Entity Component Systems Usefulness, Literature Review

Carson Webster

Abstract—Entity-Component-System (ECS) architecture is a relatively new software design pattern that has gained widespread popularity in the gaming industry for its flexibility and performance benefits. ECS architecture favors composition over inheritance, allowing for greater flexibility by defining an entity by its traits rather than using a fixed inheritance tree. This allows for seamless changes during simulation, and makes ECS well-suited for co-simulation applications.
This paper reviews four recent studies that explore the application of ECS architecture in various fields beyond gaming, such as co-simulation, energy service interfaces, interaction programming, and realtime interactive systems. These studies demonstrate the potential benefits of using ECS architecture outside of gaming and highlight its applicability in diverse fields.

Index Terms—Entity Component Systems,

1 Introduction

Enity-Component-System (ECS) architecture is a relatively new software design pattern that has gained widespread popularity in the gaming industry due to its flexibility and performance benefits. ECS separates an object's data and behavior into two distinct components: entities and components. An entity is an object in the game world, while a component is an attribute or behavior of that entity. This architecture provides a framework for managing these entities and components, allowing for a more efficient and organized development process.

ECS offers several advantages over other software design patterns. Firstly, ECS architecture allows for easy modification and extension of code, which is crucial for game development where requirements are constantly changing. Secondly, ECS offers better performance than other design patterns due to its focus on data-oriented design, which allows for better memory usage and cache locality. Thirdly, ECS architecture favors composition over inheritance, allowing for greater flexibility by defining an entity by its traits rather than using a fixed inheritance tree.

The objectives of this paper are to provide an overview of the ECS architecture, discuss its advantages over other software design patterns, and highlight its use in game development. Additionally, the paper aims to provide a comprehensive literature review of ECS, and to discuss its current state of development and future potential. The paper is organized as follows: section II dives into the literature review of entity component systems in practical applications, section III discusses the technical theory behind core concepts of ECS and its implementation, and section IV concludes the paper and summarizes the key findings.

1.1 Definition of Entity Component Systems (ECS) architecture

Entity Component Systems are an architectural pattern used in software development that separates an object's data representation (components) from its behavior (systems). In ECS architecture, entities are the fundamental building blocks that represent the game objects or entities in the game world. Entities are composed of a collection of components that store the data needed to define the entity's properties and characteristics.

Components in ECS architecture are responsible for representing the data of an entity. Each component defines a single aspect of the entity, such as its position, health, or velocity. Systems, on the other hand, are responsible for performing operations on components to implement the behavior of entities. Systems operate on a set of components that match a specific set of criteria, called an archetype. This approach allows for efficient processing of large numbers of entities with similar properties, making it particularly suited for game development and simulation.

1.2 Advantages of ECS over other software design patterns

One of the main advantages of using ECS over other software design patterns is the improved scalability and flexibility it offers. Traditional object-oriented programming (OOP) approaches can become increasingly complex and difficult to manage as a system grows and more features are added. In contrast, ECS separates the data and behavior of entities, allowing for easier modification and addition of new features without the need to modify existing code. This makes it easier to maintain and expand the system over time, especially for large and complex applications.

Another advantage of ECS is the potential for improved performance in certain types of applications. Since ECS stores data in a contiguous block of memory, it can be more cache-friendly and reduce memory fragmentation, leading to faster access times and improved performance. Additionally, ECS can facilitate parallel processing and multithreading, allowing for better use of modern hardware and faster execution times. However, it should be noted that the performance gains from ECS are not always guaranteed and depend on the specific application and implementation.

1.3 Overview of the literature review

The literature review for this paper focuses on the use of ECS in game development. It covers various aspects of ECS, including its history, implementation, and best practices. The literature review also discusses the advantages and disadvantages of ECS, as well as its impact on game development.

1.4 Organization of the paper

The paper is organized as follows. Section II we dive into the literature review of entity component systems in practical applications. Section III discusses the technical theory behind core concepts of ECS and it's implementation. Section IV concludes the paper and summarizes the key findings.

2 Literature Review

2.1 Search and Research Process

In order to gather relevant literature on the topic of entity component systems (ECS), a search was conducted on Web of Science, focusing on highly cited and referenced papers. Four papers were selected based on their relevance to the topic and the popularity of their cited works.

