NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

SECTOR OF COMPUTER SCIENCE

Dynamic NFT exchanges in a blockchain environment using graph algorithms

DIPLOMA THESIS

of

Christos Koutras

Supervisor: Vassilios Vescoukis
Professor, NTUA

Athens, April 2025

(this page is left intentionally blank)

NATIONAL TECHNICAL UNIVERSITY OF ATHENS
SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING
SECTOR OF COMPUTER SCIENCE

Dynamic NFT exchanges in a blockchain environment using graph algorithms

DIPLOMA THESIS

OF

Christos Koutras

Supervisor: Vassilios Vescoukis
Professor, NTUA

Approved by the examination committee on 4th of March, 2025.

.....	.....	.....
Vassilios Vescoukis Professor, NTUA	Nikolaos Papaspyrou Professor, NTUA	Aris Pagourtzis Professor, NTUA

Athens, April 2025.

.....

Christos Koutras
Graduate of the School of Electrical and Computer Engineering,
National Technical University of Athens

Περίληψη

Η παρούσα εργασία παρουσιάζει το Barterplace, ένα αποκεντρωμένο σύστημα ανταλλαγής non-fungible tokens (NFTs), το οποίο ενσωματώνει τη θεωρία γράφων και την τεχνολογία blockchain για την υποστήριξη πολυμερών ανταλλαγών χωρίς την ανάγκη ενδιάμεσων. Σε αντίθεση με τις παραδοσιακές αγορές NFT που βασίζονται στην άμεση αγορά και πώληση, το Barterplace αναπαριστά τις προθέσεις ανταλλαγής ως κατευθυνόμενο γράφο και εφαρμόζει αλγορίθμους γράφων για την ανίχνευση κύκλων, διασφαλίζοντας αυτόματη και απρόσκοπτη εκτέλεση συναλλαγών.

Το σύστημα υλοποιείται πάνω στο Ethereum blockchain, αξιοποιώντας τα έξυπνα συμβόλαια για ασφάλεια και διαφάνεια, ενώ το InterPlanetary File System (IPFS) παρέχει ένα αποκεντρωμένο μέσο αποθήκευσης για τα metadata των NFTs. Μέσω των κυκλικών ανταλλαγών, το Barterplace ενισχύει τη ρευστότητα των NFTs, αυξάνοντας τις πιθανότητες επιτυχημένων ανταλλαγών, εξαλείφοντας τους μεσάζοντες, μειώνοντας τα τέλη και διασφαλίζοντας δίκαιες συναλλαγές.

Για τη βελτίωση της αποδοτικότητας και της επεκτασιμότητας, προτείνονται μελλοντικές βελτιστοποιήσεις όπως ο off-chain υπολογισμός των κύκλων και η σχεδίαση συμβολαίων με χαμηλό κόστος gas. Πειραματικές αξιολογήσεις επιβεβαιώνουν τη αξία της προσέγγισης, αναδεικνύοντας τη δυναμική του Barterplace να επαναπροσδιορίσει τις ανταλλαγές ψηφιακών περιουσιακών στοιχείων στο οικοσύστημα του Web3.

Λέξεις-κλειδιά

NFT, blockchain, έξυπνα συμβόλαια, ανταλλαγή NFTs, θεωρία γράφων, Decentralized Exchange (DEX), ανίχνευση κύκλων, Ethereum, IPFS, πολυμερείς ανταλλαγές

(this page is left intentionally blank)

Abstract

This thesis introduces Barterplace, a decentralized non-fungible token (NFT) barter system that integrates graph theory and blockchain technology to enable trustless, multi-party swaps. Unlike traditional NFT marketplaces that focus on direct buying and selling, Barterplace represents trade intents as a directed graph and employs Depth-First Search (DFS) for cycle detection, ensuring seamless and automated trade execution.

The system is implemented on the Ethereum blockchain, leveraging smart contracts for security and transparency, while the InterPlanetary File System (IPFS) provides decentralized storage for NFT metadata. By facilitating swap cycles, Barterplace enhances NFT liquidity, increasing the likelihood of asset exchanges while eliminating intermediaries, reducing fees, and ensuring fair transactions.

To improve efficiency and scalability, future optimizations such as off-chain computation and gas-efficient smart contract design are proposed. Experimental evaluations confirm the feasibility of this approach, demonstrating Barterplace's potential to redefine digital asset trading in the Web3 ecosystem.

Keywords

NFT, blockchain, smart contract, barter, swap, graph theory, Decentralized Exchange (DEX), cycle detection, Ethereum, IPFS, multi-party swaps

(this page is left intentionally blank)

Acknowledgments

I would like to express my sincere gratitude to my supervisor, Vassilios Vescoukis, for his invaluable guidance, patience, and support throughout the course of this thesis.

I am also deeply thankful to Ioannis Tzannetos and Christos Chatzichristofi for their insightful feedback and helpful suggestions, which greatly enhanced the quality of this research.

To my family and friends, thank you for your unwavering love and belief in me. Your support has been a constant source of strength and motivation throughout this journey, even in the face of challenges.

Finally, I wish to thank all those who contributed, directly or indirectly, to the completion of this thesis.

Contents

Chapter 1

Introduction

Background

Blockchain technology has gained significant popularity as a decentralized platform for securely storing and verifying data. One application of blockchain technology that has gained particular attention is non-fungible tokens (NFTs). NFTs are *unique digital assets* recorded on a blockchain and used to certify ownership and authenticity. They have a wide range of use cases, including the ability to represent assets such as art, collectibles, and even virtual real estate. The potential for NFTs to revolutionize how the Internet works has attracted significant interest from industry and academia.

Consequently, many decentralized applications (dApps) across blockchain networks allow users to buy, sell, trade, and auction NFTs. Although there are many classic decentralized NFT marketplaces such as OpenSea and Rarible, as well as some centralized ones like Binance [33], what all these marketplaces share is that they focus on providing essential buying, selling, and bidding/auctioning functionalities.

In this chapter, we will explore the motivations behind this thesis by describing the problem it tries to solve, the proposed solution, and finally, the usefulness of that solution.

Trading in blockchain

A Decentralized Exchange (DEX) is a cryptocurrency exchange that operates on a blockchain instead of relying on a centralized entity. DEXs allow for peer-to-peer trading directly between users without intermediaries. These exchanges utilize *smart contracts* deployed on networks like Ethereum to execute trades based on predefined rules automatically. Some of the most

popular DEXs include Uniswap [46] and SushiSwap [44].

The smart contracts powering DEXs implement an Automated Market Maker (AMM), the underlying protocol connecting users and their crypto assets. Unlike traditional exchanges that use order books and matching buyers with sellers, AMMs operate on the principle of liquidity pools. These pools are filled with users' funds, enabling instant trades without the need for counterparties. AMMs use algorithms to dynamically adjust asset prices based on supply and demand, ensuring continuous liquidity for traders.

Stable assets and StableSwap

Stable assets and, more specifically, *stablecoins*, such as USDC and USDT, are digital currencies with a value strongly tied to fiat currencies like the US dollar or physical assets like gold [30], which results in maintaining a steady value against their reference assets. So when users trade same-valued assets on DEXs, they should have gotten a near-1:1 value of exchanged tokens.

To enhance the efficiency of stablecoin swapping within the Ethereum ecosystem, Michael Egorov introduced a solution outlined in [21]. His proposal revolves around the concept of liquidity pools designed to maintain a consistent relative price for these assets. This protocol is now known as Curve [35], and the liquidity pools using this protocol are called Curve pools.

The proposal

Bartering, or swapping, is a method of trading that has been used since ancient times, but there is currently a lack of focus on this aspect of trading in the NFT market. Some of those existing DApps offer swapping features, but these tend to be limited to trades between two parties and do not consider the broader ecosystem of potential traders.

In exploring the realm of decentralized trading, it becomes essential to address the question: why not extend *stable trades* to NFTs? Traditional methods of NFT trading, such as atomic swaps and auctions, have laid the groundwork for facilitating transactions within the NFT space. The most notable consequence of our proposed framework is creating a ledger-based community comprising equal value mapping — a byproduct emerging from constructing our decentralized trading ecosystem.

This proposal presents an on-chain graph data structure designed to track users' "trade intents", enabling a novel approach to decentralized NFT trading. By organizing trade intents in a graph, valid trades occur when these in-

tents form a *cycle*, ensuring that each participant receives their desired NFT. The proposed protocol supports atomic swaps involving multiple parties, subject to certain operational constraints. Within this graph, the resulting connected components resemble liquidity pools found in traditional DEXs, where assets are held to facilitate new trades. This innovative structure broadens the scope of decentralized trading, providing enhanced flexibility and efficiency in exchanging NFT assets within a decentralized framework.

Usefulness

The usefulness of the Barterplace can be summarized in the following key concepts:

1. *Autonomy*: By eliminating the need for a trusted third party, the on-chain approach enhances both trustworthiness and fairness within the system [19].

2. *Transparency and Verifiability*: Storing trade data on the blockchain allows all participants access to the same information, which increases transparency and ensures data can be independently verified.

3. *Open-Source Architecture*: The open-source nature of smart contracts enables third-party auditing, promoting reliability and security.

4. *Efficiency and Flexibility*: A multiparty trade structure allows multiple trades to be completed simultaneously, improving efficiency. Additionally, as more users participate, the volume of possible trades increases, allowing for faster matching.

5. *Community-Driven Valuation*: The value of each individual NFT is determined solely by the community and its collective trade intents, reinforcing a decentralized, user-focused model.

Structure

This thesis is composed of four chapters. Chapter 2 gives a theoretical background of a blockchain and core concepts about graph theory necessary for our implementation. Chapter 6 provides a high-level overview of the system's functionality, along with four detailed use cases. In Chapter 6, the Barterplace platform is described from a technical perspective, including specific implementation details. Finally, Chapter 6, benchmarks the proposal and evaluates potential directions for future work.

Chapter 2

Theoretical Background

2.1 Blockchain

Blockchain technology, first conceptualized by Stuart Haber and W. Scott Stornetta in 1991 [5], was initially devised to implement a system where document timestamps could not be tampered with. However, it wasn't until 2008 that blockchain gained significant attention when an individual (or group) under the pseudonym Satoshi Nakamoto introduced Bitcoin [12].

Nakamoto's whitepaper, "Bitcoin: A Peer-to-Peer Electronic Cash System", proposed a decentralized digital currency system, utilizing blockchain as its underlying technology. This innovation marked the first practical and widespread application of blockchain, creating a ledger system that is both transparent and secure due to its decentralized nature and cryptographic hashing.

Since then, blockchain has evolved beyond cryptocurrency, finding potential applications in various fields such as finance, supply chain management, healthcare, and voting systems, heralding a new era of secure and transparent digital transactions.

2.1.1 Basic components

2.1.1.1 Block and chain

At the heart of a blockchain is the concept of a "block". Each block contains a collection of data that varies depending on the purpose of the blockchain. For example, in a cryptocurrency blockchain like Bitcoin, a block contains transactional data [12].

Blocks also contain a unique identifier called a hash. A hash is a fixed-size alphanumeric string generated from input data of any size through a

cryptographic hash function, uniquely representing the original data and widely used to ensure data integrity and security.

The "chain" in blockchain comes from how these blocks are linked. When a new block is created, it includes the hash of the most recent block in the chain. This creates a chronological chain of blocks, and this technique ensures the immutability of the blockchain. If any block's data were altered, its hash would change, breaking the chain of hashes and making the tampering evident.

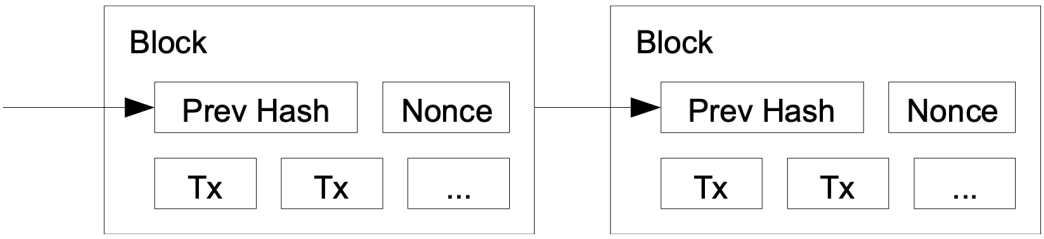


Figure 2.1: Representation of a chain of blocks [12]

2.1.1.2 Nodes and Network

In a blockchain network, "nodes" are the individual computers that collectively maintain and validate the blockchain. Each node has a copy of the entire blockchain ledger, which includes all the blocks of transactions ever executed in the network. These nodes are interconnected, forming a decentralized distributed network, meaning no central authority or single point of failure exists. This architecture guarantees strength and durability, as the network can keep running even if some nodes malfunction or act maliciously.

Nodes have different functions; some validate and transmit transactions, others participate in the consensus process to agree on the state of the blockchain, and specific specialized nodes, usually referred to as *miners* in networks like Bitcoin or *validators* in Ethereum, do the computationally demanding job of adding new blocks to the chain.

This decentralized and distributed structure is vital to the security and integrity of blockchain technology, as it makes the alteration of recorded data extremely difficult and ensures transparency and trust among all participants in the network.

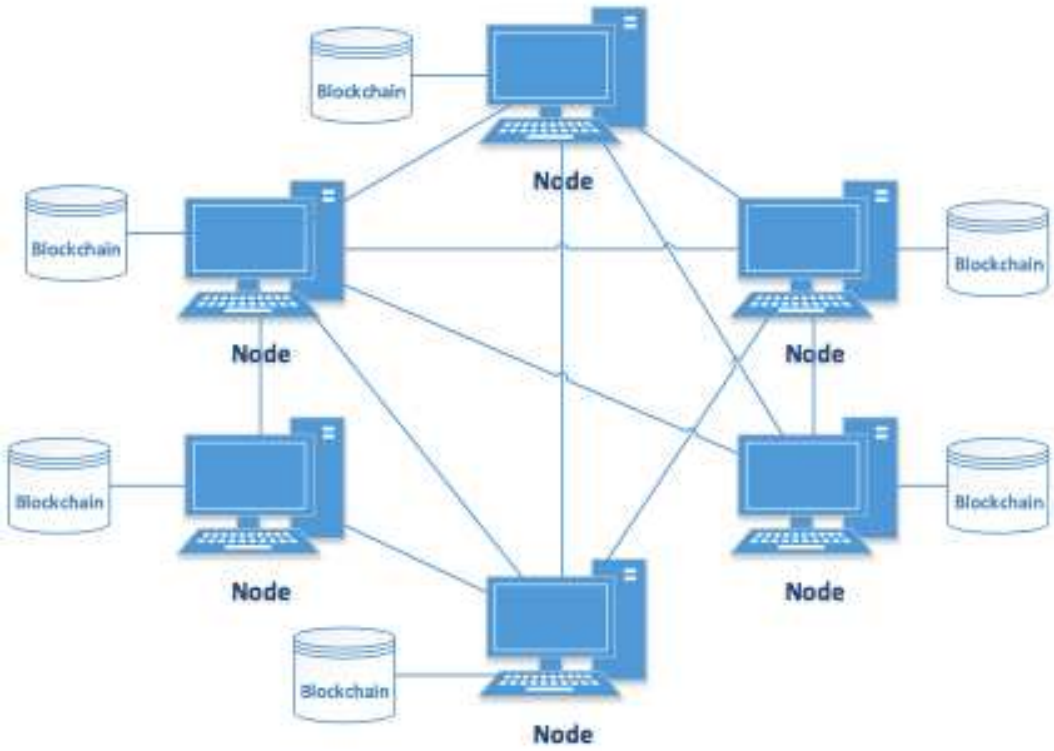


Figure 2.2: A peer-to-peer network of nodes [18]

2.1.1.3 Transactions

Transactions represent the fundamental units of operation within the network. Each transaction typically records the transfer of value or data between parties. For instance, in the context of Bitcoin, a transaction would detail the transfer of coins between user wallets. The sender digitally signs these transactions to ensure authenticity and prevent tampering.

Once a transaction is initiated, it is broadcast to the network and awaits validation. This validation is performed by the nodes of the network. They verify the transaction's legitimacy and compliance with the network's rules, such as checking if the sender has sufficient balance for the transaction. Upon successful validation, transactions are grouped into a block, which is then added to the existing blockchain.

This addition is done through a consensus mechanism, ensuring that all nodes in the network agree on the state of the ledger. Once recorded on the blockchain, a transaction becomes immutable, meaning it cannot be altered or deleted, providing a transparent and tamper-proof record of all network activities.

