛЬСКИХ ИГР С ИСПОЛЬЗОВАНИЕМ ШАБЛОНА ENTITY-COMPONENT-SYSTEM

Д.В. Ольшевский
студент 4 курса направления подготовки
02.03.03 «Математическое обеспечение и
Администрирование информационных систем»
Курский государственный университет
e-mail: d.olshevskii@mail.ru

Научный руководитель:
Е.А. Захарчук
кандидат филол. наук, доцент,
доцент кафедры иностранных языков
и профессиональной коммуникации
Курский государственный университет
e-mail: elena_nezhura@mail.ru

Автор статьи изучает разработку серверной системы для многопользовательской стратегической игры с использованием архитектурного шаблона Entity-Component-System (ECS). Целью данного проекта является повышение производительности и надежности системы за счет обеспечения эффективной синхронизации состояний клиентов, обработки данных и управления игровыми сессиями.

Ключевые слова: *многопользовательская игра, серверная система, Сущность-компонент-система, синхронизация, управление игровыми сессиями.*

Библиографический список

Вендров А.М. Проектирование программного обеспечения экономических информационных систем. М.: Финансы и статистика, 2003. 347 с.

Booch, G., Rumbaugh, J., & Jacobson, I. The Unified Modeling Language User Guide. Moscow: DMK Press, 2008. ISBN 5-94074-334-X.

Go Programming Language Documentation [Website]. URL: https://go.dev/doc/ (дата обращения: 17.02.2025).

Olypher, V., & Olypher, N. Computer Networks: Principles, Technologies, Protocols. St. Petersburg: Peter, 2016.

Skypjack. Entity-Component-System (ECS) Articles [Website]. – URL: https://skypjack.github.io/2019-02-14-ecs-baf-part-1/ (дата обращения: 19.02.2025).

UML Specification. Official UML 2.5.1 Specification [Website]. – URL: https://www.omg.org/spec/UML/2.5.1/ (дата обращения: 22.02.2025).

Visual Paradigm Tutorials [Website]. URL: https://www.visual-paradigm.com/tutorials/ (дата обращения: 17.02.2025).