The first paper, authored by Hatledal et al. [1], presents Vico, an ECS-based co-simulation framework. The second paper, written by Lange et al. [2], discusses the use of wait-free hash maps in the ECS pattern for realtime interactive systems. Raffaillac and Huot [3] describe in their paper the application of the ECS model to interaction programming. Finally, Slay et al. [4] propose an application for an ECS in an Energy Services Interface. These papers were chosen because they provide a comprehensive overview of the ECS architecture and its applications in different domains.

2.2 Analysis of the papers

2.2.1 Vico: An entity-component-system based co-simulation framework

This paper presents a co-simulation framework called Vico, built on the Entity-Component-System (ECS) architecture. The framework supports the Functional Mock-up Interface and System Structure and Parameterisation standards, and allows for easy integration of various systems. Vico is compared to four similar co-simulation frameworks by simulating a quarter-truck system.

The ECS architecture employed by Vico allows for greater flexibility through composition over inheritance. Entities are defined by their traits, which can be changed seamlessly during simulation. This architecture allows for easy integration of physics engines, plotting, 3D visualization, co-simulation masters, and other types of systems in a modular way.

2.2.2 Wait-Free Hash Maps in the Entity-Component-System Pattern for Realtime Interactive Systems

This paper explores the use of high-performance wait-free hash maps for the system access of components within the Entity-Component-System (ECS) pattern in Realtime Interactive Systems (RIS). The authors present centralized and decentralized approaches for reducing the memory demand of these memory-intensive wait-free hash maps, which can lead to highly responsive, low-latency data access while maintaining a consistent data state. Their implementation of the new method in a current RIS shows that their approach is able to efficiently reduce the memory usage of wait-free hash maps by more than a factor of ten while still maintaining high performance.

The authors derive best practices from their numerical results for different use cases of wait-free hash map memory management in diverse RIS applications. This paper expands on the ECS architecture by providing a solution for efficient memory management in RIS applications that require highly responsive, low-latency data access. The presented approaches can be integrated into existing ECS-based architectures and can potentially enhance their performance and memory usage.

2.2.3 Applying the Entity-Component-System Model to Interaction Programming

This paper proposes a new GUI framework based on the Entity-Component-System (ECS) model for interaction programming. The framework allows interactive elements (Entities) to acquire any data (Components) and behaviors are managed by continuously running processes (Systems) which select entities based on the components they possess. This approach enables the handling and reuse of behaviors and allows for the global definition of interaction modalities as a set of systems. The authors implement their approach in an experimental toolkit called Polyphony and demonstrate its use with a sample application. The paper provides a detailed explanation of their interpretation of the ECS model in the context of GUIs.

2.2.4 Proposed Application for an Entity Component System in an Energy Services Interface

This paper proposes the adoption of an Entity Component System (ECS) framework to address the needs of an Energy Service Interface (ESI). While ECSs have been well-established in the video game industry for their performance benefits, they have not been widely examined or adopted outside of that industry. The paper examines the needs of an ESI and provides an overview of open-source ECS libraries, as well as preliminary performance results for ECSs. Additionally, the traditional approach of fulfilling the needs of an ESI with database architectures is explored.

By adopting an ECS framework, the paper argues that the ESI can benefit from the flexibility and modularity provided by the ECS architecture. The paper highlights the potential for ECSs to facilitate the creation and management of complex systems, as well as improve performance through data-oriented design. The paper concludes by discussing the potential limitations and challenges of adopting an ECS framework for an ESI, and suggests future directions for research in this area.

2.3 Findings and commonalities

My review of the four papers on entity-component systems highlights several commonalities and findings. All four papers propose using entity-component systems in different applications to improve performance, flexibility, and modularity.

In particular, the first paper introduces a co-simulation framework built on an entity-component system architecture, which allows for the seamless integration of various systems into the framework. The second paper presents a high-performance wait-free hash map for system access of components in real-time interactive systems, which maintains a consistent data state while allowing for non-locking read and write operations. The third paper introduces a GUI framework based on the entity-component system model, where behaviors are managed by continuously running processes that select entities by the components they possess, facilitating the handling and reuse of behaviors. Finally, the fourth paper proposes adopting an entity-component system framework to serve the needs of an Energy Service Interface.