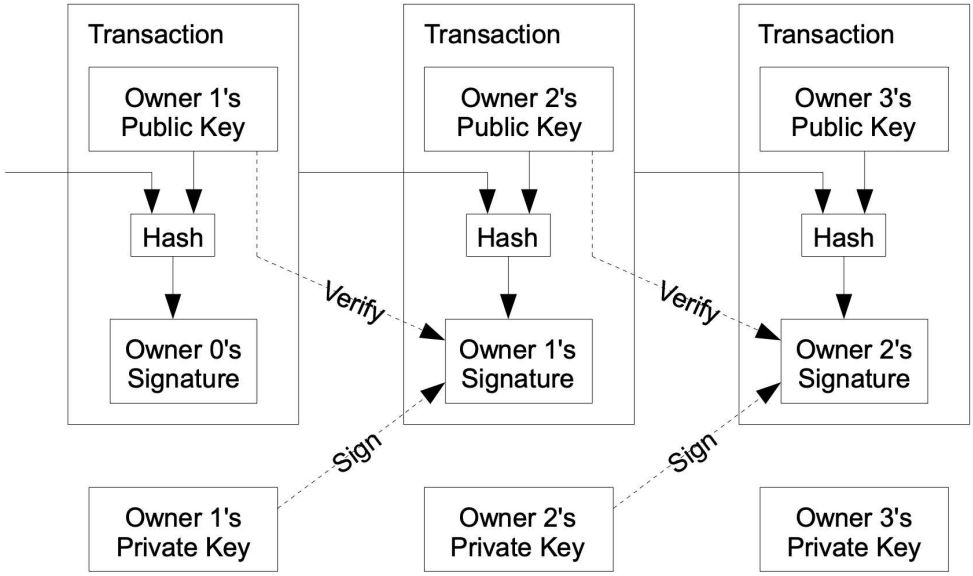


Figure 2.3: Transactions structure [12]

2.1.1.4 Consensus mechanisms

Consensus mechanisms are essential components of blockchain technology, providing a way for network nodes to agree on the legitimacy of transactions and the resulting state of the distributed chain of blocks. These protocols are necessary for network synchronization and for preserving the security and integrity of the blockchain, making sure that all copies of the distributed ledger are consistent.

One of the most well-known consensus mechanisms is Proof of Work (PoW), used by Bitcoin, where miners solve complex cryptographic puzzles to validate transactions and create new blocks. However, PoW is energy-intensive, leading to the development of alternative mechanisms such as Proof of Stake (PoS). In PoS, validators create new blocks based on the amount of cryptocurrency they stake as collateral, making the process more energy-efficient.

Other mechanisms include Proof of Authority (PoA), Delegated Proof of Stake (DPoS), and Byzantine Fault Tolerance (BFT) algorithms, among

many others, each offering different approaches to achieving consensus and ensuring network security. An extensive analysis of the consensus mechanisms can be found in [25].

2.1.2 Smart contracts and NFTs

Smart contracts are self-executing contracts with the terms of the agreement directly written into lines of code [6]. They run on blockchain networks, such as Ethereum [14], and automatically enforce and execute the terms of a contract when predetermined conditions are met. These digital contracts eliminate the need for intermediaries, reducing time and cost while increasing transparency and security.

Smart contracts are versatile and can be used for a wide range of applications, from automating complex financial transactions to managing decentralized applications (dApps). The code and the agreements it contains are spread over a decentralized blockchain network, making them unalterable and irreversible once they have been deployed. This immutability ensures that once a smart contract is executed, the outcome is final, which promotes trust and reliability in digital transactions. Smart contracts represent a significant shift in how agreements are managed and executed in the digital age, offering a more efficient, transparent, and secure way of handling contractual obligations.

Non-Fungible Tokens (NFTs) are digital assets that utilize smart contracts on blockchain networks to establish ownership, authenticity, and transaction mechanisms. Unlike fungible tokens, NFTs represent unique, indivisible assets, such as digital art, virtual real estate, and collectibles, with ownership information stored on the blockchain. The fundamental block of an NFT is a smart contract, which encodes details such as the asset's unique identifier, metadata, and transfer conditions [32].

The Ethereum blockchain pioneered NFT standards, such as ERC-721 and ERC-1155, which define the structure of NFT smart contracts and govern how they interact with decentralized applications (dApps). These standards allow NFTs to be minted, transferred, and traded seamlessly while preserving their authenticity and ownership history [27]. Smart contracts facilitate transactions and implement royalty mechanisms, automatically compensating creators whenever an NFT is resold. This innovation has enabled the development of NFT marketplaces, gaming platforms, and digital identity solutions, expanding the use cases beyond simple asset ownership [31].

2.1.3 Key concepts

In this section, four of the most important characteristics of a blockchain are discussed - decentralization, transparency, immutability, and security - and their significance.

2.1.3.1 Decentralization

Decentralization is a core characteristic that distinguishes blockchain from traditional centralized systems. In a decentralized blockchain network, the ledger is not stored in a single location or controlled by a single entity. Instead, it is distributed across a wide network of nodes, each holding a copy of the entire ledger. This means that no single node or participant has complete control over the entire network, enhancing security and reducing the risk of centralized points of failure or manipulation.

Decentralization encourages transparency and trust, as all transactions are confirmed by consensus among network members and are openly documented on the blockchain, making them easily verifiable by anyone. This structure gives power back to the users and eliminates the need for intermediaries.

2.1.3.2 Transparency and Immutability

Blockchain technology provides a level of transparency that is unprecedented in data management and storage. All transactions are visible and verifiable by any participant in the network, as the data is stored across a distributed network of nodes rather than in a single, centralized database. Each node holds a copy of the entire ledger, making transaction records publicly accessible and consistent throughout the network. This ensures immutability, meaning that data cannot be altered. In the case of blockchain, it means that once a transaction is added and confirmed, it is almost impossible to modify or remove it. This is because of the cryptographic connection of blocks: each block has a unique hash of the preceding block, forming a secure chain of records that is hard to tamper with.

Together, transparency and immutability build a foundation of trust and accountability, as all network participants can verify transactions independently, and the historical record is preserved intact, free from the risk of manipulation or revision. These features make blockchain an ideal platform for applications that require high levels of trust and data integrity, such as financial services, supply chain management, and legal contracts [20].

2.1.3.3 Security

Security assurances in the context of a blockchain are of great importance. Most blockchains possess features that make them particularly robust against fraud, hacking, and unauthorized alterations, distinguishing them from traditional data management systems [23].

The security of a blockchain at the network level primarily derives from its distributed architecture and cryptographic principles. Each transaction recorded on the blockchain is cryptographically signed and linked to previous transactions, forming an immutable ledger. This linkage is maintained through cryptographic hash functions, which generate a unique digital fingerprint (hash) for each block. Any alteration of the data within a block would result in a completely different hash, thereby invalidating all subsequent blocks and making tampering immediately detectable.

Moreover, the decentralized architecture of blockchain ensures that there is no central point of failure. The ledger is replicated across a distributed network of nodes, each maintaining a copy of the entire blockchain. For an attacker to successfully alter or compromise the ledger, they would need to simultaneously modify the majority of these copies or control a substantial number of nodes. This type of attack, known as a Sybil attack [9], involves creating a large number of fake identities to manipulate the network's consensus process. However, in large, well-maintained blockchains, this is computationally infeasible due to the underlying security mechanisms and economic incentives that discourage such behavior.

Most importantly, blockchains operate based on consensus mechanisms, such as Proof of Work (PoW) or Proof of Stake (PoS), which require nodes to reach agreement on transaction validity before appending them to the blockchain [25]. This consensus process reinforces security by ensuring that no single entity can arbitrarily alter the ledger's state, thereby maintaining the system's integrity and trustworthiness.

2.2 Graph theory

Graph theory is a significant area of mathematics and computer science that focuses on the study of graphs, which are mathematical structures used to model pairwise relations between objects [8]. The field was first formalized in the 18th century by the Swiss mathematician Leonhard Euler with his solution to the Königsberg bridge problem, which laid the foundation for the study of networks and connectivity [1].

Since then, graph theory has evolved into a robust tool, essential in var-

ious scientific fields, including biology, computer science, and engineering. Its applications are diverse, including the analysis of social networks, the functioning of search engines, and the study of molecular structures in chemistry. Graph theory's ability to simplify and solve intricate problems through abstract representations makes it a powerful and versatile tool in both theoretical and applied subjects.

In the context of Barterplace, graph theory plays a crucial role in structuring and facilitating multi-party trades. The Barterplace system is modeled as a directed graph where each node represents a unique NFT, and edges signify trade intentions between owners. This structure enables the identification of valid trade cycles using Depth-First Search (DFS), ensuring that swaps occur seamlessly when cycles are detected.

2.2.1 Graphs

A graph is pair of set $G = (V, E)$ that consists of two basic components [7]:

- V represents the set of *nodes* in the graph. Each node $v \in V$ is a distinct entity within the graph.
- E represents the set of *edges* in the graph. In an undirected graph, each edge $e \in E$ is an unordered pair (v_i, v_j), while in a directed graph, it is an ordered pair (v_i, v_j) where $v_i, v_j \in V$

They are broadly categorized into two types: *directed* and *undirected* graphs based on the type of edges in E.

For example, consider a directed graph with three nodes. The graph can be represented as:

$$V = \{v_1, v_2, v_3\}$$
$$E = \{(v_1, v_2), (v_2, v_3)\}$$

This represents a directed graph where there is a directed edge from v_1 to v_2 and from v_2 to v_3.

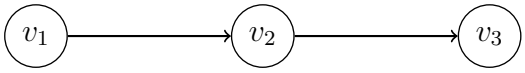


Figure 2.4: Simple graph with three nodes and two edges

2.2.2 Paths and Cycles

A *path* in a graph is a sequence of edges that connects a sequence of *distinct* nodes. For a graph $G = (V, E)$, a path P from node v_1 to node v_n is a sequence of vertices $v_1, v_2, \ldots, v_n$ such that for each i (where $1 \leq i < n$), there is an edge $(v_i, v_{i+1}) \in E$.

For example, consider a path in a graph with vertices v_1, v_2, v_3, and v_4. The path from v_1 to v_4 can be represented as:

$$P = (v_1, v_2, v_3)$$

This implies that there are edges (v_1, v_2) and (v_2, v_3) in the graph.

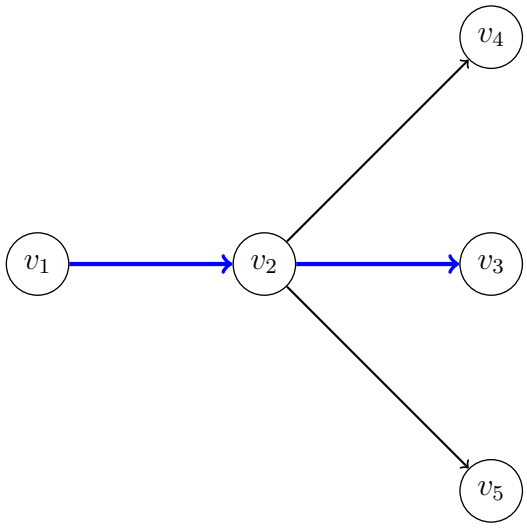


Figure 2.5: Graph with path v_1, v_2, v_3 highlighted

A *cycle* is a path that begins and ends at the same node, forming a closed loop. In a graph $G = (V, E)$, a cycle is a sequence of vertices $v_1, v_2, \ldots, v_k$ with $k \geq 3$ each consecutive pair (v_i, v_{i+1}) for $1 \leq i < k$ is an edge in E. The cycle is closed by the edge (v_k, v_1). Note that in a directed graph, the direction of the edges matters in the definition of a cycle.

For example, a cycle in a graph with vertices v_1, v_2, and v_3 might be represented as:

$$C = (v_1, v_2, v_3, v_1)$$

This implies that there are edges (v_1, v_2), (v_2, v_3), and (v_3, v_1) in the graph, forming a closed loop.

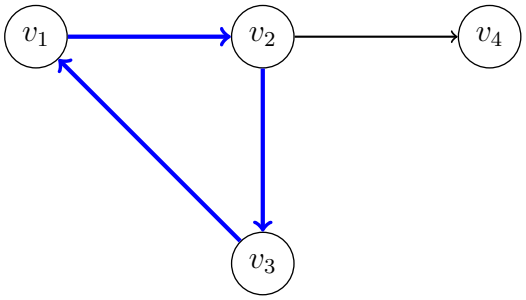


Figure 2.6: Graph with cycle v_1, v_2, v_3, v_1 highlighted

2.2.3 Depth-First Search (DFS)

The Depth-First Search (DFS) algorithm gained prominence with the advent of computer science in the 1950s and 1960s, when it became recognized as an essential method for traversing or searching tree and graph structures. The algorithm was further developed and refined by researchers such as Robert Tarjan [3], who explored its applications in various domains including pathfinding, connectivity checking, and topological sorting. Its importance grew with the rise of algorithmic theory, especially in the study of complexity and optimization, and it operates with a time complexity of $O(V + E)$, where V is the number of NFTs (nodes) and E represents the trade intents (edges) in the graph.

Today, DFS is a fundamental algorithm taught in computer science, providing a foundational approach to problem-solving in areas ranging from artificial intelligence to network analysis [10].

Definition 2.2.1 (Adjacency List)**.** For a directed graph $G = (V, E)$ where V is the set of nodes and E the set of directed edges, an *adjacency list* represents each node $u \in V$ as a list of nodes v such that there exists an edge $e : (u, v) \in E$.

An adjacency list is a data structure where each node is associated with a list of its directly connected neighbors. It is one of the common ways to represent a graph in algorithms and is essential for implementing DFS as shown below. Here's a basic overview of how DFS works:

1. Starting Point: DFS begins at a selected node u.

2. Traversal: The algorithm explores as far as possible along each branch. After visiting a node, it proceeds to an unvisited adjacent node.

3. Backtracking: If a node has no unvisited neighbors, the algorithm backtracks to the nearest ancestor node that has unvisited neighbors and continues the search from there.

4. Marking as Visited: Nodes are marked as *visited* once they are explored to prevent revisiting them.

DFS can be implemented using recursion [15] or iteratively using a stack data structure [4]. The choice between these two approaches usually depends on the specific requirements of the application and the size of the graph. The pseudocode below describes a recursive implementation of Depth-First Search.

Algorithm 1 Recursive Depth-First Search (DFS) algorithm

```
procedure RecursiveDFS(G, v, visited)
    if v is not marked as visited then
        Mark v as visited
        for each neighbor u of v do
            RecursiveDFS(G, u, visited)
        end for
    end if
end procedure
```

Chapter 3

Barterplace conceptualization

3.1 The graph

At the core of Barterplace lies an **graph** data structure designed to capture and represent the dynamic trade intentions of NFT owners. As a directed graph implemented and maintained by a smart contract, Barterplace enables secure, multi-party NFT exchanges that transcend traditional one-to-one trading limitations with enhanced efficiency and cost-effectiveness.

A directed graph is the ideal data structure for multiparty NFT swaps, providing flexibility and enabling efficient trade execution through graph algorithms. One of them used in the concept and implementation of Barterplace is the Depth-First Search (see Section 2.2.3).

In Barterplace's interpretation of the graph:

Nodes as NFTs: Each node encapsulates a unique NFT, comprehensively representing its ownership status and intrinsic metadata.

Edges as trade intents: Each edge symbolizes a unidirectional trade proposal, linking the source NFT (currently owned) to the target NFT (desired by the owner).

Cycles as valid swaps: A complete cycle within the graph represents a topologically coherent trade sequence that can be simultaneously executed.

For analytical clarity, the terms "node" and "NFT", as well as "edge" and "trade", will be used interchangeably throughout this discussion.

3.1.1 How a trade intent is added

The most fundamental graph operation is adding an edge, which signifies the owner's intention to exchange their current NFT for a specific target NFT.

Example 1:
If Alice owns a *duck* NFT and wishes to trade it for Bob's *dog* NFT, the graph can be seen in Figure 3.1.

Figure 3.1: Graph representation of a trade intent between two NFTs

Each node is intrinsically linked with its owner's metadata, ensuring complete traceability and potential reversibility of trade intents.

3.1.2 When a swap occurs

As nodes and edges accumulate in the graph, a cycle inevitably forms, signaling a valid swap for the interconnected NFTs. A modified Depth-First Search (DFS) algorithm detects cycles by tracing a *path* from the **target** node back to the **source** of the most recently inserted trade intent.

Example 2: Building upon the previous scenario, Bob wants to trade his NFT for Alice's. This trade intent completes a cycle in the graph illustrated in Figure 3.2.

Figure 3.2: Graph demonstrating a complete trade cycle

3.1.3 What happens after a trade is matched

Upon identifying a valid swap, a precise mechanism ensures each participant receives their desired asset. The key mechanism involves creating a chain of ownership transfers where each participant receives the NFT they desire. Importantly, these ownership transfers occur exclusively between the NFTs participating in the specific trade cycle, ensuring that no external NFTs are affected by the exchange.

Consider a more complex scenario: Charlie owns a *horse* NFT and seeks to participate in a trade. Bob, initially interested in trading his *dog* NFT with Alice's *duck* NFT, now pivots his intention towards Charlie's NFT, while Charlie wishes to acquire Alice's. The resulting graph is illustrated in Figure 3.3.

Example 3:

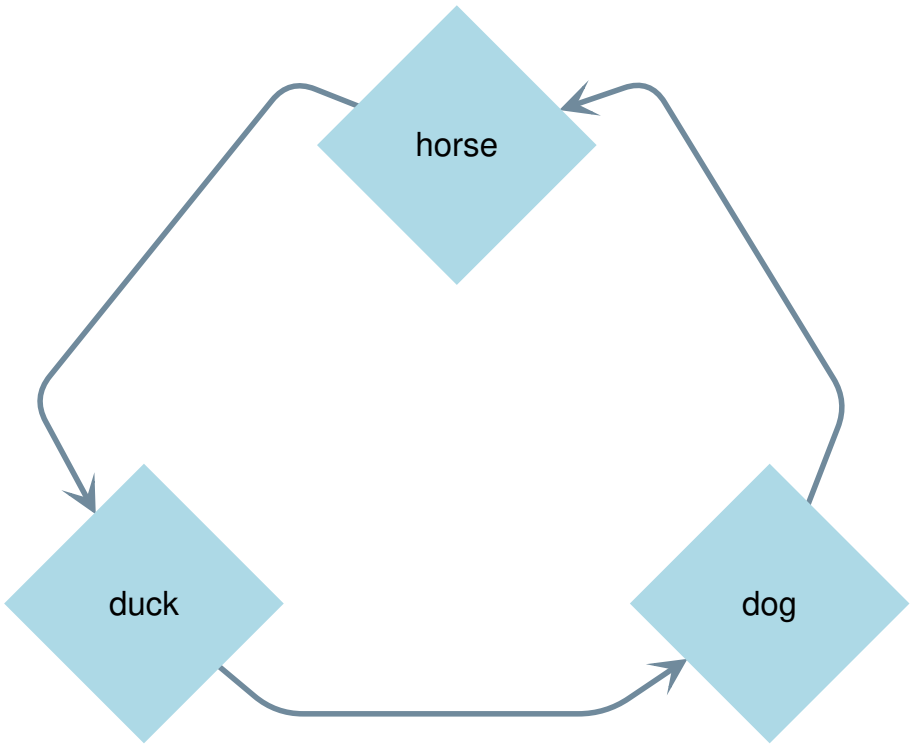


Figure 3.3: Three-party NFT trade cycle representation

By executing this transfer chain, each participant can ultimately acquire their desired asset. This mechanism is highly flexible, supporting complex trade cycles with a theoretically unlimited number of participants.

