D-ECS: Towards decentralising video games

Divi De Lacour

D-ECS: Towards decentralising video games

Divi De Lacour
Orange Innovation, IMT Atlantique, Inria
divi1.delacour@orange.com

Abstract

Current multiplayer video games are hard to scale in the number of players and simulated entities. Multiplayer architectures beyond basic client-server architectures are not provided by game engines and must be re-implemented for each game. Peer-to-peer approaches have been proposed to improve scalability, either to reduce the server network load created by broadcasts, or to create a scalable server infrastructure based on multiple machines. However, this approach increases the complexity of video game architectures. We provide a model to adapt games for p2p multiplayer, separating game and network logic. It allows for a game to be in a client-server or peer-to-peer setting seamlessly for any kind of game. We show how it can be applied to improve the performances and identify its security requirements and associated countermeasures.

Keywords: online video games, decentralization, entity component system, ECS, security

1 Introduction

The market for online games is experiencing significant growth, with revenues projected to reach 27.97 billion USD by 2024. This drives the growth of various applications, including metaverse use cases [6] and the simulation of multi-agent systems [5]. Multiplayer online games and massively multiplayer online games (MMOGs) are at the forefront of this expansion, offering complex and immersive experiences to players worldwide. They are to be distinguished from online social networks [14] which aim to provide communications without the ability to simulate a virtual environment. Multiplayer games can be designed using either centralized or decentralized networking architectures [19]. Centralized architectures, such as the traditional client-server model, are easier to design and offer robust protection against cheating. In these configurations, a central server manages game state and player interactions, ensuring consistency and security. However, centralized architectures come with higher hosting costs and can become bottlenecks as the number of players increases [19].

In contrast, decentralized architectures aim to reduce hosting costs by distributing the computational and networking load across multiple machines or even directly among players. While this approach can enhance scalability and reduce server costs, sensitive tasks are often delegated to untrusted devices, introducing significant challenges related to security and cheating [16].

One of the primary challenges in developing multiplayer games is scaling the number of simultaneous players and the number of entities that need to be simulated. A common solution to this problem is to distribute both execution and data. For instance, games like World of Warcraft (WoW) divide the game world into distinct areas or zones (Interest Management [11]), each managed by different servers. This approach allows for horizontal scaling, enabling the game to support a large number of players by distributing the load across multiple servers. However, such games are typically built using client-server architectures that can become complex and cumbersome. The mixing of game logic with distribution logic often results in a complicated server infrastructure, with for example the use of separate inventory, authentication, and area of interest servers.

Another significant challenge is the lack of a standard formalization for games, which complicates research and development efforts focused on game distribution. Most games are developed using object-oriented programming (OOP) with serialization, which can be difficult to synchronize and may introduce security vulnerabilities by introducing payloads. The Entity-Component-System (ECS) paradigm [17] offers a solution to these issues, separating data and code to be executed. ECS allows for efficient management of a large number of entities and is used in game engines like Bevy and Unity DOTS.

In this paper, we propose a model for flexible online video games that is adaptable to any ECS-based game and can operate in both centralized and decentralized architectures. Inspired by aspect oriented programming [9], our model separates game logic from networking logic, making it easier to scale and secure. We also study the security implications of our model and propose various optimizations to enhance performance.

The structure of this paper is as follows: Section 2 reviews the related works on ECS and decentralising video games, accounting for performances and security. Section 3 presents our proposed architecture. Section 4 identifies the security countermeasures available and how they can be added to our architecture.

2 Related Works

2.1 Formalising video games

There is no standard formalization for video games, which complicates research and development. Traditionally, most games are developed using object-oriented programming

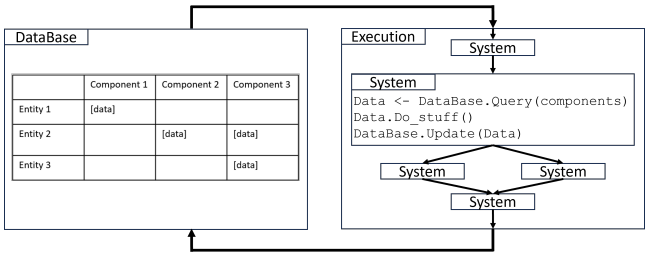

Figure 1. ECS illustration

(OOP). However, this approach limits the ability to use multithreading to improve performances and makes synchronization harder, as it involves both data and code.
Data-oriented game design treats the game as a database, focusing on efficient data management. A specific paradigm in data-oriented game design is the Entity-Component-System (ECS) architecture [17], which is used for managing a large number of entities, as seen in games like Cities: Skylines 2. Better performances have been observed in games developed with a data-oriented approach [18], which is enhanced by its ability to parallelize execution into multiple threads [13]. The ECS paradigm consists of three elements (figure 1):
Entities: Unique identifiers representing game objects. Entities themselves do not contain data or behavior; they are simply references that can be associated with various components.
Components: Data containers that store properties of entities, such as position, velocity, or health. Components are simple data structures and do not integrate any game logic.
Systems: Logic processors that execute game logic on entities based on their components. Systems iterate over entities with specific components and update their state accordingly.