Despite the different applications and focuses of each paper, there are several commonalities. Firstly, all four papers highlight the advantages of using an entity-component system architecture, which allows for greater flexibility and modularity compared to traditional inheritance-based architectures. Secondly, all papers emphasize the importance of performance in their respective applications, with each proposing solutions to improve performance using entity-component systems. Finally, all papers demonstrate the potential of entity-component systems to improve and streamline various applications beyond just the video game industry where it was first popularized.

Overall, the findings from this literature review suggest that entity-component systems can be applied in various domains beyond video games to improve performance and increase flexibility and modularity.

2.4 Core concepts of ECS architecture

Entity Component System (ECS) is a software architecture pattern that separates the representation of objects into entities composed of independent components. The three main concepts of ECS architecture are entities, components, and systems.

2.4.1 Entities, components, and systems

Entities, components, and systems are the three core concepts of ECS architecture. An entity is an object or a concept that exists within the game world and can be manipulated by the player or the game itself. In video games, an entity can be a character, a weapon, a vehicle, or any other object that has properties and behaviors. For example, in a racing game, a car can be an entity with properties like speed, acceleration, and handling.

Components are the building blocks of an entity and represent the various properties or attributes of an entity. Each component encapsulates a specific piece of data or functionality. For example, in a platformer game, a character entity can have a sprite component for its graphical representation, a physics component for its movement, and an audio component for its sound effects.

Systems are the logic that governs the behavior of entities. They are responsible for processing and updating the components of entities based on certain conditions or rules. For example, in a strategy game, an AI system can be responsible for making decisions for the enemy units based on the current game state. Another example is a collision detection system that checks for collisions between entities and updates their components accordingly.

In ECS architecture, entities are made up of components and their behavior is controlled by systems. By separating the entities, components, and systems into distinct modules, ECS allows for greater flexibility and modularity in game development.

2.4.2 Composition over inheritance

Composition over inheritance is a design principle that favors the use of composition (i.e., building complex objects by combining simpler ones) instead of inheritance (i.e., creating new classes by extending existing ones). In the context of ECS architecture, composition is used to build entities from components rather than inheriting properties from a base class.

This principle promotes flexibility and modularity in code, as it allows for the creation of entities with specific functionalities by combining relevant components. For example, a player entity in a game can be composed of a position component, a sprite component, a physics component, and an input component. Instead of inheriting all these properties from a generic entity class, the player entity is created by combining these specific components.

Composition also reduces coupling between classes, making it easier to change or replace individual components without affecting the rest of the system. This can improve the overall maintainability and scalability of the codebase. Additionally, composition enables the creation of entities with dynamic properties, as components can be added or removed during runtime based on the needs of the game. Composition over inheritance is a design principle that favors the use of composition (i.e., building complex objects by combining simpler ones) instead of inheritance (i.e., creating new classes by extending existing ones). In the context of ECS architecture, composition is used to build entities from components rather than inheriting properties from a base class.

This principle promotes flexibility and modularity in code, as it allows for the creation of entities with specific functionalities by combining relevant components. For example, a player entity in a game can be composed of a position component, a sprite component, a physics component, and an input component. Instead of inheriting all these properties from a generic entity class, the player entity is created by combining these specific components.

Composition also reduces coupling between classes, making it easier to change or replace individual components without affecting the rest of the system. This can improve the overall maintainability and scalability of the codebase. Additionally, composition enables the creation of entities with dynamic properties, as components can be added or removed during runtime based on the needs of the game. Advantages of ECS architecture include improved performance, scalability, and modularity. It also makes it easier to implement new features and modify existing ones.

2.5 Implementation of ECS architecture in software design

2.5.1 Design patterns for ECS architecture

There are several design patterns that are commonly used in ECS architecture, each with its own benefits and tradeoffs. One such pattern is the Singleton pattern, which is often used for managing global resources in the ECS. This pattern involves creating a single instance of a class that provides access to a shared resource, such as a renderer or an audio system. Another common pattern is the Factory pattern, which involves creating specialized classes that can create and initialize entities or components.