3.1.4 What happens if multiple trades are valid

When multiple trade cycles emerge upon adding a new edge to the graph, a deterministic conflict resolution policy governs the expected behavior. This policy prioritizes trades based on their chronological insertion, granting precedence to older trade intents.

Example 4:

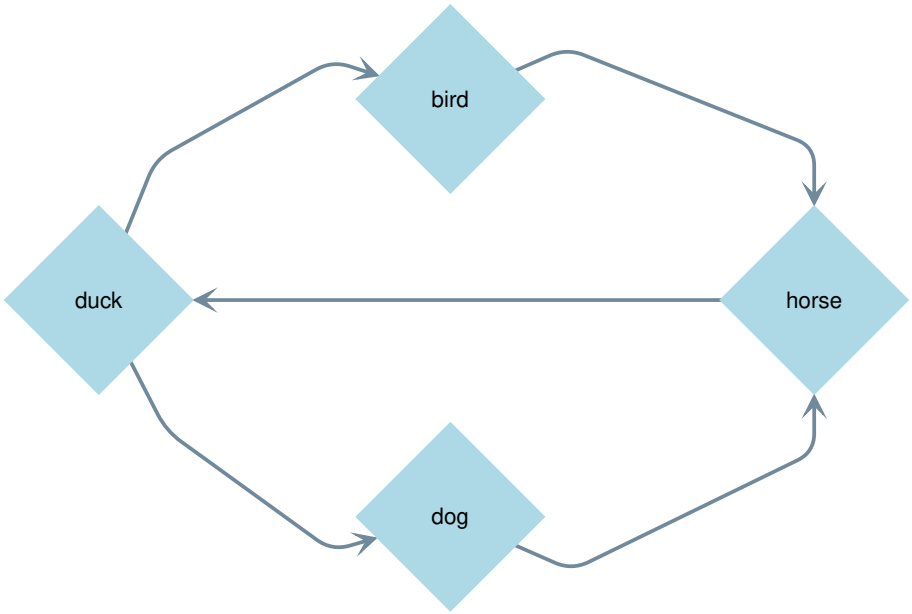


Figure 3.4: Complex graph demonstrating multiple potential trade cycles

In this graph, assume the $(horse, duck)$ edge was the most recently added, simultaneously creating two distinct cycles:

$$C_1 = (duck, bird, horse, duck)$$

$$C_2 = (duck, dog, horse, duck)$$

With each edge addition, the modified Depth-First Search (DFS) algorithm initiates from the edge's target node (here, *duck*), systematically iterates through chronologically sorted trades, and searches for a path to the edge's source node (in this instance, *horse*).

3.2 Functionality and features

3.2.1 Barterplace as an escrow agent

An **escrow agent** is a neutral third party responsible for managing the funds or assets held in an escrow account during the course of a transaction [34]. The escrow agent's role is to ensure that all conditions of a contract are met before the **transfer of assets** between the buyer and the seller occurs. This process helps mitigate the risks for both parties involved by ensuring that neither side can alter the terms of the agreement once the escrow process has begun.

In traditional transactions like real estate deals, an escrow agent keeps the buyer's deposit until certain conditions are met, such as completing a home inspection or securing financing. The agent only releases the money or property after both the buyer and seller fulfill the terms of the escrow agreement. This process builds trust between the parties by making sure the seller gets paid only after meeting their obligations, and the buyer receives the property only after making the payment.

In the context of Barterplace, the escrow agent plays a role similar to that in traditional escrow systems but leverages the power of **smart contracts** to enhance security and transparency. By doing so, the system ensures the enforcement of the terms of the agreement between traders without the need for manual intervention. The smart contract acts as an impartial, self-executing digital agreement, automatically triggering the transfer of the **NFT** only when all predefined conditions are met. Once the conditions are verified, the NFT is securely transferred exclusively to the new owner, ensuring a seamless and trustworthy exchange.

Ultimately, the escrow agent's responsibility is to act impartially and follow the terms outlined in the escrow agreement. Whether in traditional or digital environments, the primary goal of the escrow agent is to protect the interests of all parties while ensuring that the transaction proceeds smoothly and securely.

Barterplace's escrow design introduces several distinctive operational characteristics:

- When initiating an NFT trade, the asset's ownership is temporarily transferred to the Barterplace smart contract, ensuring trade availability while preserving the original ownership structure.
- Users can reclaim an NFT "locked" in the smart contract by removing all outbound trade edges, with the understanding that a locked asset always maintains active outbound trade intentions.

- Upon trade validation, each NFT is systematically transferred to its new designated owner.

3.2.2 Cycle detection

Barterplace is designed to identify potential cycles involving a specific edge. This is achieved by using a Depth-First Search (DFS) algorithm to trace a directed path that connects the target node to the starting node of the given edge. The trade intent matching algorithm relies on this process to form the core mechanism of Barterplace.

A key aspect of the algorithm is the resolution policy, which dictates which edge will be chosen by the DFS at each step of the traversal. This policy ensures that the DFS explores edges in a way that aligns with the participants' trade intents and resolves any potential conflicts or ambiguities in the trade process deterministically. For more detailed explanations of the matching algorithm and the resolution policy, please refer to Section 3.3 and Chapter 6.

3.3 Common use cases

This section demonstrates four of the most common use cases in a system like this. Each example contains a detailed description and a visualization of the state of the graph, along with a step-by-step explanation of the execution of the DFS algorithm that detects the cycle.

3.3.1 Color-coding legend

Nodes

The node color-coding scheme serves as a critical visual representation, enabling better comprehension of ownership patterns and token states within the network.

- All **NFTs** are represented as **diamond-shaped** polygons.
- The **fill color** of each NFT node indicates its **ownership** status, with multiple nodes of the same color representing NFTs owned by a single user.
- A **red outline** signifies that the NFT is temporarily transferred to and **locked** within a smart contract for trading purposes.

Edges

Edge coloration provides crucial information about the network's trading dynamics.

- **Black edges** are trades that did not form a cycle when added.
- **Teal edges** indicate trades that have completed at least one **full cycle** within the network when added.

During the step-by-step demonstration of the Depth-First Search (DFS) algorithm:

- **Black edges** represent the edges that have not been visited yet.
- **Green edges** represent the current traversal path.
- **Red edges** signify those that have been visited but do not contribute to the formation of a cycle.

3.3.2 Simple 2-way swap

Two owners own one NFT each (Figure 3.5).

Figure 3.5: Initial state of the graph

3.3.2.1 Cycle formation

The owner of NFT 1 wants to trade it for NFT 2, so an edge is added to the graph (Figure 3.6).

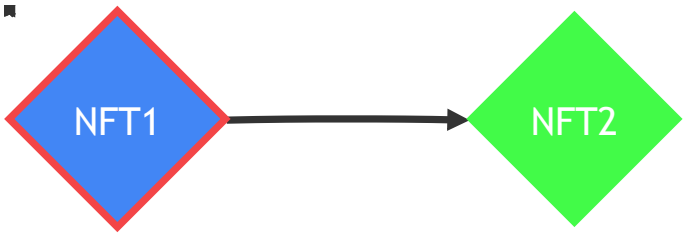


Figure 3.6: Trade added NFT 1 $\bullet\!\!\rightarrow$ NFT 2

The owner of NFT 2 wants to trade it for NFT 1, which, by adding this edge (colored in *teal*), a cycle is created (Figure 3.7).

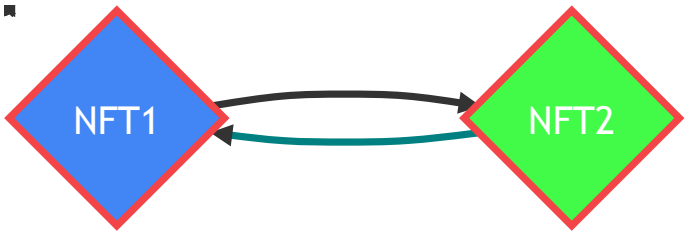


Figure 3.7: Trade added NFT 2 $\bullet\!\!\rightarrow$ NFT 1, and a cycle is formed

3.3.2.2 DFS algorithm

The DFS algorithm initiates its traversal from NFT 1 (the target node of the *teal* colored edge) and attempts to find a path that connects to NFT 2 (the source node of the *teal* edge) (Figure 3.8).

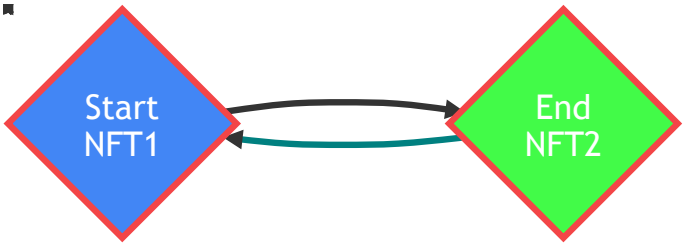


Figure 3.8: Starting DFS from NFT 1

Considering the outgoing edges of the starting node NFT 1, the algorithm selects the only edge from NFT 1 to NFT 2. The existence of a path from NFT 1 to NFT 2 is a *necessary and sufficient* condition for finding a cycle. The algorithm stops and returns the path that it took to reach this NFT (Figure 3.9), in this case:

$$P = (\text{NFT } 1, \text{NFT } 2)$$

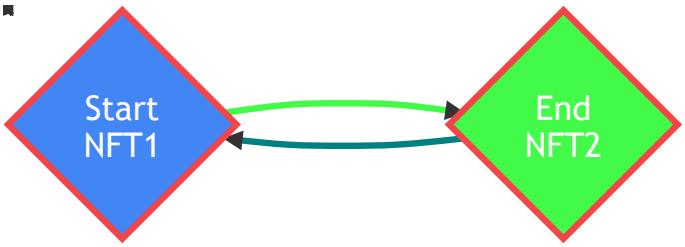


Figure 3.9: Path $NFT1 \rightsquigarrow NFT2$ found successfully

After the path is found, a transfer of ownership is performed so every participant receives their requested asset (Figure 3.10).

1. Owner of NFT 1 receives NFT 2
2. Owner of NFT 2 receives NFT 1

Figure 3.10: Graph after the transfers are completed

3.3.3 Simple 3-way swap

Like the previous example, we have three accounts with one NFT each (Figure 3.11).

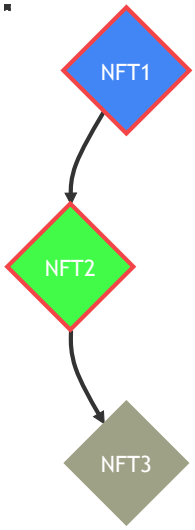
Figure 3.11: Initial state of the graph.

3.3.3.1 Cycle formation

The owner of NFT 1 wants to trade it for NFT 2, so an edge is added to the graph (Figure 3.12a).

The owner of NFT 2 wants to trade it for NFT 3, so an edge is added to the graph (Figure 3.12b).

(a) Trade added NFT 1 $\bullet\!\!\rightarrow$ NFT 2.



(b) Trade added NFT 2 $\bullet\!\!\rightarrow$ NFT 3.

Figure 3.12

The owner of NFT 3 wants to trade it for NFT 1, so an edge is added to the graph (Figure 3.13), and a cycle is formed.

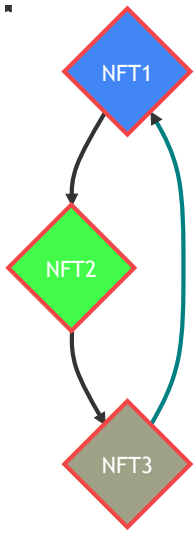
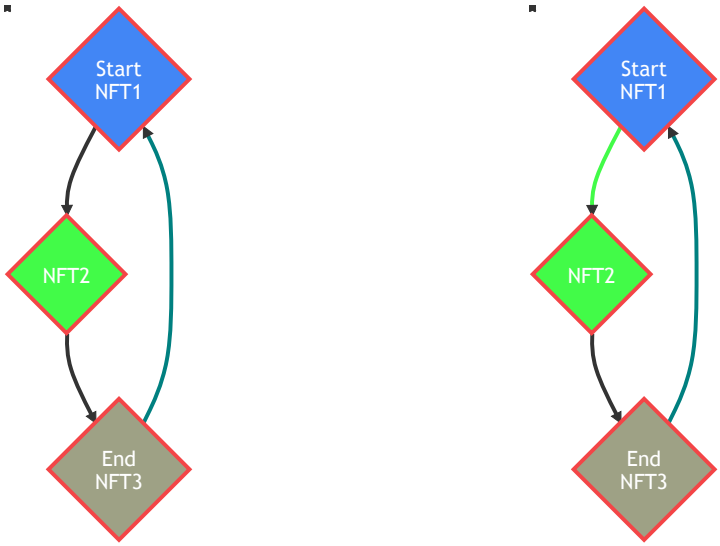


Figure 3.13: Trade added NFT 3 $\bullet\!\!\rightarrow$ NFT 1, and a cycle is formed.

3.3.3.2 DFS algorithm

The DFS algorithm starts at node NFT 1, trying to find a path to NFT 3 (Figure 3.14a). Considering the outgoing edges of the starting node NFT 1, the algorithm selects the only edge from NFT 1 to NFT 2 (Figure 3.14b).

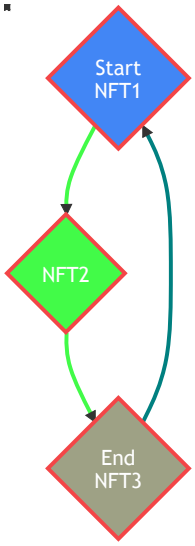


(a) Starting DFS from NFT 1. (b) Follow edge NFT 1 $\bullet\!\!\rightarrow$ NFT 2.

Figure 3.14

We repeat the same process for node NFT 2. Considering the outgoing edges of node NFT 2, there's only an edge from NFT 2 to NFT 3, which also gets selected. The existence of a path from NFT 1 to NFT 3 is a *necessary and sufficient* condition for detecting a cycle. The algorithm stops and returns the path that it took to reach this NFT (Figure 3.15a), in this case:

$$P = (\text{NFT } 1, \text{NFT } 2, \text{NFT } 3)$$



(a) Follow edge NFT 2 $\bullet\!\!\rightarrow$ NFT 3. Path $NFT1 \rightsquigarrow NFT3$ found successfully.

(b) Graph after the transfers are completed.

Figure 3.15

After the path is found, a transfer of ownership is performed so every participant receives their requested asset (Figure 3.15b).

3.3.4 Complex 3-way swap

This case consists of 7 NFTs and 3 owners.

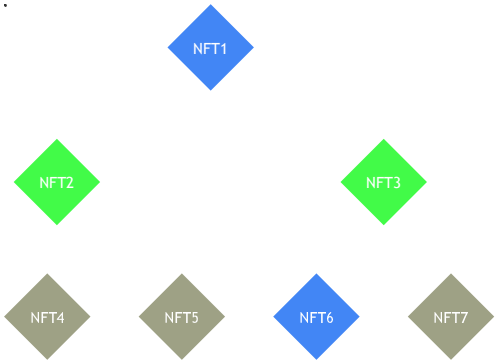
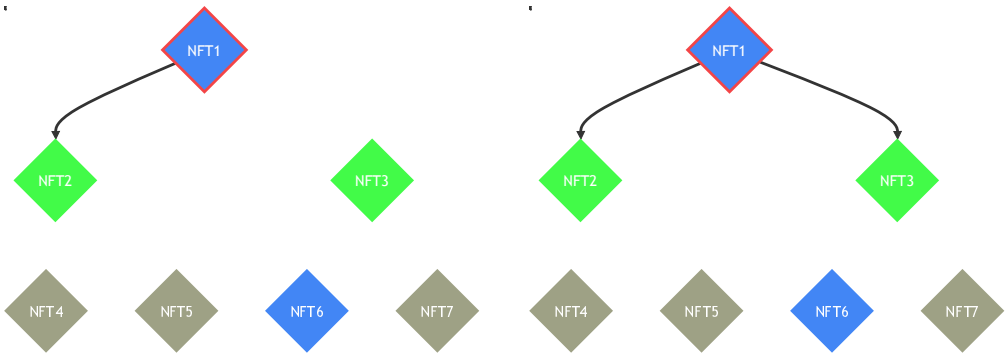


Figure 3.16: Initial state of the graph.

3.3.4.1 Cycle formation

The owner of NFT 1 wants to trade it for NFT 2, so an edge is added to the graph (Figure 3.17a).