2.2 Multiplayer video games

In multiplayer online games, clients exchange information about game objects in two approaches (figure 2):
Client-Server Model: a central server verifies actions and access rights. It is responsible for receiving updates from clients and broadcasting the new game status to all connected clients, ensuring consistency and security. The server infrastructure can be distributed into multiple servers communicating with each other.
Peer-to-Peer (P2P) Model: clients communicate directly with one another. They share information about the game environment without the need for a central server.
Scalability in multiplayer games encompass several critical challenges. Scalability of execution is a significant concern when the computational requirements of game logic exceed available resources, resulting in performance degradation. Additionally, the scalability in the number of simulated entities poses a challenge, as the management of an excessive volume of data related to game objects can overwhelm system capabilities and hinder responsiveness. Furthermore, the

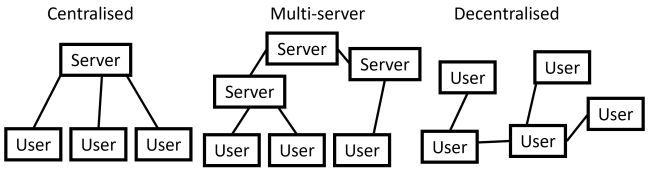

Figure 2. Multiplayer architectures

scalability in the number of players complicates the architecture, as broadcasting status updates within constrained timeframes becomes increasingly difficult with a growing player base. A promising solution to these scalability issues is distribution, specifically through the segmentation of the game into multiple semi-independent worlds of interest, a methodology referred to as Interest Management. This strategy enables horizontal scaling, allowing the system to accommodate increased loads by distributing the processing across multiple servers.

2.3 Peer-to-Peer approaches

Peer-to-peer (P2P) approaches, as surveyed in [19], offer a cost-effective alternative to traditional client-server models by distributing the computational and networking load among multiple peers. These approaches can either involve a full P2P model with no central server or a hybrid model where a federation of servers handles different parts of the game world. An example of the latter is Second Life, which operates on a vast network of servers. While P2P architectures can significantly reduce server costs and improve scalability, they require the game to be designed with specific distribution strategies in mind, making them difficult to change and adapt. Additionally, P2P models pose significant security challenges, as any peer can potentially illegally read and modify game variables.

2.4 Interest management

Interest management [11] improves scalability in multiplayer online games by dividing the game world into distinct zones, each managed by different servers, this reduces the number of entities each player or server needs to track. This distribution of data, bandwidth, and execution load reduces the load on individual servers.
As illustrated in figure 3, players are dispatched into different zones, each handled by a different server. Servers communicate with each other to handle the transfer of players between zones. Distinct servers handle the players authentication and manage the zone servers.
Strategies for interest management include static zoning, with predefined regions, and dynamic zoning, which adapts to the current distribution of players and entities[19].
However, adapting an already existing game to interest management is challenging as the game logic and distribution logic are mixed in the game architecture.

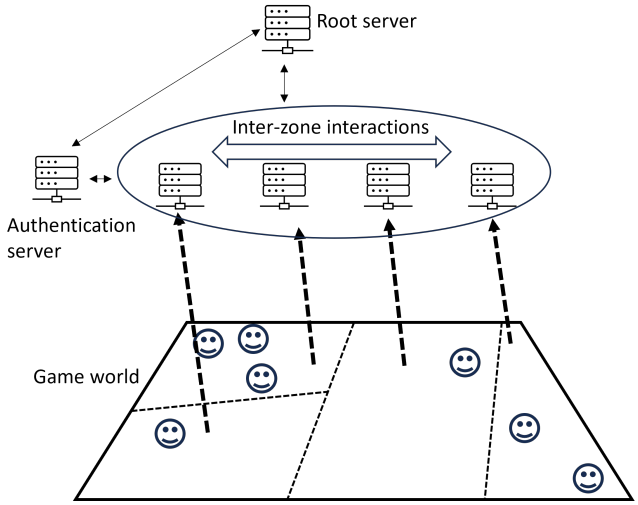

Figure 3. Interest management illustration

2.5 Security of online games

Online games are vulnerable to various types of cheating and security threats [16]. Unauthorized writing of values can lead to incorrect game execution and corrupted game data, undermining the integrity of the game. Unauthorized reading of values can compromise player privacy and provide unfair advantages.
Some other cheating techniques such as aim bots and modified controllers exploit automation on the user side, they use legal game actions to gain an edge over other players. Players can also use design errors in the game logic (bugs) to gain an unfair advantage.
Network attacks are also a significant concern. Withholding information during status dissemination can disrupt the game's state synchronization, and finding other players' IP addresses can lead to targeted attacks.