Other patterns used in ECS architecture include the Observer pattern, which is used for event-based communication between systems and components, and the Flyweight pattern, which is used to reduce memory usage by sharing common data between entities or components. The Decorator pattern is also used to add additional functionality to entities or components without modifying their underlying implementation.

Choosing the right design pattern for a particular use case can have a significant impact on the overall performance and maintainability of an ECS-based system. It is important to consider factors such as the size and complexity of the system, the type of game being developed, and the performance requirements when selecting a pattern.

2.5.2 Best practices for ECS implementation

When implementing ECS architecture, it is recommended to keep systems and components as decoupled as possible to avoid dependency issues. It is also important to keep the entity framework lightweight, with only necessary data stored in entities. Additionally, using an event-driven approach to communication between systems can improve performance and maintainability. Lastly, regularly testing and profiling the system can help identify potential issues and improve overall performance.

2.5.3 Tools and libraries for ECS development

Several tools and libraries are available for ECS development, including Bevy, an open-source ECS game engine built on Rust. Bevy provides a flexible and efficient way to build games and other interactive applications using ECS architecture. Other popular ECS libraries include Unity's Entity Component System (ECS), Artemis-odb, and Entit.

3 Conclusion

3.1 Summary of the main findings

In this paper, we provided an overview of entity component systems (ECS) architecture, highlighting its advantages over other software design patterns. We explored the core concepts of ECS architecture, including entities, components, and systems, and discussed the benefits of using composition over inheritance. We also examined the implementation of ECS architecture in software design, discussing common design patterns, best practices for implementation, and available tools and libraries.

3.2 Contributions of the paper to the field of software design

This paper makes several contributions to the field of software design. First, it provides a comprehensive overview of ECS architecture, making it accessible to developers who may be new to this approach. Second, it highlights the advantages of ECS over other design patterns, demonstrating how it can lead to more modular, maintainable, and scalable software. Finally, it provides practical guidance for implementing ECS in software design, drawing on best practices and available tools and libraries.

3.3 Limitations of the study and future research directions

While this paper provides a thorough overview of ECS architecture and its implementation in software design, there are some limitations to our study. First, we focused primarily on the design and implementation aspects of ECS and did not delve into performance considerations. Second, we only examined a few available tools and libraries for ECS development, and there may be others that could also be useful.

Future research could explore these limitations in greater detail, as well as investigate additional aspects of ECS architecture and its use in software design. For example, researchers could investigate the impact of ECS on software performance and explore strategies for optimizing ECS-based systems. Additionally, further research could examine the use of ECS in specific application domains, such as game development, to better understand its strengths and limitations.

3.4 Final thoughts and recommendations

In conclusion, entity component systems offer a promising approach to software design that can lead to more modular, maintainable, and scalable code. By understanding the core concepts of ECS architecture and following best practices for implementation, developers can take advantage of the benefits of this approach while avoiding common pitfalls. We recommend that developers who are new to ECS explore available tools and libraries, such as Bevy, and experiment with different design patterns to find what works best for their specific use cases. With further research and development, ECS architecture could become an increasingly important approach to software design in a variety of domains.

References

[1] L. I. Hatledal, Y. Chu, A. Styve, and H. Zhang, *"Vico: An entity-component-system based co-simulation framework,"* Simulation Modelling Practice and Theory, vol. 108, p. 102243, 2021.

[2] P. Lange, R. Weller, and G. Zachmann, *"Wait-free hash maps in the entity-component-system pattern for Realtime Interactive Systems,"* 2016 IEEE 9th Workshop on Software Engineering and Architectures for Realtime Interactive Systems (SEARIS), 2016.

[3] T. Raffaillac and S. Huot, *"Applying the entity-component-system model to interaction programming,"* Proceedings of the 30th Conference on l'Interaction Homme-Machine, 2018.

[4] T. Slay, G. B. Spitzer, and R. B. Bass, *"Proposed application for an entity component system in an Energy Services Interface,"* 2022 IEEE Conference on Technologies for Sustainability (SusTech), 2022.