The owner of NFT 1 wants to trade it for NFT 3, so an edge is added to the graph (Figure 3.17b).

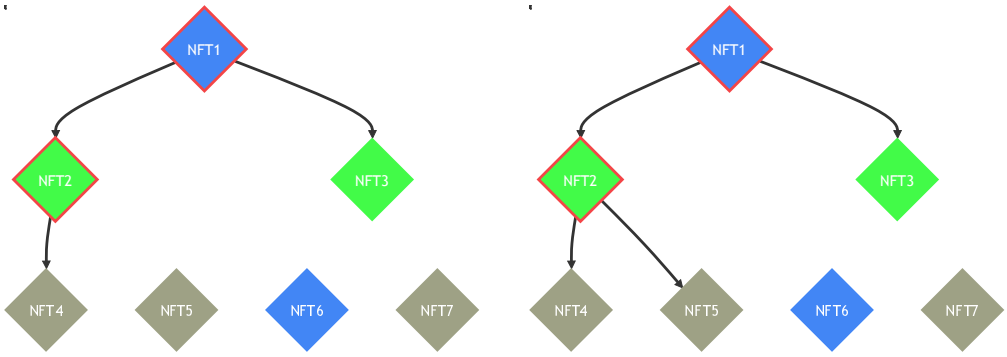


(a) Trade added NFT 1 ⟷ NFT 2. (b) Trade added NFT 1 ⟷ NFT 3.

Figure 3.17

The owner of NFT 2 wants to trade it for NFT 4, so an edge is added to the graph (Figure 3.18a).

The owner of NFT 2 wants to trade it for NFT 5, so an edge is added to the graph (Figure 3.18b).

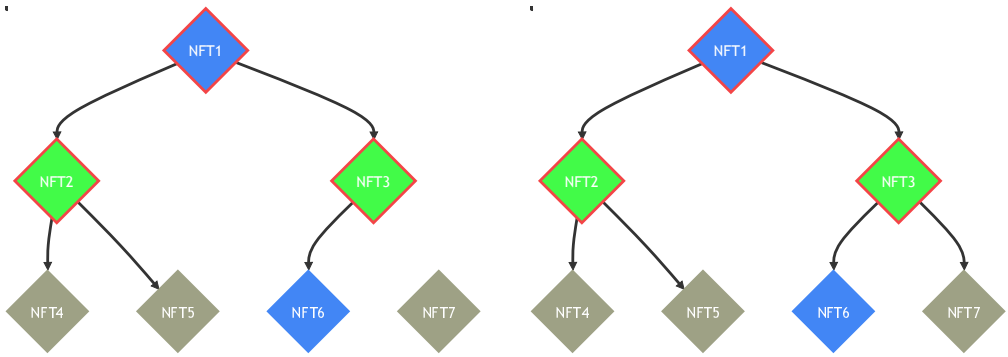


(a) Trade added NFT 2 ⟷ NFT 4. (b) Trade added NFT 2 ⟷ NFT 5.

Figure 3.18

The owner of NFT 3 wants to trade it for NFT 6, so an edge is added to the graph (Figure 3.19a).

The owner of NFT 3 wants to trade it for NFT 7, so an edge is added to the graph (Figure 3.19b).



(a) Trade added NFT 3 ⊷ NFT 6. (b) Trade added NFT 3 ⊷ NFT 7.

Figure 3.19

The owner of NFT 5 wants to trade it for NFT 1, so an edge is added to the graph (Figure 3.20), and a cycle is formed.

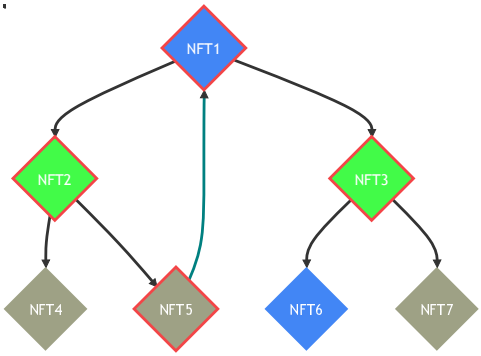
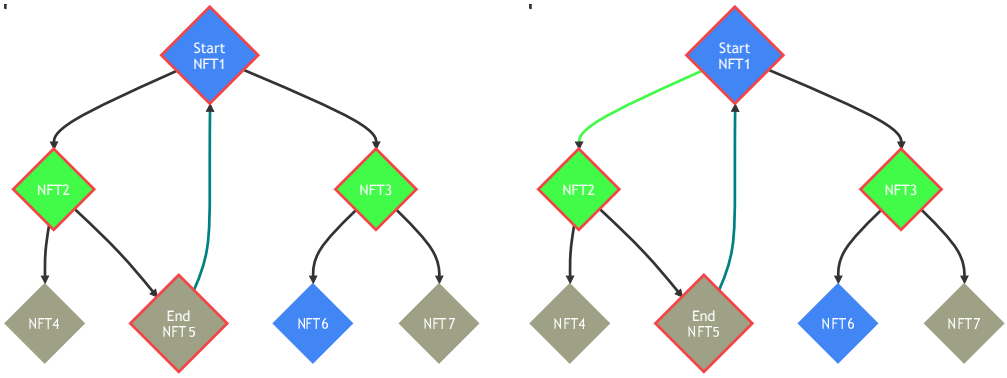


Figure 3.20: Trade added NFT 5 ⊷ NFT 1 and a cycle is formed.

3.3.4.2 DFS algorithm

The DFS algorithm starts from node NFT 1 (Figure 3.21a), considering the outgoing edges of the starting node NFT 1. Between the edges (NFT 1, NFT 2) and (NFT 1, NFT 3), the first is selected as it is the first added in chronological order (Figure 3.21b).



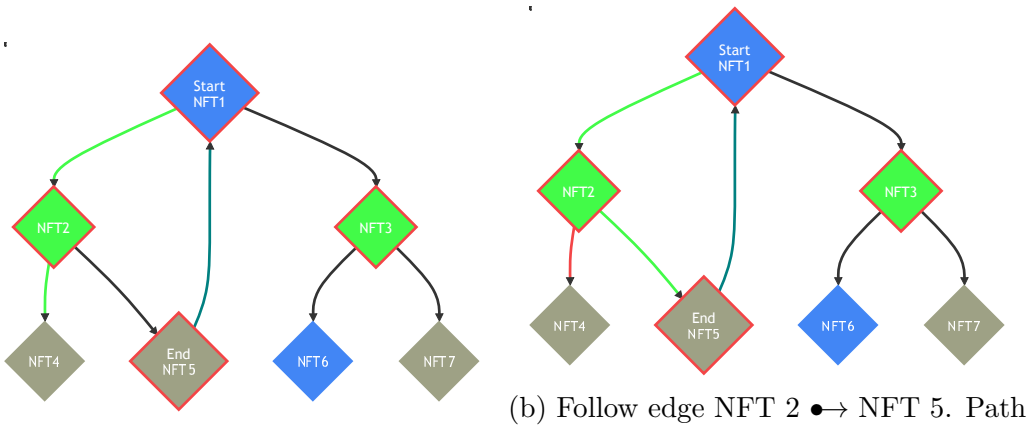
(a) Starting DFS from NFT 1. (b) Follow edge NFT 1 $\bullet\!\!\rightarrow$ NFT 2.

Figure 3.21

We repeat the same process for node NFT 2. Considering the outgoing edges of node NFT 2, (NFT 2, NFT 4) and (NFT 2, NFT 5), the first one is selected as it was added before edge (NFT 2, NFT 5).

At this point, by repeating the process for node NFT 4 it is evident that there are no other edges starting from this node so we continue by following the next of this nodes parent. This edge is (NFT 2, NFT 5) and it denotes the existence of a path from NFT 1 to NFT 5, which is a necessary and sufficient condition for detecting a cycle. The algorithm stops and returns the path that it took to reach this NFT (Figure 3.22b), in this case:

$$P = (\text{NFT } 1, \text{NFT } 2, \text{NFT } 5)$$



(a) Follow edge NFT 2 $\bullet\!\!\rightarrow$ NFT 4. (b) Follow edge NFT 2 $\bullet\!\!\rightarrow$ NFT 5. Path $NFT1 \rightsquigarrow NFT5$ found successfully.

Figure 3.22

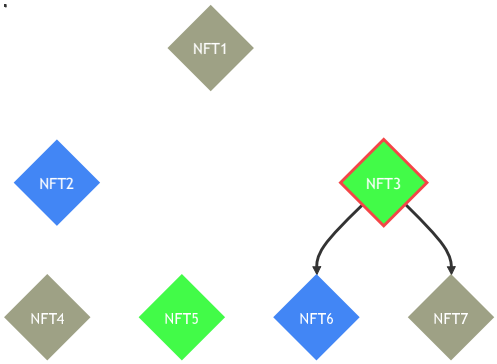


Figure 3.23: Graph after the transfers are completed.

After the path is found, a transfer of ownership is performed so every participant receives their requested asset (Figure 3.23).

3.3.5 Simple 3-way swap with multiple cycles

This case consists of 4 NFTs and 3 owners.

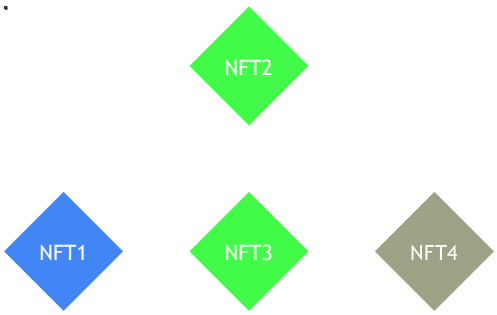
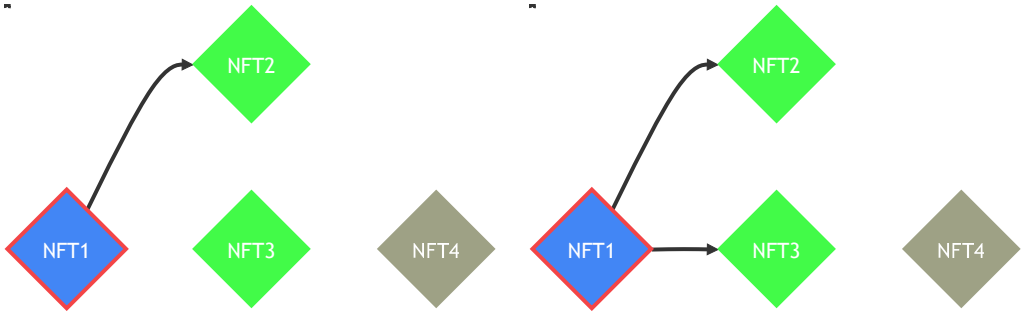


Figure 3.24: Initial state of the graph.

3.3.5.1 Cycle formation

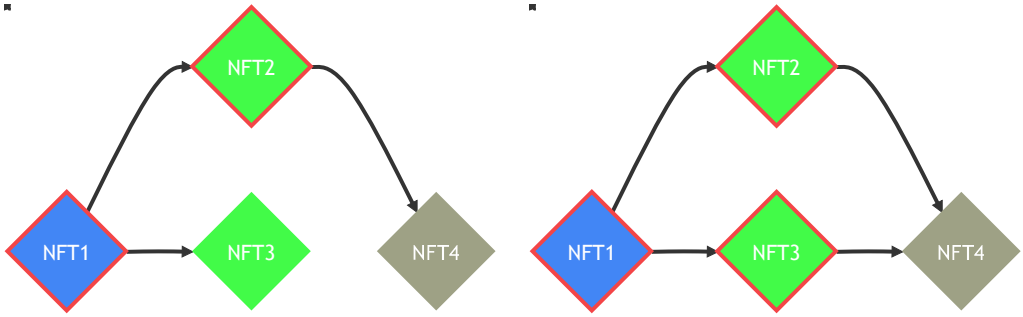
The owner of NFT 1 wants to trade it for NFT 2, so an edge is added to the graph 3.25a).



(a) Trade added NFT 1 ⊷ NFT 2. (b) Trade added NFT 1 ⊷ NFT 3.

Figure 3.25

The owner of NFT 1 wants to trade it for NFT 3, so an edge is added to the graph 3.25b). The owner of NFT 2 wants to trade it for NFT 4, so an edge is added to the graph 3.26a).



(a) Trade added NFT 2 ⊷ NFT 4. (b) Trade added NFT 3 ⊷ NFT 4.

Figure 3.26

The owner of NFT 3 wants to trade it for NFT 4, so an edge is added to the graph 3.26b). The owner of NFT 1 wants to trade it for NFT 2, so an edge is added to the graph 3.27), and a cycle is formed.

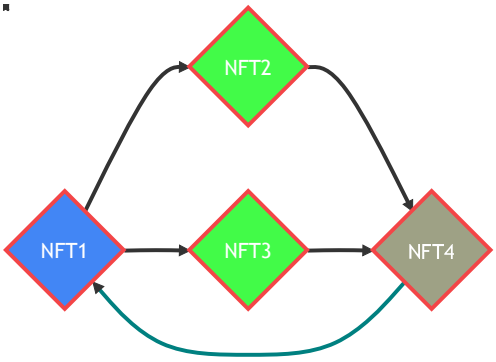
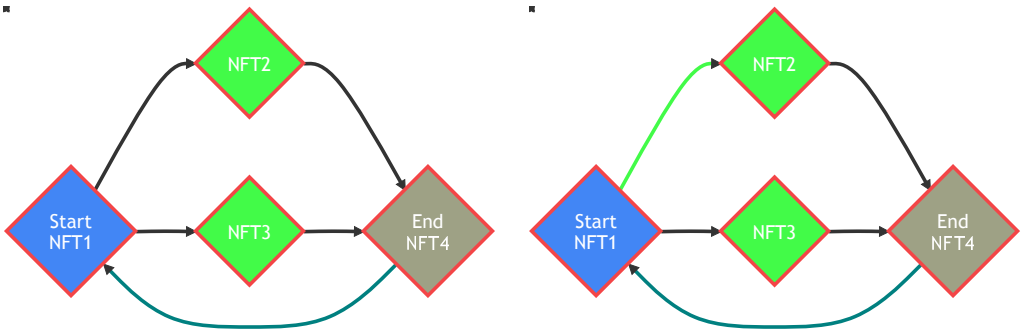


Figure 3.27: Trade added NFT 4 $\bullet\!\!\rightarrow$ NFT 1 and a cycle is formed.

3.3.5.2 DFS algorithm

The DFS algorithm starts from node NFT 1 (Figure 3.5α′), considering the outgoing edges of the starting node NFT 1. Between the edges (NFT 1, NFT 2) and (NFT 1, NFT 3), the first is selected as it is the first added in chronological order (Figure 3.5β′).

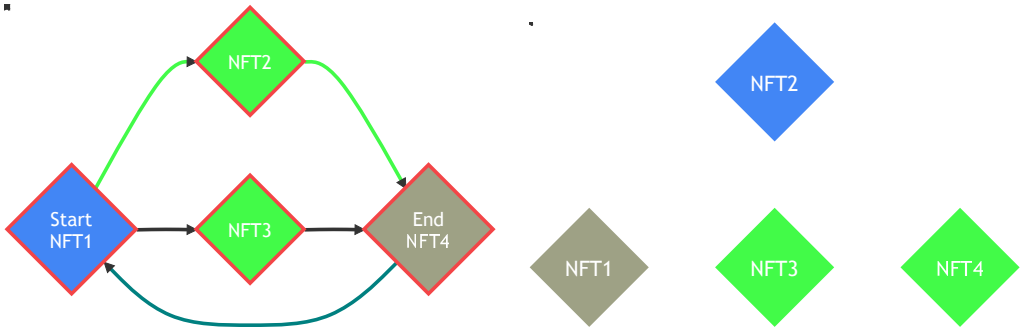


(a) Starting DFS from NFT 1. (b) Follow edge NFT 1 $\bullet\!\!\rightarrow$ NFT 2.

Figure 3.28

We repeat the same process for node NFT 2. Considering the outgoing edges of node NFT 2, there's only an edge from NFT 2 to NFT 4, which gets selected. The existence of a path from NFT 1 to NFT 4 is a *necessary and sufficient* condition for detecting a cycle. The algorithm stops and returns the path that it took to reach this NFT (Figure 3.6α′), in this case:

$$P = (\text{NFT } 1, \text{NFT } 2, \text{NFT } 4)$$



(a) Follow edge NFT 2 $\bullet\!\!\rightarrow$ NFT 4. Path $NFT1 \rightsquigarrow NFT4$ found successfully.

(b) Graph after the transfers are completed.

Figure 3.29

After the path is found, a transfer of ownership is performed so every participant receives their requested asset (Figure 3.6β′).

Chapter 4

Project implementation

4.1 Ethereum and Solidity

Expanding from Section 2.1, where the basic concepts of a blockchain were introduced, in this section, the scope is narrowed down to the Ethereum blockchain, its inner workings, and how these are utilized in the project.

4.1.1 Ethereum Virtual Machine (EVM)

The Ethereum Virtual Machine (EVM), as also mentioned in the Ethereum whitepaper [14], is the core component that powers Ethereum. It is a decentralized computation engine that allows developers to build and deploy smart contracts and decentralized applications (dApps). The EVM acts as the runtime environment for smart contracts, enabling them to be executed in a secure and consistent manner across all Ethereum nodes. By providing a Turing-complete [2] environment, the EVM can execute any computable logic with some limitations.