3 The D-ECS model

We extend the ECS model for multiplayer games with the D-ECS model, making it flexible for either client-server or full p2p online games. It can be compared to real-time database synchronisation [12]. This allows to easily port existing games (written with the ECS paradigm) into multiplayer and to separate game and networking logic for any game engine; To allow developers to focus on the core game development first and on multiplayer security and optimizations later, as it often happens in game development processes.

3.1 Proposed model

Figure 4 shows our proposed architecture:

- **ECS engine**: is the engine used when creating the game.
- **Executor**: executes systems in a function like approach: *function (current_state, action) -> new_world_state*, this makes it compatible with FaaS approaches [10].
- **Network component**: is in charge of the game state synchronization and network security
- **Distribution engine**: defines the policy on the distribution and synchronization and replication, it handles conflicts in values. In practice, it sets the synchronization as specific systems run in the ECS Engine, and regularly calls them.

We keep the existing ECS engine as it may have its own database optimizations[12].
Developers can set behavior for distribution (e.g. scope for interest management areas, regularity of synchronization) and security (e.g. data access rights) of entities by adding components (meta-components) which are parsed by the decentralization systems. This allows to change the multiplayer policy without changing the logic of the game handled by normal components. This way, systems can be limited to a specific scope to reduce execution load. Specific values can also be anticipated by a client before the network update as in dead reckoning [3] which is used to anticipate the trajectories of objects in distributed virtual environments.

3.2 Performance optimization

The presented architecture can be extended for improved performances
Decentralizing the architecture of multiplayer games can help improve the performances by helping scaling and resilience.
For that, the network component can utilize either a centralized server or decentralized pub/sub systems like Libp2p pubsubs [2] to propagate updates or decentralized databases such as OrbitDB [1] hosting the entity-component data.
Scope composition allows to dynamically handle the load change in the different scopes. It can be done hierarchically with scopes featuring sub-scopes. The composition does not rely on spatial assumptions (such as a world being in 2D or 3D world), this allows for greater flexibility.
The distribution manager is in charge of fusing or spliting the scopes. It updates the entities scopes. It must make sure no entity is lost because no peer monitored it.
Inside/outside interface – In some cases, interactions between two scopes can be very limited. For instance, the precise positions of planets within a stellar system can be disregarded when viewed from another stellar system, which can be treated as a single point with all its mass concentrated. As illustrated in Figure 5, developers can provide a specific API to interact with a scope as a whole. This requires creating an entity symbolizing the entire scope, which has an *is_an_interface* component. Only one such entity can represent a scope, it can be present in multiple scopes and must be present in the scope it represents.