4.1.1.1 Architecture of the EVM

The EVM operates as a stack-based architecture, where computations are performed using a last-in, first-out (LIFO) data structure. Each smart contract is written in a high-level programming language, such as Solidity which is specifically designed for EVM, and compiled into bytecode before deployment. This bytecode is a low-level, machine-readable format that the EVM executes. To maintain deterministic outcomes, every node in the Ethereum network executes the same bytecode. This ensures consensus across the decentralized network, as all nodes validate and agree on the state changes resulting from contract execution [16].

4.1.1.2 Gas and Resource Management

One of the most critical aspects of the EVM is its gas mechanism, which regulates computational resource usage. Every operation in the EVM has an associated gas cost, reflecting the computational complexity and resources required for execution. For instance, simple arithmetic operations consume less gas compared to more complex storage or cryptographic functions. Users initiating transactions must specify a gas limit and gas price, which determine the maximum amount of gas they are willing to spend and the fee they are prepared to pay per unit of gas, respectively.

The gas system serves multiple purposes: it incentivizes miners to process transactions, prevents malicious actors from overloading the network with resource-intensive operations, and ensures fair resource allocation. Any remaining gas that was allocated but unused is returned to the sender, but the portion of gas already consumed is permanently lost to the network as compensation to the miners or validators for their computational effort [14].

4.1.1.3 Smart contract deployment and execution

Smart contracts deployed on Ethereum are essentially autonomous programs that interact with the state of the blockchain. These programs define the rules and functions of the contract, including how they operate and under what conditions they execute specific actions. Once a contract is deployed, nobody can alter it anymore; its code and terms are permanent on the Ethereum blockchain, providing transparency and security. This is done by sending a transaction to the network that contains the contract's code. When the transcation is verified, the contract is assigned a unique address on the Ethereum blockchain [14].

A very accurate representation of the deployment process of an Ethereum smart contract is defined by [26] which is described below and is shown in Figure 4.1.

1. Definition of .sol file, which contains the Solidity code of a smart contract.

2. Compilation of smart contract's code produces the bytecode deployed on EVM and the Application Binary Interface (ABI).

3. Deployment of the bytecode on the Ethereum blockchain that creates an instance of the smart contract at a specific address.

4. A provider hosted by a network node exposes multiple APIs like HTTP or JSON gRPC that allow calling the blockchain's methods.

5. Interactions with the smart contract are typically performed through a library like Web3.js [47] or Ethers.js [36] that connects to the provider using the ABI.

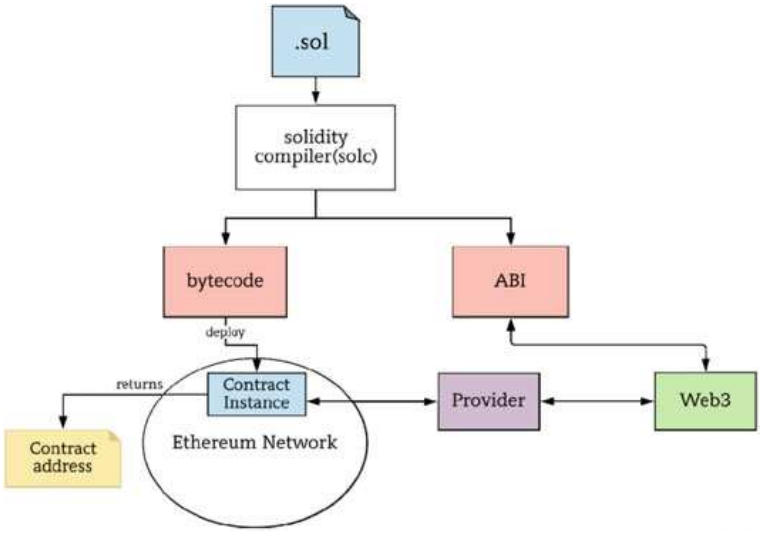


Figure 4.1: Deployment process as demonstrated in [26]

When a contract is invoked, the EVM creates an isolated environment known as the "execution context". This context provides access to essential resources, such as the blockchain state, input data, and the address of the contract being executed.

To manage data efficiently during execution, the EVM uses three types of storage:

1. **Storage**: Persistent storage tied to a specific contract, which is expensive to read and write.

2. **Memory**: Temporary storage that exists during the execution of a contract and is cleared afterward.

3. **Stack**: The primary area for computation, where intermediate values are stored during execution.

The EVM's deterministic behavior ensures that the execution of a smart contract produces identical results on every node, guaranteeing network-wide consensus. This automation and decentralization are key features that reduce the need for trust and the possibility of manipulation.

4.1.2 Events

Ethereum smart contract events are a crucial feature for tracking changes and interactions within a contract. The process begins with defining the event within the smart contract's code written in Solidity. Once the event is declared, it can be emitted within the contract's functions. When a function emits an event is executed, the event data are recorded on the Ethereum blockchain but not in the primary blockchain state. This makes it a cost-effective way of storing transaction execution details [16].

These events are not accessible from within the contracts but can be monitored and read externally. When an event is emitted, it generates a log that includes the event's *data*, the *address* of the contract that emitted it, and the transaction's *hash*. This log is then indexed by the Ethereum network, allowing applications to query and retrieve these logs. This feature is particularly useful for creating user interfaces that react to contract state changes and for off-chain applications that need to track transactions and interactions with a smart contract.

Barterplace smart contract defines two events that help off-chain components, namely NFT discovery, which is explained in greater detail in Section 4.4.5.

1. Emitted when an edge is *added* to the graph state stored on-chain.

```
event TradeAdded(
    address indexed owner,
    address indexed ownerNftContract,
    uint256 ownerTokenId,
    address indexed targetNftContract,
    uint256 targetTokenId
);
```

2. Emitted when an edge is *removed* from the graph state stored on-chain.

```
event TradeRemoved(
    address indexed owner,
    address indexed ownerNftContract,
    uint256 ownerTokenId,
    address indexed targetNftContract,
    uint256 targetTokenId
);
```

4.1.3 Libraries

On-chain Ethereum libraries are smart contracts that act as reusable code modules, enhancing smart contract development by promoting code reusability and optimizing gas usage. Smart contracts can link to these libraries, referencing commonly used code deployed on the blockchain. This linkage reduces the contract's size and deployment costs while ensuring code consistency and security through shared, auditable library code.

An honorable mention to the OpenZepellin framework, which provides an open-source collection of secure, standardized, and audited smart contract components [42]. These contracts are considered standard best practices designed to address common challenges, such as security vulnerabilities and implementation errors, significantly reducing the risk of bugs and exploits. For these reasons, it is widely used by most Ethereum projects, including Barterplace.

For Barterplace's implementation needs, three libraries were created, which are the main contract links, mainly for honoring the Single Responsibility Principle (SRP) purposes and possibly deploying some reusable code for other developers in the future. Great inspiration was drawn from Rob Hitchens' Unordered Key Set [22] for implementing these libraries.

1. *DirectedGraph*: Implements a directed graph data structure used by the main contract to store the trades and NFTs users add. Supports all the basic functionality needed from a graph like insert/remove node, insert/remove edge, etc. which can also be seen in the class diagram shown in Figure 4.2.

2. *LinkedList*: Implements a simple linked list data structure. It is required to store the edges, inbound and outbound, for each graph node. The linked list is chosen because keeping the order in which the edges are added to the graph is essential.

3. *UnorderedKeySet*: Implements an unordered key set data structure that stores unique key values. They are used mainly for validation checks that prevent referencing empty memory addresses when accessing Solidity mappings.

UML class diagram

Below is the class diagram representing DirectedGraph, LinkedList, and UnorderedKeySet libraries.

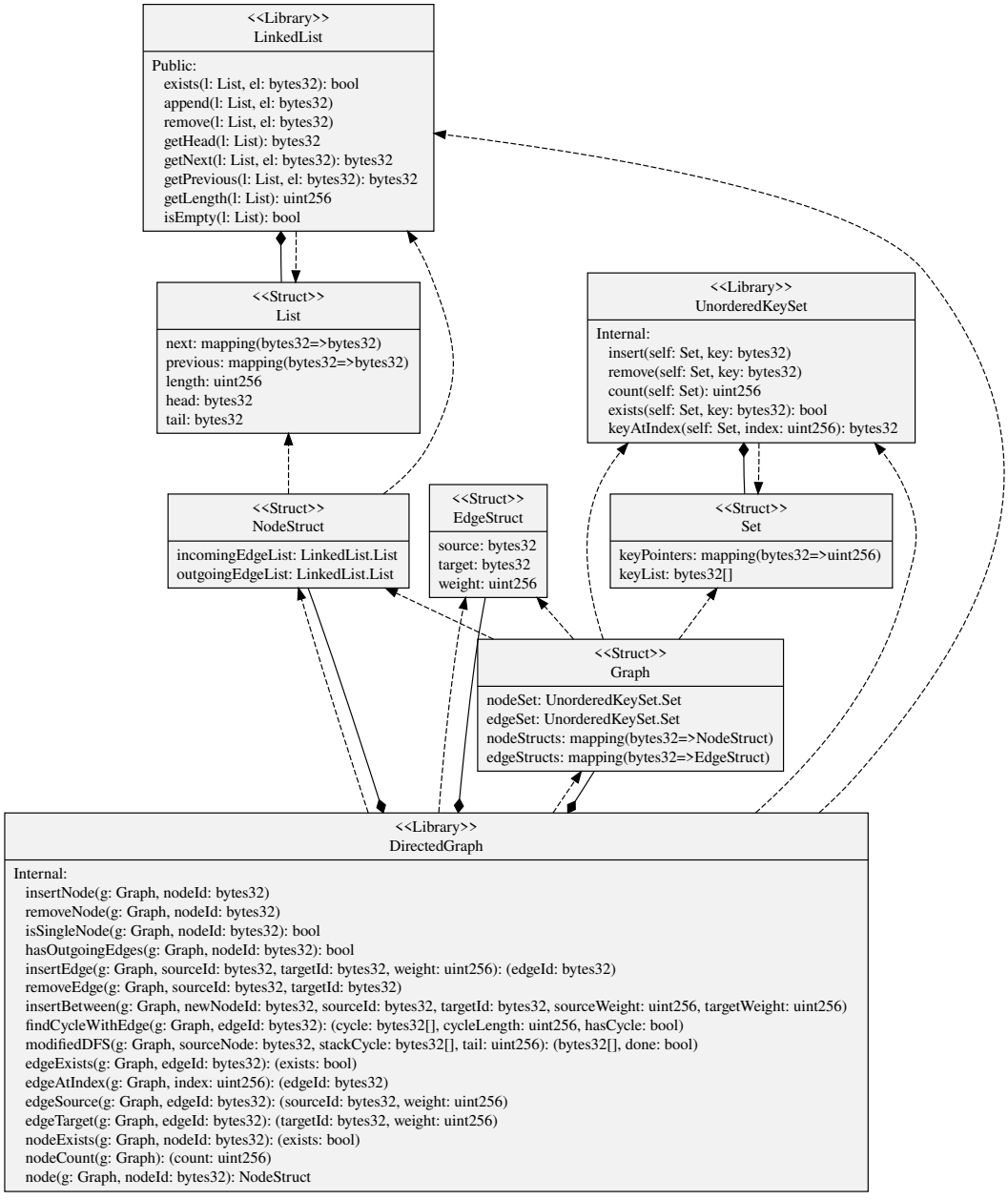


Figure 4.2: Class diagram of DirectedGraph, LinkedList and UnorderedKeySet

4.1.4 Setting up a private Ethereum network

Setting up a private network was performed to familiarize myself with the client's instructions and to understand the process of setting up an Ethereum network from scratch. The following steps describe the above process:

1. **Install geth:**

   Firstly, geth is installed, the Go implementation of an Ethereum node [37].

2. **Create the genesis file:**

   The *genesis.json* file is a JSON configuration file that defines the characteristics of the blockchain. Clique PoA is selected as the protocol, which allows for faster block times and doesn't require mining, making it ideal for testing purposes.

3. **Initialize the Blockchain:**

   Geth is used to initialize the node of the local network with the genesis file. The command is:

   ```
   geth init /path/to/genesis.json --datadir /path/to/node
   ```

4. **Create an account:**

   At least one account needs to be created that is used as the signer. The command is:

   ```
   geth --datadir /path/to/node account new
   ```

5. **Start the node:**

   Start the node with geth. The network ID, the data directory, and enable the HTTP need to be specified. The command looks like this:

   ```
   geth --datadir /path/to/node --networkid 1234 --http --http.port 8545
   --http.addr 127.0.0.1 --http.api eth,net,web3,txpool,miner --mine
   ```

6. **Interact with the blockchain:**

   Now that the local Ethereum network is up and running, smart contracts can be deployed and transactions can be made similar to the main Ethereum network.

4.1.5 Developer experience

While working on this thesis, I explored various tools developed within the Ethereum ecosystem that help make the lives of developers easier. More specifically, these tools help streamline parts of some basic workflows, such as coding, testing, deployment, and documentation, which otherwise would be busy work. The most noteworthy are the following.

4.1.5.1 Remix IDE

Remix IDE is a powerful open-source web and desktop application that offers an accessible, user-friendly interface that significantly eases the process of writing, testing, and deploying smart contracts [43]. It is an excellent platform for both novice and experienced developers, which helps speed up development processes, encourages learning, and enables the production of reliable smart contracts in the Ethereum environment.

It was mainly used in the early stages of development of the project while still learning the Solidity language.

4.1.5.2 Hardhat

Hardhat is an advanced development environment, testing framework, and asset pipeline, all crucial for efficient smart contract development [38]. Hardhat simplifies compiling, deploying, testing, and debugging Ethereum smart contracts. Its local Ethereum network deployment feature is particularly beneficial, allowing developers to deploy and test their contracts in a controlled environment before launching them on the live blockchain. This saves time and resources and significantly reduces the risk of costly errors.

Hardhat was especially helpful during the later stages of the project when unit testing and deployment were necessary. Furthermore, this tool resolved one of the significant issues, creating a command-line interface (CLI) as shown in Chapter 4.3.6, by taking advantage of its tasks feature.

4.1.5.3 Ethers.js

Ethers.js is a prominent library in the Ethereum development environment, renowned for its lightweight yet powerful features tailored for interacting with the Ethereum blockchain and its smart contracts [36]. The library provides comprehensive functionality, including wallet creation and management, connection to Ethereum nodes, and crafting and signing transactions.

Hardhat extensively integrates Ethers.js at its core, augmenting it with additional functionalities. Beyond the combined use of Ethers.js and Hard-

hat, the NFT discovery service (see Section 4.4.5) leveraged this library to subscribe to published blocks, as well as to fetch and parse events emitted by the Barterplace smart contract.

4.1.5.4 Docker

Docker is a widely used containerization platform that simplifies the development, deployment, and execution of applications by packaging them and their dependencies into isolated containers. In the context of Ethereum development, Docker provides a consistent environment across different systems, ensuring that applications work seamlessly regardless of the host machine's configuration.

For the Barterplace project, Docker played a crucial role in deploying and managing certain services efficiently. It was particularly beneficial for setting up a reproducible development environment, eliminating dependency conflicts, and streamlining the deployment of key services such as the NFT discovery service (see Section 4.4.5), IPFS service (see Section 4.2) and off-chain database (see Section 4.4).

4.2 IPFS

4.2.1 What is IPFS

IPFS, which stands for InterPlanetary File System, is a revolutionary protocol and network designed by Juan Benet to create a peer-to-peer method of storing and sharing hypermedia in a distributed file system [13].

IPFS is especially relevant in the realm of blockchain technology. Unlike traditional file storage systems that depend on centralized servers, IPFS stores files across a network of nodes, eliminating any single point of failure and enhancing censorship resistance. Every file and piece of content on IPFS is uniquely identified by a cryptographic hash of its content, making it *content addressable*, which guarantees integrity. As stated in the paper, "Objects are permanent". This decentralized approach increases file redundancy and availability and reduces the need for centralized cloud storage providers, potentially decreasing costs.

It has been widely used in various areas, including the blockchain space, for storing large data objects off-chain, such as images and videos, particularly in NFT projects. Storing large NFT assets directly on-chain is impractical due to high gas fees and storage limitations on Ethereum. IPFS provides a robust way to store NFT metadata and digital assets without overloading the blockchain.

4.2.2 Deployment

A local IPFS network deployment was essential for this thesis, as most NFTs use it for metadata storage. *Kubo* implementation[39] of the IPFS protocol is chosen because it is the first created and the most popular.

1. **Download:** Download the latest version of Kubo and install it.

2. **Init:** Initialize the IPFS server in a directory where all content and configurations are stored, typically *$HOME/.ipfs/*.

   ```
   ipfs init --profile=server
   ```

3. **Configure:** Set up the server's configuration. When the daemon runs, these settings expose the IPFS gateway and the WebUI to the localhost. *LIBP2P_FORCE_PNET* environment variable raises an error if the configuration is not correctly set for private network support.

```
ipfs config Addresses.API /ip4/127.0.0.1/tcp/5001
ipfs config Addresses.Gateway /ip4/127.0.0.1/tcp/8080
export LIBP2P_FORCE_PNET=1
```

4. **Bootstrap nodes:** Since this is the first node of the private network, no bootstrap nodes are needed.

   ```
   ipfs bootstrap rm --all
   ```

5. **Swarm key:** Generate a swarm key by installing *ipfs-swarm-key-gen* and running the following command. This key must be shared if any other node needs to be added to the network.

   ```
   ipfs-swarm-key-gen > $HOME/.ipfs/swarm.key
   ```

6. **Start daemon:** Start hosting the server.