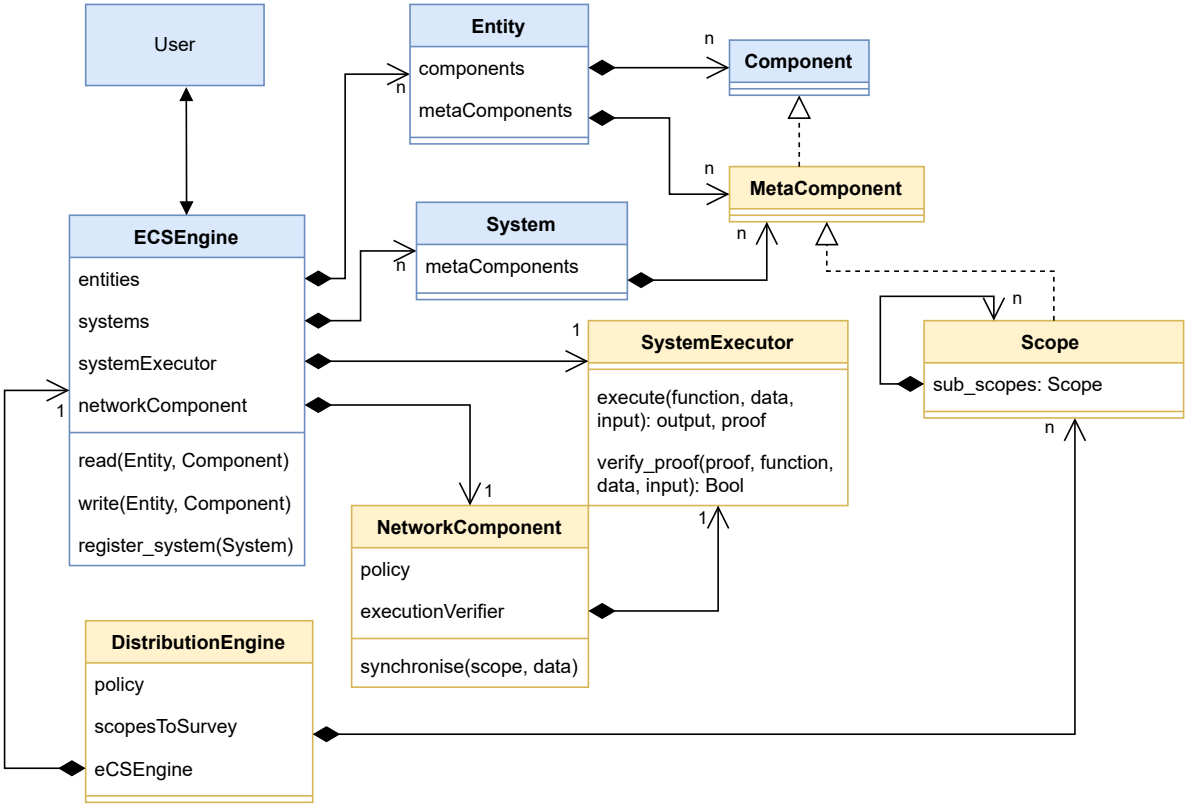

Figure 4. Proposed architecture

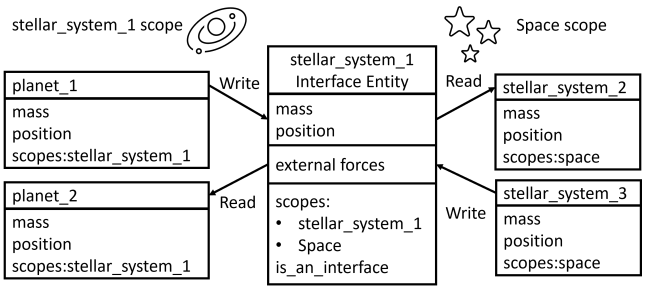

Figure 5. Interface for stellar system isolation

Specific systems must be added to update the interface entity, both from inside and outside. For example, in a stellar system simulated using mechanics: **From inside**: Update its mass and position components by processing all planets. **From outside**: receive and aggregate the external forces from other stellar systems.

4 Security countermeasures

We review here the available security countermeasures for our architecture:

Proactive approaches aim to prevent attacks: Access rights policies are applied to the game status, for reading and writing. They are applied by the network component that may refuse to share certain data (entities or components) or restrict writing permissions [8]. Additionally, it can verify the result of a system execution with the system executor.
Network security – Techniques such as mixnets and onion routing [4] can help hide IP addresses to protect user privacy. Network resilience can be bolstered by employing p2p networks, which distribute data across multiple nodes, reducing the risk of failure and improving overall connectivity.
TEEs [15] provide a secure execution environment within a processor, ensuring that code and data loaded inside are protected with respect to confidentiality and integrity. They can be used in the system executor to guarantee the integrity of system execution and game data privacy.
Reactive countermeasures aim to detect and limit the impact of on-going attacks:
Reputation systems [7] can be set in the network component where peers verify systems executions to guarantee the game status integrity.
Behavior monitoring based on AI systems can be set to detect cheating that is legitimate from the execution flow point of view (e.g. aim bots, modified controllers).

Bug exploitation can be limited by regular monitoring by the game developpers, patching the game to keep it equilibrate.

5 Conclusion

We proposed here a model for customizable online games. We showed how performances can be improved by segmenting the game trough interest management and distributed in a decentralised way. We identified the security properties and countermeasures. Remains the question of trade-offs between performances and security. For that, an implementation an benchmarking of D-ECS is needed.

References

[1] [n. d.]. OrbitDB - Home. https://orbitdb.org/

[2] Pedro Agostinho, David Dias, and Luís Veiga. 2022. SmartPubSub: Content-based Pub-Sub on IPFS. In *2022 IEEE 47th Conference on Local Computer Networks (LCN)*. 327–330. https://doi.org/10.1109/LCN53696.2022.9843795 ISSN: 0742-1303.