   ```
   ipfs daemon
   ```

WebUI or the Kubo command line interface (CLI) can be used to add files to the network. These files stored on the network can be addressed with their content identifier (CID) and retrieved from *localhost:8080/ipfs/:cid*.

WebUI is an administrative graphical interface of the node, hosted on *localhost:5001/webui*. The code of this UI is distributed by the IPFS protocol, similar to all other files stored. Before proceeding with the steps above, the daemon is started without removing the default bootstrap nodes and navigating to WebUI's address so that the code for the interface is fetched from the public network.

The gateway addresses must be set to *0.0.0.0* to expose the gateway port to the public.

```
ipfs config Addresses.Gateway /ip4/0.0.0.0/tcp/8080
```

4.3 Barterplace smart contract

The Barterplace smart contract enables secure and automated NFT trading by handling trade requests, adding and removing trades, verifying trade cycles and facilitating the ownership transfers. It ensures that NFTs are only exchanged when a valid trade cycle is detected, preventing incomplete or invalid transactions.

4.3.1 NFT states

All NFTs deployed in the blockchain environment of Barterplace have a state. A state diagram including the transition from one state to another and the definitions appears below.

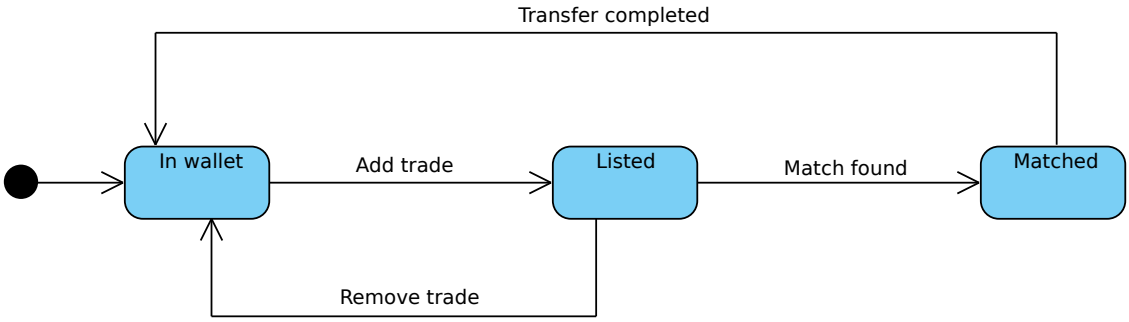


Figure 4.3: NFT state diagram

Definition of states:

1. *In wallet:* An NFT is created (minted), or a transaction is completed or removed from the Barterplace graph. As the state denotes, these NFTs are in the owner's "wallet".

2. *Listed:* An NFT with active outbound trades in the Barterplace graph. This means that it is temporarily locked in the smart contract's possession.

3. *Matched:* An NFT is included in a complete trade cycle. This transient state signifies that the NFT is about to be traded.

4.3.2 Add trade

Users can declare their intent to exchange one of their NFTs for another non-owned asset through a structured process:

1. Temporarily transfer ownership of the traded NFT to Barterplace.
2. Create an edge originating from the user's owned NFT and targeting the desired NFT.
3. Verify if the added edge forms a complete cycle within the graph.
4. If a cycle is detected:
   (a) Transfer ownership of each asset to the requesting user.
   (b) Remove all outbound edges from cycle nodes.
   (c) Eliminate incoming edges to the traded NFTs originating from their new asset owner.

UML Activity diagram

The activity diagram of the process of adding a trade to the Barterplace can be found below:

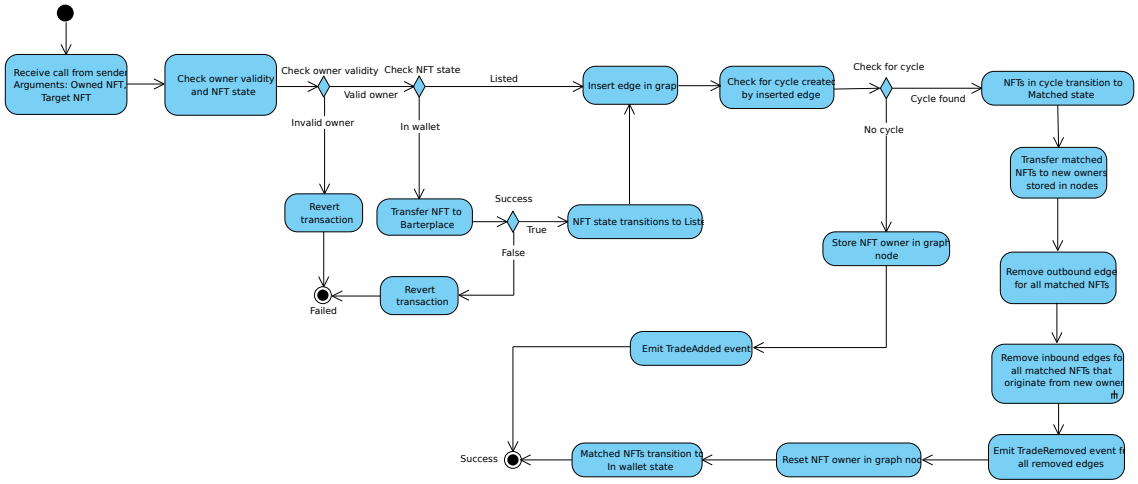


Figure 4.4: Add trade method activity diagram

4.3.3 Remove trade

Users retain the flexibility to cancel their submitted NFT trades at any point:

1. Remove the specific edge originating from an NFT owned by the user.
2. If the NFT no longer possesses outbound edges, restore ownership to the user.

UML Activity diagram

The activity diagram of the process of removing a trade from the Barterplace can be found below:

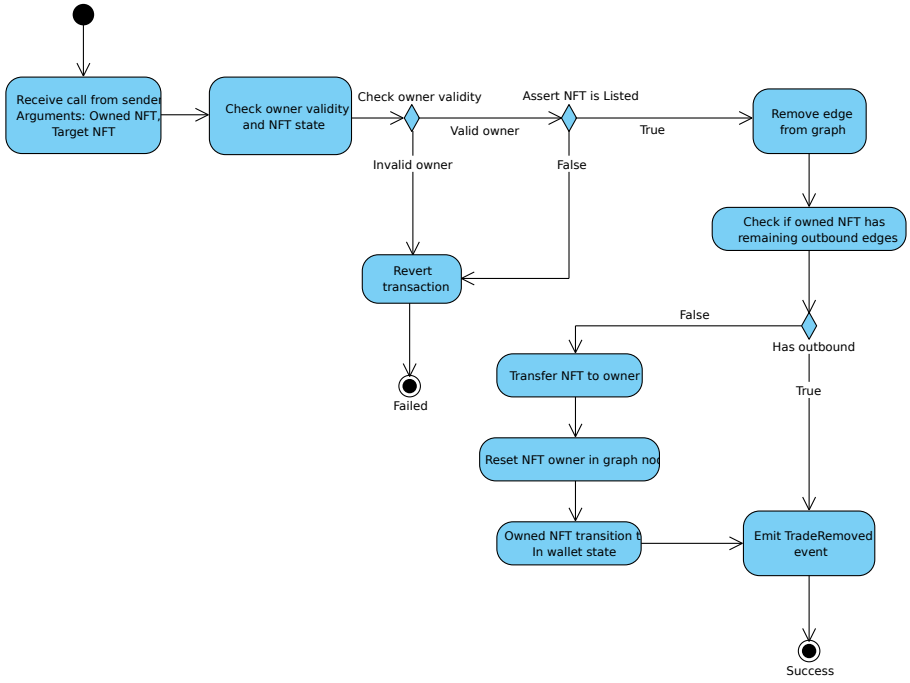


Figure 4.5: Remove trade method activity diagram

4.3.4 Cycle detection

When the `Add trade` method is called, an edge $e : (s, t)$ is inserted into the directed graph. This triggers a check for cycles that include edge e. The cycle detection is performed using a DFS traversal starting from node t, searching for an existing path $P : t \rightsquigarrow s$ in the graph.

Definition 4.3.1 (Cycle Detection Condition)**.** Given a directed graph $G = (V, E)$, when an edge $e : (s, t)$ is added to the graph, a cycle is formed if and only if there exists a directed path $P : t \rightsquigarrow s$ prior to the insertion of e.

As already demonstrated in Section 3.4, a single addition of a trade (edge) can create multiple cycles in the Barterplace graph. This case has been considered, resulting in a *conflict resolution policy* incorporated into the DFS path-finding algorithm, which makes the cycle search deterministic.

Definition 4.3.2 (Conflict Resolution Policy)**.** When an edge $e : (s, t)$ is added to the directed graph $G = (V, E)$, a DFS is initiated from t to check for the existence of a path $P : t \rightsquigarrow s$. The DFS explores edges in the order they were added, ensuring that they are processed chronologically at each recursive step.

Algorithm 2 Finds path P from $v_{start} \rightsquigarrow v_{target}$ using DFS and returns the sequence of edges forming P

```
1: procedure PathFindingDFS(G, v_start, v_target, edge_stack)
2:     found_path ← FALSE
3:     if v_start = v_target then
4:         found_path ← TRUE
5:         return found_path, edge_stack
6:     end if
7:     for each e_next in outbound edges of v_start do
8:         edge_stack.push(e_next)
9:         v_next ← target node of e_next
10:        found_path, path_edges ← PathFindingDFS(G, v_next, v_target, edge_stack)
11:        if found_path then
12:            return found_path, path_edges
13:        end if
14:        edge_stack.pop()
15:    end for
16:    return found_path, edge_stack
17: end procedure
```

In line 7 of the DFS path-finding algorithm is the point where the Conflict Resolution Policy 4.3.2 is being applied. The data structure that stores the outbound edges of each node maintains them in the chronological order in which users insert their trades.

4.3.5 Class diagram

A class diagram of the Barterplace smart contract and its libraries can be seen below:

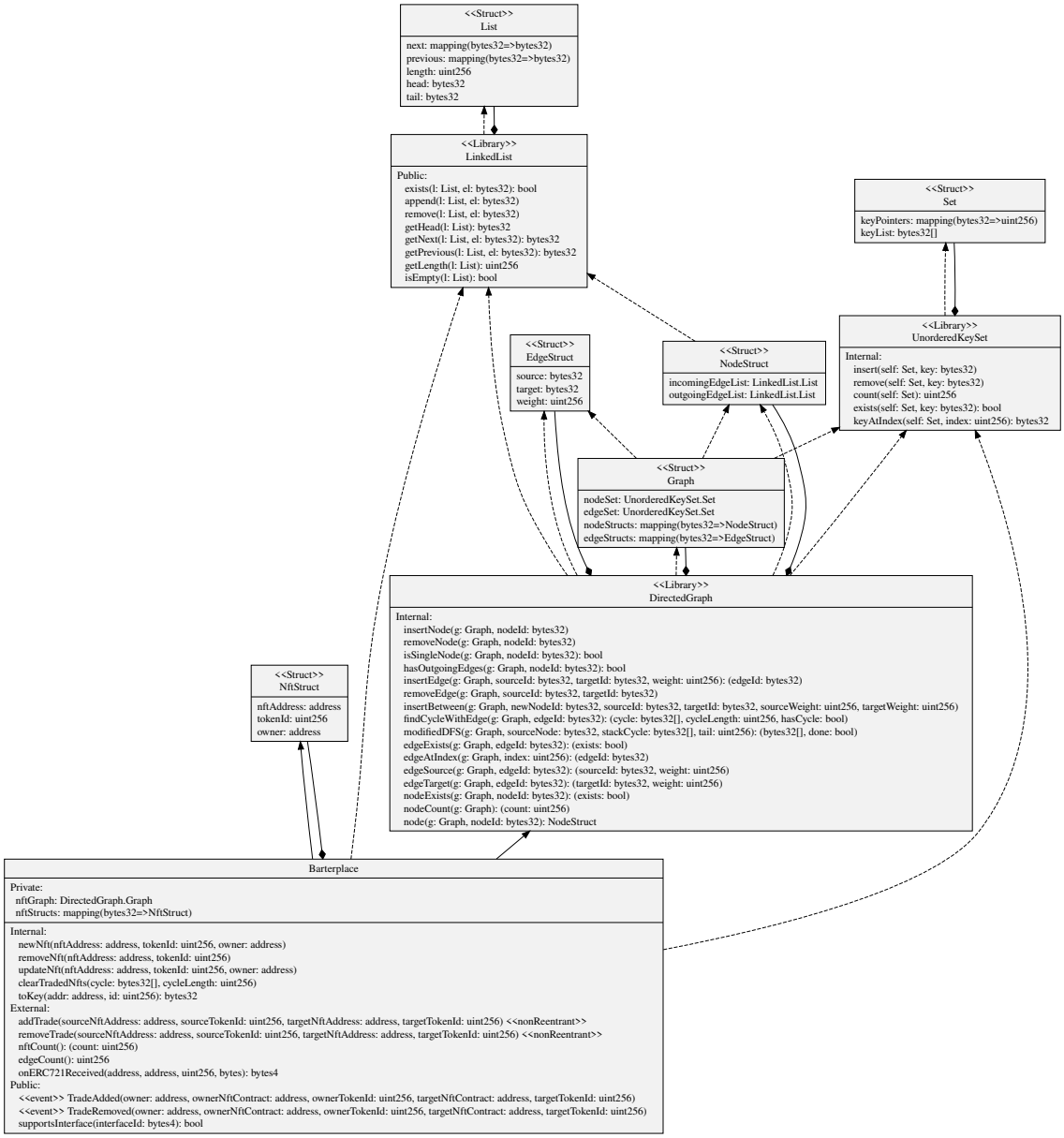


Figure 4.6: Barterplace smart contract class diagram

4.3.6 CLI (Command Line Interface)

A CLI was created that interfaces the methods of the smart contract. These actions are defined as Hardhat tasks [38].

Methods

1. *add-trade*:
   Calls Barterplace method addTrade

   Arguments:
   sourceAddress: Contract address of source NFT collection
   sourceId: ID of the source NFT
   targetAddress: Contract address of target NFT collection
   targetId: ID of the target NFT
   Options:
   --*account*: The account address of the transaction signer
   Example:
   npx hardhat add-trade 0x123... 21 0x456... 45 --account 0x789...

2. *remove-trade*:
   Calls Barterplace method removeTrade

   Arguments:
   sourceAddress: Contract address of source NFT collection
   sourceId: ID of the source NFT
   targetAddress: Contract address of target NFT collection
   targetId: ID of the target NFT
   Options:
   --*account*: The account address of the transaction signer
   Example:
   npx hardhat remove-trade 0x123... 21 0x456... 45 --account 0x789...

3. *deploy-barterplace*:
   Deploys Barterplace smart contract

   Options:
   --*account*: The account address of the transaction signer
   Example:
   npx hardhat deploy-barterplace --account 0x789...

4. *deploy-nft-collection*:
   Deploys ERC-721 NFT collection smart contract

   Arguments:
   collectionName: Name of the NFT collection
   Options:
   --account: The account address of the transaction signer
   Example:
   npx hardhat deploy-nft-collection MyFirstCollection --account 0x789...

5. *mint-nft*:
   Mints an NFT from the specified ERC-721 contract

   Arguments:
   collectionAddress: Contract address of NFT collection
   Options:
   --account: The account address of the transaction signer
   Example:
   npx hardhat mint-nft 0x123... --account 0x789...

6. *set-approval*:
   Calls ERC-721 contract to set approval for Barterplace as operator

   Arguments:
   collectionAddress: Contract address of NFT collection
   Options:
   --account: The account address of the transaction signer
   Example:
   npx hardhat set-approval 0x123... --account 0x789...

7. *transfer-nft*:
   Transfers an NFT to another account(wallet)

   Arguments:
   collectionAddress: Contract address of NFT collection
   tokenId: ID of the NFT
   targetAccountAddress: Address of the account that the NFT will be transferred to
   Options:
   --account: The account address of the transaction signer
   Example:
   npx hardhat transfer-nft 0x123... 45 0x987... --account 0x789...

4.4 Off-chain database

4.4.1 Motivation

The motivation behind deploying a database is to overcome two limitations from Ethereum's side.

The **first** limitation is the time-consuming calls to the blockchain. By replicating the exact state of the blockchain smart contract in an off-chain database, queries are made much faster, more flexible, and more extensible.

For the **second** issue, in Ethereum, there is no simple way for users to see all the NFTs in their "wallet". This is true because the NFTs are not in their owner's wallet, but instead the NFTs exist as data stored within the state of their ERC-721 smart contract, and the ownership is represented by a pointer to the owner's address [27]. A graphical representation of this relationship can be seen in Figure 4.7. This makes it difficult to discover all the NFTs owned by a specific account because we would have to call all NFT smart contracts of the network each time.

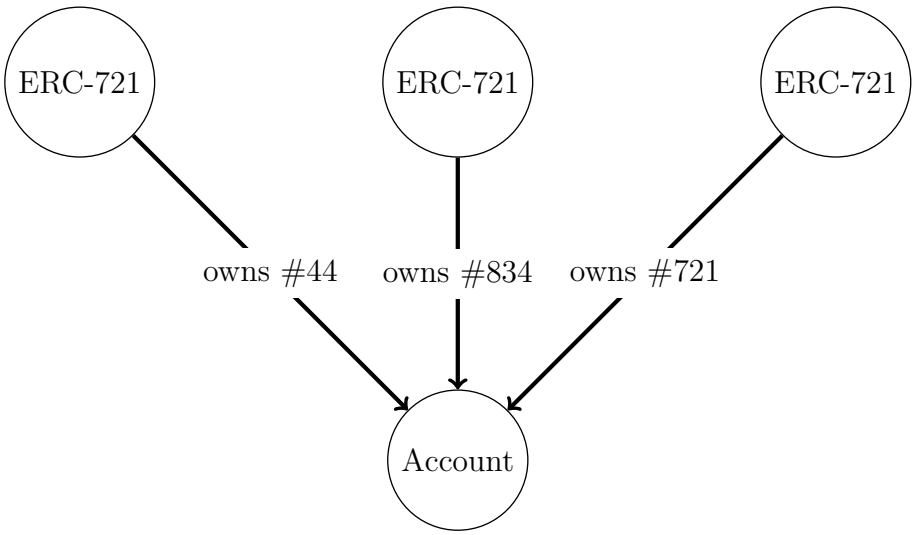


Figure 4.7: A graphical representation of NFTs in Ethereum

For the reasons outlined, efficient data retrieval and comprehensive visualization of NFT ownership are critical prerequisites for developing robust and user-friendly front-end interfaces in blockchain applications.

4.4.2 Graph Database

The technology used in the database is Neo4j graph database implementation. Neo4j is a JVM-based NoSQL database initially released in 2007 [40]. It was mainly selected because it is the most popular graph database, offering many useful features and functionality.

What are the advantages of a graph database?

1. Graphs provide a more natural way of representing data. A node can store all information about an entity, and related information can be displayed by the edges connected to it, making it easier to visualize [11].

2. Searching for information is more efficient than relational databases, as it takes advantage of proximity data from one or more starting points (main nodes) of the graph database [17].

3. For the Barterplace use case, it is the perfect fit since both are related to graph manipulation and their algorithms. Moreover, it helps with the Barterplace graph visualization through the Neo4j administrative graphical interface, which replaces the need for a visualization tool.

4.4.3 Schema

A graph database's schema is represented by a graph containing all types of nodes (labels) and edges (relationships).

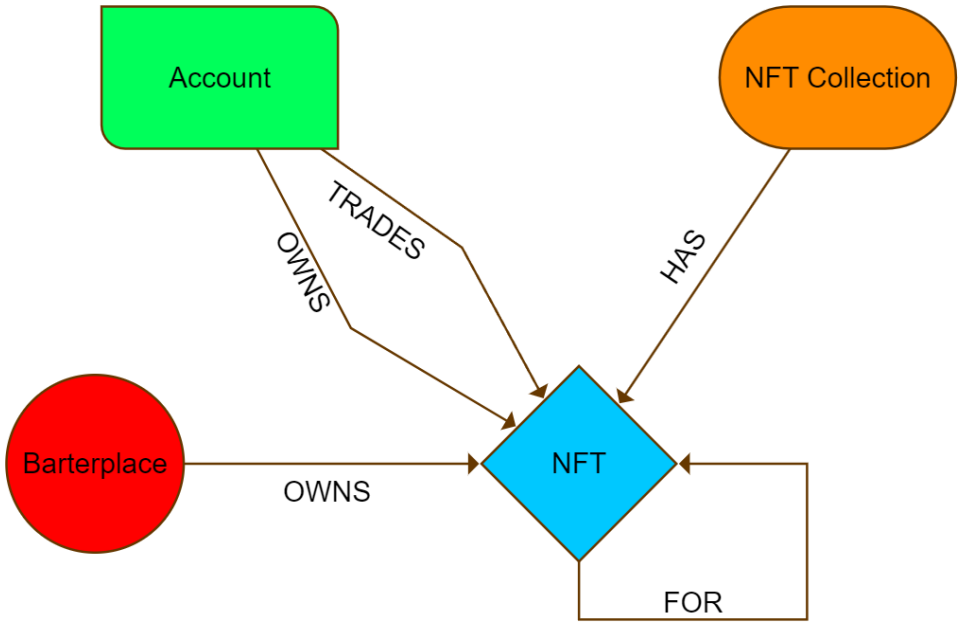


Figure 4.8: Graph database schema

Nodes

→ *Account*: Represents user accounts.

→ *NFT Collection*: Represents collections of NFTs.

→ *Barterplace*: Refers to the decentralized marketplace.

→ *NFT*: Represents a single NFT.

Relationships

→ *TRADES*: Indicates trading activity by accounts.

→ *OWNS*: Shows ownership of NFTs by accounts.

→ *HAS*: Shows which NFTs belong to which collections.

→ *FOR*: Represents NFT-for-NFT trade intents.

4.4.4 API

A REST API, implemented in Golang [45], exposes the database to the outside world. The endpoints that return a *list* of entries implement a pagination feature that separates large volumes of data into smaller chunks (pages). This is achieved with two option query parameters that determine the page size (*limit*) and a cursor where the last response stopped (*cursor*).

The supported endpoints cover the necessary use cases listed below:

Endpoints

1. **List all collections**:
   GET /collections
   Returns a list of all NFT collections.

   Parameters:
   limit - int (optional query parameter): Upper limit of returned entries
   cursor - string (optional query parameter): Used for pagination to fetch the next page

2. **List all NFTs per collection**:
   GET /collections/{contract_address}
   Returns a list of NFTs that exist under a given collection address.

   Parameters:
   contract_address - string: Address of collection contract
   include_metadata - boolean (optional query parameter): Include NFT metadata in the payload. Default value is *false*
   limit - int (optional query parameter): Upper limit of returned entries
   cursor - string (optional query parameter): Used for pagination to fetch the next page

3. **List all NFTs owned by an account**:
   GET /nfts/account/{account_address}
   Returns a list of NFTs owned by a specific account.

   Parameters:
   account_address - string: Address of account wallet
   include_metadata - boolean (optional query parameter): Include NFT metadata in the payload. Default value is *false*
   limit - int (optional query parameter): Upper limit of returned entries
   cursor - string (optional query parameter): Used for pagination to fetch the next page

4. **List NFT transfer history**:
   GET /nfts/{contract_address}/{token_id}/history
   Returns the transfer history of a specific NFT.

   Parameters:
   contract_address - string: Address of collection contract
   token_id - string: Token ID of the NFT
   limit - int (optional query parameter): Upper limit of returned entries
   cursor - string (optional query parameter): Used for pagination to fetch the next page

5. **List NFT active trades in Barterplace**:
   GET /nfts/{contract_address}/{token_id}/trades
   Returns the available outgoing edges that exist in Barterplace's graph for a specific NFT.

   Parameters:
   contract_address - string: Address of collection contract
   token_id - string: Token ID of the NFT
   include_metadata - boolean (optional query parameter): Include NFT metadata in the payload. Default value is *false*
   limit - int (optional query parameter): Upper limit of returned entries
   cursor - string (optional query parameter): Used for pagination to fetch the next page

6. **Get NFT metadata**:
   GET /nfts/{contract_address}/{token_id}/metadata
   Returns the on-chain metadata of a specific NFT.

   Parameters:
   contract_address - string: Address of collection contract
   token_id - string: Token ID of the NFT

7. **Get NFT owner**:
   GET /nfts/{contract_address}/{token_id}/owner
   Returns the owner of a specific NFT.

   Parameters:
   contract_address - string: Address of collection contract
   token_id - string: Token ID of the NFT

4.4.5 NFT discovery service

The NFT Discovery Service, implemented in Node.js [41], is responsible for tracking and updating the database with all NFTs and their on-chain states, processing transactions while listening to smart contract events emitted by the Barterplace contract, and maintaining synchronization with the blockchain by sequentially handling each newly published block.

Process Flow

Step 1: Service startup

The initialization of the NFT discovery service.

Step 2: Configuration loading

The service loads required configurations, including network settings, contract addresses, and API keys.

Step 3: Blockchain connection

The service establishes a connection with the Blockchain to listen for incoming blocks.

Step 4: New block detection

The Blockchain publishes a new block that includes NFT-related transactions.

Step 5: Parsing and filtering block data

The NFT discovery extracts relevant transactions and filters smart contract events.

Step 6: Processing smart contract events

The system listens for contract events such as:

- `TradeAdded` (an NFT trade was initiated)
- `TradeRemoved` (a trade was canceled)

Step 7: Database update

The extracted data is used to update the off-chain database, ensuring NFT records remain up to date.

Step 8: Synchronization check

If the database is outdated, the service triggers a historical data fetch to reconcile past transactions.

Step 9: Ready state

Once all previous steps are completed, the service enters an active listening state, continuously processing new blocks.

UML sequence diagrams

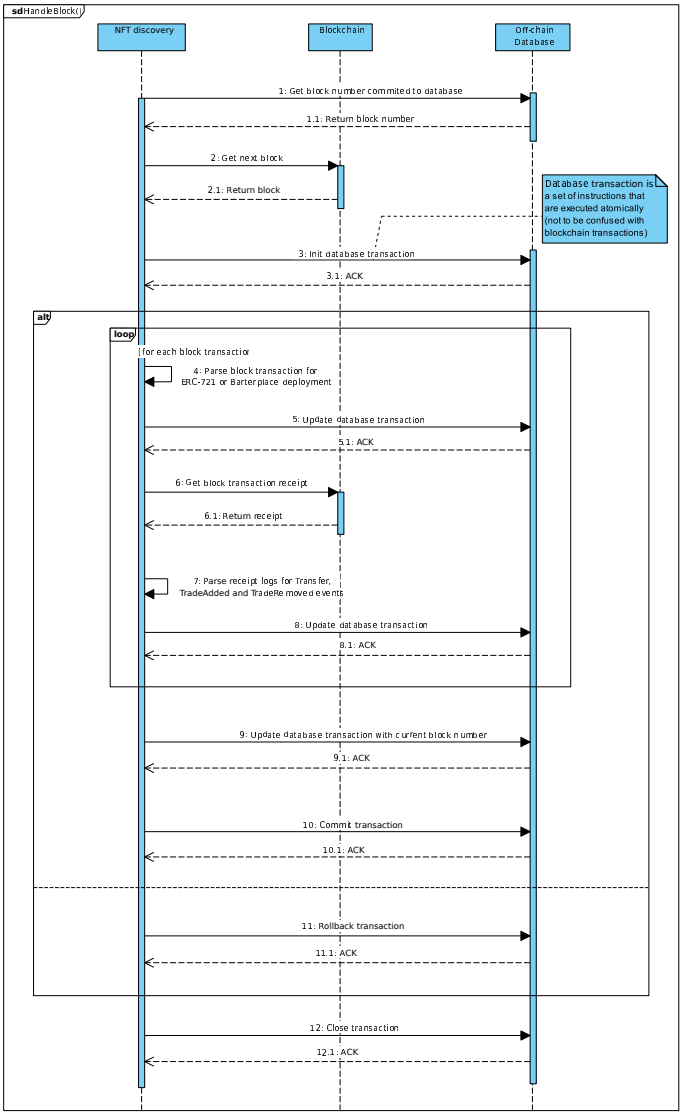


Figure 4.9: UML sequence diagram of HandleBlock function

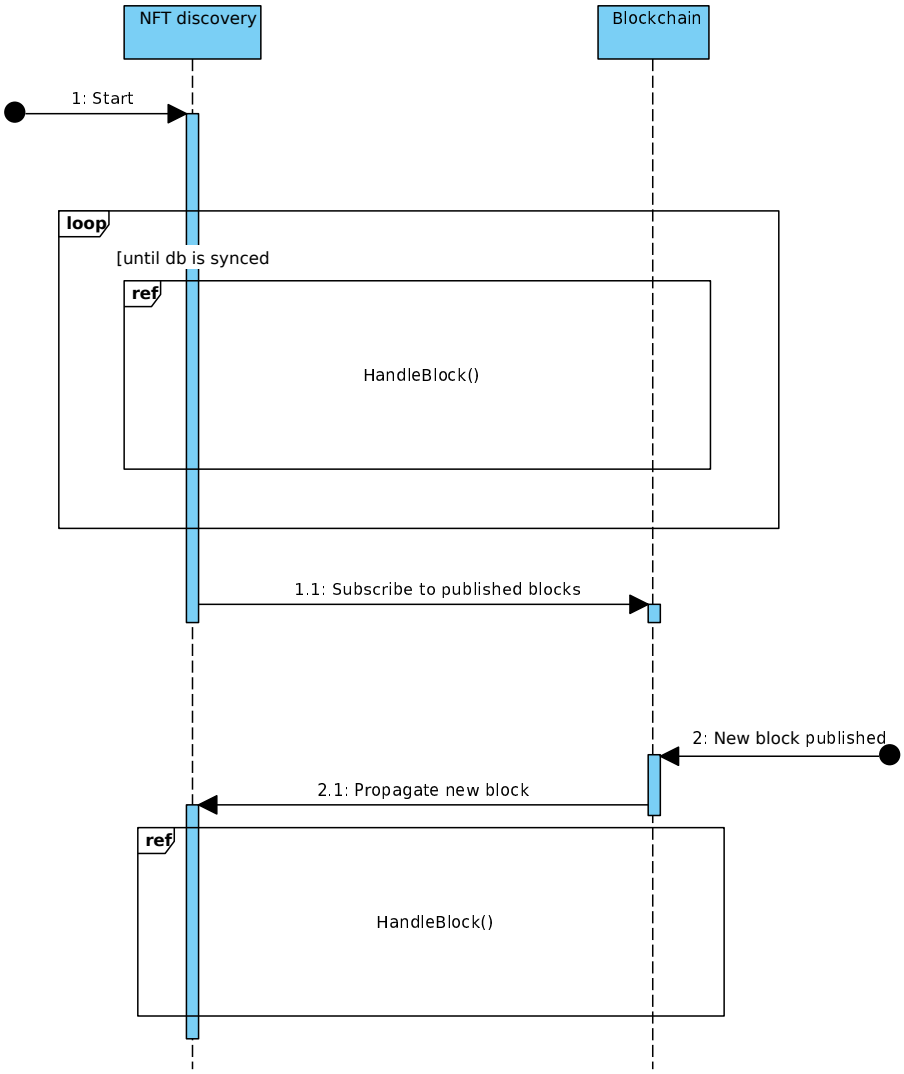


Figure 4.10: UML sequence diagram: initialization of NFT discovery service

Chapter 5

Disussion

The implementation of the NFT barterplace system revealed both promising potential and notable limitations that require careful examination. While the platform successfully demonstrates the feasibility of NFT-based bartering, several efficiency constraints emerged during testing, particularly in scalability. These limitations, along with emerging technological developments in the blockchain space, point to various opportunities for future enhancements and refinements. This chapter critically evaluates these constraints and explores potential improvements that could elevate the platform's functionality.

5.1 Gas cost analysis

A thorough analysis of the gas costs associated with the methods of the smart contract is essential to assess the efficiency of the implementation for the Barterplace project. Gas costs directly impact the usability and scalability of the platform, so understanding these costs is key to optimizing performance.

To evaluate the gas costs, a simulation was conducted with 1,000 users, each owning 5 NFTs. Random trade transactions were executed between NFTs owned by different users, simulating real-world activity on the platform. The results of the analysis were collected while the Barterplace contract was running on the Solidity compiler version 0.8.17.

The Ethereum network's block limit is set at 30 million gas, which is an important factor to consider, as it defines the maximum computational resources available for each transaction block. The gas cost measurements provide insights into the efficiency of the contract's methods, and help determine whether the Barterplace system can handle a large volume of users

and transactions effectively.

Method gas costs					
Contract	**Method**	**Min**	**Max**	**Avg**	**# Calls**
Barterplace	addTrade	375199	4762760	565738	5000
Barterplace	removeTrade	127983	137983	85434	52
SimpleCollection	mintNFT	82451	116651	86265	5000
SimpleCollection	setApprovalForAll	-	-	46648	1000

Table 5.1: Gas Usage analysis

Deployment gas costs		
Contract	**Avg**	**% of Limit**
Barterplace	2931923	9.8%
LinkedList	669509	2.2%
SimpleCollection	2535147	8.5%

Table 5.2: Deployment Information

The gas cost analysis reveals significant scalability concerns for the Barterplace smart contract. Notably, the *addTrade* function, with a maximum gas cost of $4,762,760$, consumes nearly one-sixth of the Ethereum block gas limit (30 million), making large-scale adoption impractical. This high cost arises from the complexity of cycle detection, which grows with the number of trades in the system. While other operations such as *removeTrade* remain relatively efficient, the trade execution overhead indicate that Barterplace, in its current form, would struggle to support high transaction volumes on a public blockchain. Future optimizations, such as reducing on-chain computations through off-chain operations or more gas-efficient data structures, are necessary to improve scalability.

5.2 Future improvements

In the context of this thesis, while not every objective was fully realized, the project successfully addressed key aspects of the problem space. Certain limitations, such as scalability constraints and technical challenges, influenced the scope of implementation, but they also provided valuable learning opportunities. Despite these challenges, the research yielded meaningful insights and identified promising directions for future enhancements and optimizations.

5.2.1 Smart contract optimization

The gas limit in Ethereum is an upper bound on the amount of computational effort a transaction or smart contract operation can consume, ensuring that operations are processed successfully and efficiently. However, for this scale of usage, the gas limits of Barterplace are manageable.

Smart contract optimization strategies can further enhance performance and reduce costs. Implementing gas-efficient coding patterns, such as minimizing storage writes and using calldata instead of memory, can significantly decrease gas fees. Loop unrolling and efficient use of mappings instead of arrays can further improve efficiency. Additionally, transaction batching allows multiple operations to be executed in a single call, reducing redundant execution costs [24]. Another promising optimization technique involves using proxy contracts for contract upgrades, preserving the storage state while allowing logic changes [28].

By combining these techniques, the Barterplace system can improve transaction efficiency, minimize costs, and enhance overall scalability.