[3] Youfu Chen and Elvis S. Liu. 2018. Comparing Dead Reckoning Algorithms for Distributed Car Simulations. In *Proceedings of the 2018 ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*. ACM, Rome Italy, 105–111. https://doi.org/10.1145/3200921.3200939

[4] Roger Dingledine, Nick Mathewson, and Paul Syverson. [n. d.]. Tor: The Second-Generation Onion Router. ([n. d.]), 18.

[5] Alexey Dosovitskiy, German Ros, Felipe Codevilla, Antonio Lopez, and Vladlen Koltun. 2017. CARLA: An Open Urban Driving Simulator. In *Proceedings of the 1st Annual Conference on Robot Learning*. PMLR, 1–16. https://proceedings.mlr.press/v78/dosovitskiy17a.html ISSN: 2640-3498.

[6] Andreas Haeberlen, Linh Thi Xuan Phan, and Morgan McGuire. 2023. Metaverse as a Service: Megascale Social 3D on the Cloud. In *Proceedings of the 2023 ACM Symposium on Cloud Computing*. ACM, Santa Cruz CA USA, 298–307. https://doi.org/10.1145/3620678.3624662

[7] Ferry Hendrikx, Kris Bubendorfer, and Ryan Chard. 2015. Reputation systems: A survey and taxonomy. *J. Parallel and Distrib. Comput.* 75 (Jan. 2015), 184–197. https://doi.org/10.1016/j.jpdc.2014.08.004

[8] Vincent C Hu. 2024. *Access control on NoSQL databases.* Technical Report NIST IR 8504. National Institute of Standards and Technology (U.S.), Gaithersburg, MD. NIST IR 8504 pages. https://doi.org/10.6028/NIST.IR.8504

[9] G. Kiczales. 1996. Aspect-oriented programming. *Comput. Surveys* 28, 4es (Dec. 1996), 154. https://doi.org/10.1145/242224.242420

[10] Yongkang Li, Yanying Lin, Yang Wang, Kejiang Ye, and Chengzhong Xu. 2023. Serverless Computing: State-of-the-Art, Challenges and Opportunities. *IEEE Transactions on Services Computing* 16, 2 (March 2023), 1522–1539. https://doi.org/10.1109/TSC.2022.3166553

[11] Elvis S. Liu and Georgios K. Theodoropoulos. 2014. Interest management for distributed virtual environments: A survey. *Comput. Surveys* 46, 4 (April 2014), 1–42. https://doi.org/10.1145/2535417

[12] Alessandro Margara, Gianpaolo Cugola, Nicolò Felicioni, and Stefano Cilloni. 2023. A Model and Survey of Distributed Data-Intensive Systems. *Comput. Surveys* 56, 1 (Aug. 2023), 16:1–16:69. https://doi.org/10.1145/3604801

[13] Moreno Marzolla and Gabriele D'Angelo. 2020. Parallel Data Distribution Management on Shared-memory Multiprocessors. *ACM Transactions on Modeling and Computer Simulation* 30, 1 (Jan. 2020), 1–25. https://doi.org/10.1145/3369759

[14] Newton Masinde and Kalman Graffi. 2020. Peer-to-Peer-Based Social Networks: A Comprehensive Survey. *SN Computer Science* 1, 5 (Sept. 2020), 299. https://doi.org/10.1007/s42979-020-00315-8

[15] Bohdan Trach, Oleksii Oleksenko, Franz Gregor, Pramod Bhatotia, and Christof Fetzer. 2019. Clemmys: towards secure remote execution in FaaS. In *Proceedings of the 12th ACM International Conference on Systems and Storage*. ACM, Haifa Israel, 44–54. https://doi.org/10.1145/3319647.3325835

[16] Steven Daniel Webb and Sieteng Soh. [n. d.]. Cheating in networked computer games – A review. ([n. d.]).

[17] Dennis Wiebusch and Marc Erich Latoschik. 2015. Decoupling the entity-component-system pattern using semantic traits for reusable realtime interactive systems. In *2015 IEEE 8th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS)*. IEEE, Arles, France, 25–32. https://doi.org/10.1109/SEARIS.2015.7854098

[18] David Wingqvist, Filip Wickström, and Suejb Memeti. 2022. Evaluating the performance of object-oriented and data-oriented design with multi-threading in game development. In *2022 IEEE Games, Entertainment, Media Conference (GEM)*. 1–6. https://doi.org/10.1109/GEM56474.2022.10017610

[19] Amir Yahyavi and Bettina Kemme. 2013. Peer-to-peer architectures for massively multiplayer online games: A Survey. *Comput. Surveys* 46, 1 (Oct. 2013), 1–51. https://doi.org/10.1145/2522968.2522977