5.2.2 Off-chain computation

Off-chain computation can significantly improve the efficiency of the Barterplace system by reducing the amount of on-chain processing required. By moving certain computationally expensive operations off-chain, the system can mitigate gas costs and alleviate the constraints imposed by the Ethereum Virtual Machine (EVM). One approach is to use an external verification system where trade path calculations and validations occur off-chain, with only the final transaction settlement of transfers recorded on-chain.

A potential implementation involves a separate smart contract that enforces correctness using collateral staking. Before execution, contract owners stake collateral, and trade paths are computed off-chain. If an invalid trade is submitted, users can provide fraud proof. Upon validation, the contract self-destructs, and the staked collateral is awarded to the submitter.

This mechanism deters fraud, reduces gas costs by offloading computation, and incentivizes accurate trade execution, ensuring a secure and efficient NFT barter system.

5.2.3 Rollups

Rollups are a layer two scaling solution that processes transactions off-chain while ensuring their validity on the main blockchain [29]. They bundle multiple transactions together and submit a compressed proof to the main

Ethereum network, reducing congestion and lowering transaction fees. There are two main types of rollups, Optimistic Rollups, which assume transactions are valid unless proven otherwise, and ZK-Rollups, which use zero-knowledge proofs to verify transactions instantly and securely.

By leveraging rollups, the Barterplace system can compute trade paths off-chain while ensuring security and efficiency. The off-chain system calculates optimal trade routes and submits a proof to the rollup contract. This proof is verified on-chain, allowing transactions to be executed in batches, significantly reducing gas costs. If an invalid trade path is detected, the rollup rejects it, preventing execution.

This method enhances scalability, minimizes on-chain processing, and maintains trustless execution through cryptographic proofs.

Chapter 6

Conclusion

This thesis introduced Barterplace, a decentralized NFT bartering system that leverages graph theory and blockchain technology to enable secure, trustless multi-party swaps. Unlike traditional NFT marketplaces, which focus on direct buying and selling, Barterplace utilizes directed graphs to represent trade intents and employs Depth-First Search (DFS) for cycle detection, ensuring seamless, automated swaps.

By implementing Ethereum smart contracts, the system guarantees security and transparency, while IPFS provides decentralized storage for NFT metadata. Key benefits of this approach include:

1. Trustless Trading – Eliminates intermediaries, reducing fees and enhancing security.

2. Improved Liquidity – Multi-party trades increase the possibility of asset exchange.

3. Automated & Fair Execution – Smart contracts enforce valid swaps without manual oversight.

While Barterplace demonstrates the feasibility of decentralized NFT bartering, several areas remain open for future improvement. One key focus is enhancing scalability, as the current gas costs, particularly for trade execution, limit real-world adoption. Optimizing smart contract logic, leveraging off-chain computation, or integrating layer-2 scaling solutions could significantly reduce transaction fees. Addressing these challenges will help refine Barterplace into a more scalable and efficient solution for decentralized asset exchange in the Web3 ecosystem.

Bibliography

[1] Leonhard Euler. «Solutio problematis ad geometriam situs pertinentis». In: *Commentarii academiae scientiarum Petropolitanae* (1741), pp. 128–140.

[2] Alan Mathison Turing et al. «On computable numbers, with an application to the Entscheidungsproblem». In: *J. of Math* 58.345-363 (1936), p. 5.

[3] Robert Tarjan. «Depth-first search and linear graph algorithms». In: *SIAM journal on computing* 1.2 (1972), pp. 146–160.

[4] John E Hopcroft, Jeffrey D Ullman, and Alfred Vaino Aho. *Data structures and algorithms*. Vol. 175. Addison-wesley Boston, MA, USA: 1983.

[5] Stuart Haber and W. Scott Stornetta. «How to Time-Stamp a Digital Document». In: *Advances in Cryptology-CRYPTO' 90*. Ed. by Alfred J. Menezes and Scott A. Vanstone. Berlin, Heidelberg: Springer Berlin Heidelberg, 1991, pp. 437–455. ISBN: 978-3-540-38424-3.

[6] Nick Szabo. «Smart contracts: building blocks for digital markets». In: *EXTROPY: The Journal of Transhumanist Thought,(16)* 18.2 (1996), p. 28.

[7] Reinhard Diestel. *Graph theory*. 2000.

[8] Douglas Brent West et al. *Introduction to graph theory*. Vol. 2. Prentice hall Upper Saddle River, 2001.

[9] John R Douceur. «The sybil attack». In: *International workshop on peer-to-peer systems*. Springer. 2002, pp. 251–260.

[10] Mark EJ Newman. «The structure and function of complex networks». In: *SIAM review* 45.2 (2003), pp. 167–256.

[11] Renzo Angles and Claudio Gutierrez. «Survey of Graph Database Models». In: *ACM Comput. Surv.* 40.1 (Feb. 2008). ISSN: 0360-0300. DOI: 10.1145/1322432.1322433. URL: https://doi.org/10.1145/1322432.1322433.

[12] Satoshi Nakamoto. «Bitcoin: A Peer-to-Peer Electronic Cash System». In: (May 2009). URL: http://www.bitcoin.org/bitcoin.pdf.

[13] Juan Benet. «IPFS - Content Addressed, Versioned, P2P File System». In: *CoRR* abs/1407.3561 (2014). arXiv: 1407.3561. URL: http://arxiv.org/abs/1407.3561.

[14] Vitalik Buterin et al. «A next-generation smart contract and decentralized application platform». In: *white paper* 3.37 (2014), pp. 2–1.

[15] Jeffrey Watumull et al. «On recursion». In: *Frontiers in Psychology* 4 (2014), p. 1017.

[16] Gavin Wood et al. «Ethereum: A secure decentralised generalised transaction ledger». In: *Ethereum project yellow paper* 151.2014 (2014), pp. 1–32.

[17] José Guia, Valéria Gonçalves Soares, and Jorge Bernardino. «Graph Databases: Neo4j Analysis». In: *ICEIS*. 2017, pp. 351–356.

[18] Bojana Koteska, Elena Karafiloski, and Anastas Mishev. «Blockchain Implementation Quality Challenges: A Literature Review». In: Sept. 2017.

[19] Wei Cai et al. «Decentralized Applications: The Blockchain-Empowered Software System». In: *IEEE Access* 6 (2018), pp. 53019–53033. DOI: 10.1109/ACCESS.2018.2870644.

[20] Sotirios Brotsis et al. «Blockchain solutions for forensic evidence preservation in IoT environments». In: *2019 IEEE Conference on Network Softwarization (NetSoft)*. IEEE. 2019, pp. 110–114.

[21] Michael Egorov. «Stableswap-efficient mechanism for stablecoin liquidity». In: *Retrieved Feb* 24 (2019), p. 2021.

[22] Rob Hitchens. *Hitchens Unordered Key Set*. Github. 2019. URL: https://github.com/rob-Hitchens/UnorderedKeySet.

[23] Huashan Chen et al. «A survey on ethereum systems security: Vulnerabilities, attacks, and defenses». In: *ACM Computing Surveys (CSUR)* 53.3 (2020), pp. 1–43.

[24] Lodovica Marchesi et al. «Design patterns for gas optimization in ethereum». In: *2020 IEEE International Workshop on Blockchain Oriented Software Engineering (IWBOSE)*. IEEE. 2020, pp. 9–15.

[25] Bahareh Lashkari and Petr Musilek. «A Comprehensive Review of Blockchain Consensus Mechanisms». In: *IEEE Access* 9 (2021), pp. 43620–43652. DOI: 10.1109/ACCESS.2021.3065880.

[26] Sandeep Kumar Panda and Suresh Chandra Satapathy. «An Investigation into Smart Contract Deployment on Ethereum Platform Using Web3.js and Solidity Using Blockchain». In: *Data Engineering and Intelligent Computing*. Ed. by Vikrant Bhateja et al. Singapore: Springer Singapore, 2021, pp. 549–561. ISBN: 978-981-16-0171-2.

[27] Qin Wang et al. *Non-Fungible Token (NFT): Overview, Evaluation, Opportunities and Challenges*. 2021. DOI: 10.48550/ARXIV.2105.07447. URL: https://arxiv.org/abs/2105.07447.

[28] Mehdi Salehi, Jeremy Clark, and Mohammad Mannan. «Not so immutable: Upgradeability of smart contracts on ethereum». In: *International Conference on Financial Cryptography and Data Security*. Springer. 2022, pp. 539–554.

[29] Louis Tremblay Thibault, Tom Sarry, and Abdelhakim Senhaji Hafid. «Blockchain scaling using rollups: A comprehensive survey». In: *IEEE Access* 10 (2022), pp. 93039–93054.

[30] Ingo Fiedler and Lennart Ante. «Stablecoins». In: *The Emerald Handbook on Cryptoassets: Investment Opportunities and Challenges*. 2023.

[31] Barbara Guidi and Andrea Michienzi. «From NFT 1.0 to NFT 2.0: A review of the evolution of non-fungible tokens». In: *Future Internet* 15.6 (2023), p. 189.

[32] Qaiser Razi et al. «Non-fungible tokens (NFTs)-survey of current applications, evolution and future directions». In: *IEEE Open Journal of the Communications Society* (2023).

[33] David Rodeck. *Top NFT Marketplaces Of 2023*. Ed. by Benjamin Curry. Jan. 2023. URL: https://www.forbes.com/advisor/investing/cryptocurrency/best-nft-marketplaces/.

[34] James Chen. *What is an escrow agreement? how it works, uses, and types*. URL: https://www.investopedia.com/terms/e/escrowagreement.asp.

[35] *Curve*. URL: https://curve.fi/.

[36] *Ethers.js*. URL: https://docs.ethers.org/.

[37] *go-ethereum*. URL: https://geth.ethereum.org/.

[38] *Hardhat*. URL: https://hardhat.org/.

[39] *ipfs/kubo: An IPFS implementation in Go*. URL: https://github.com/ipfs/kubo.

[40] *Neo4j Graph Database & Analytics*. URL: https://neo4j.com/.

[41] *Node.js*. URL: https://nodejs.org/.

[42] *OpenZeppelin*. URL: https://www.openzeppelin.com/.

[43] *Remix - Ethereum IDE & community*. URL: https://remix-project.org/.

[44] *SushiSwap*. URL: https://www.sushi.com/.

[45] *The Go Programming Language*. URL: https://go.dev/.

[46] *Uniswap Protocol*. URL: https://uniswap.org/.

[47] *Web3.js*. URL: https://web3js.readthedocs.io/.

Κεφάλαιο 1

Εισαγωγή

Ιστορικό

Η τεχνολογία blockchain έχει εξελιχθεί σε ισχυρό εργαλείο για την αποκεντρωμένη και ασφαλή καταγραφή δεδομένων. Τα NFTs αποτελούν ένα καινοτόμο παράδειγμα χρήσης, επιτρέποντας την ιδιοκτησία και ανταλλαγή ψηφιακών αγαθών. Αν και οι περισσότερες αγορές NFTs επικεντρώνονται σε μονόπλευρες αγοραπωλησίες, παραμένει αναξιοποίητη η δυνατότητα για πολυμερείς ανταλλαγές χωρίς μεσάζοντες.

Επιπλέον, οι υπάρχουσες λύσεις, είτε centralized είτε decentralized, επικεντρώνονται κυρίως σε λειτουργίες δημοπρασιών ή σταθερών τιμών. Η έννοια της άμεσης, πολυμερούς ανταλλαγής βάσει των επιθυμιών των χρηστών παραμένει ανεκμετάλευτη. Αυτή η εργασία στοχεύει στην αξιοποίηση αυτής της δυνατότητας.

Η πρόταση

Για να καλυφθεί αυτό το κενό, προτείνεται το Barterplace, ένα νέο μοντέλο όπου οι προθέσεις ανταλλαγής αποτυπώνονται ως ακμές σε έναν γράφο και οι έγκυρες ανταλλαγές πραγματοποιούνται όταν σχηματίζεται κύκλος. Αυτό το μοντέλο επιτρέπει δυναμικές, πολυμερείς ανταλλαγές NFTs και αξιοποιεί smart contracts για την ασφάλεια και αυτοματοποίηση των συναλλαγών.

Το σύστημα υλοποιείται πάνω στο Ethereum χρησιμοποιώντας Solidity για τα έξυπνα συμβόλαια (σμαρτ ζοντρακτς) και αλγορίθμους γράφων για την επεξεργασία των σχέσεων ανταλλαγής. Επιπλέον, η χρήση αποκεντρωμένης αποθήκευσης αρχείων μέσω του IPFS εξασφαλίζει την ακεραιότητα των μεταδατα των NFTs.

Κεφάλαιο 2

Θεωρητικό υπόβαθρο

2.1 Blockchain

Η τεχνολογία blockchain προτάθηκε αρχικά το 1991 για την χρονοσήμανση ψηφιακών εγγράφων, αλλά έλαβε ευρεία αναγνώριση με την κυκλοφορία του Bitcoin από τον Satoshi Nakamoto το 2008. Το blockchain λειτουργεί ως μια αποκεντρωμένη λογιστική βάση δεδομένων, αξιοποιώντας την κρυπτογραφία για τη διασφάλιση της ακεραιότητας και διαφάνειας των συναλλαγών.

2.1.1 Κύρια μέρη

2.1.1.1 Block and chain

Κάθε block περιέχει δεδομένα (π.χ. συναλλαγές), ένα hash το οποίο αποτελεί ένα μοναδικό αναγνωριστικό για κάθε block, καθώς επίσης και το hash του προηγούμενου block. Αυτή η αλυσίδα από hashes εξασφαλίζει την ακεραιότητα των δεδομένων, καθώς οποιαδήποτε αλλοίωση καταστρέφει την ακεραιότητα της αλυσίδας.

2.1.1.2 Nodes and Network

Οι κόμβοι του δικτύου (nodes) αποθηκεύουν και διανέμουν το blockchain. Λειτουργούν ως επικυρωτές και εφαρμόζουν το μηχανισμό συναίνεσης. Η απουσία κεντρικού διακομιστή καθιστά το σύστημα ανθεκτικό σε επιθέσεις και σφάλματα.

2.1.1.3 Συναλλαγές

Οι συναλλαγές (transactions) αποτελούν τις βασικές μονάδες δράσης και περιλαμβάνουν την ανταλλαγή δεδομένων ή του εκάστοτε κρυπτονομίσματος του κάθε δικτύου. Υπογράφονται ψηφιακά και επικυρώνονται πριν εισαχθούν στο blockchain, αποκτώντας μόνιμη και αμετάβλητη μορφή.

2.1.1.4 Μηχανισμοί συναίνεσης

Οι μηχανισμοί συναίνεσης (consensus mechanisms) εξασφαλίζουν ότι όλοι οι κόμβοι συμφωνούν στην τρέχουσα κατάσταση της αλυσίδας. Το Proof of Work απαιτεί υπολογιστική εργασία, ενώ το Proof of Stake βασίζεται στο κλείδωμα κρυπτονομισμάτων. Άλλες προσεγγίσεις περιλαμβάνουν PoA (Proof of Authority), DPoS (Delegated Proof of Stake) και BFT (Byzantine Fault Tolerance), προσαρμοσμένες σε διαφορετικές ανάγκες.

2.1.2 Έξυπνα συμβόλαια και NFTs

Τα έξυπνα συμβόλαια είναι αυτόματα εκτελούμενες συμφωνίες, γραμμένες σε κώδικα, οι οποίες υλοποιούνται όταν πληρούνται συγκεκριμένοι όροι. Αξιοποιούνται στο Ethereum και σε άλλα δίκτυα για τη διαχείριση dApps (decentralized apps) αποκεντρωμένες εφαρμογές για την αυτοματοποίηση πληρωμών και διαχείριση πνευματικών δικαιωμάτων.

Τα NFTs είναι μοναδικά ψηφιακά περιουσιακά στοιχεία, βασισμένα σε πρότυπα όπως ERC-721 και ERC-1155. Περιλαμβάνουν metadata, μηχανισμούς ιδιοκτησίας και royalties για δημιουργούς. Οι αγορές NFTs όπως το OpenSea επιτρέπουν την αγοραπωλησία τους.

2.1.3 Βασικά στοιχεία

2.1.3.1 Αποκέντρωση

Η αποκέντρωση (decentralization) σημαίνει ότι κανένας μεμονωμένος φορέας δεν ελέγχει το blockchain. Αυτό διασφαλίζει ανθεκτικότητα και αμεροληψία στις συναλλαγές.

2.1.3.2 Διαφάνεια

Η διαφάνεια(transparency) επιτρέπει στους συμμετέχοντες να επαληθεύουν συναλλαγές δημόσια. Το ιστορικό είναι κοινόχρηστο, μόνιμο και προσβάσιμο, γεγονός που ενισχύει την εμπιστοσύνη.

2.1.3.3 Αμεταβλητότητα

Η αμετάβλητη φύση(immutability) του blockchain αποτρέπει την τροποποίηση δεδομένων μετά την καταγραφή τους, γεγονός που καθιστά αξιόπιστο το ιστορικό του.

2.1.3.4 Ασφάλεια

Η ασφάλεια(security) επιτυγχάνεται μέσω της κρυπτογραφίας, του ελέγχου των συναλλαγών, και την κατανομή των δεδομένων σε διάφορους κόμβους. Το δίκτυο προστατεύεται από επιθέσεις τύπου 51% μέσω της αποκέντρωσης.

2.2 Θεωρία γράφων

Η θεωρία γράφων (graph theory) μελετά σχέσεις μεταξύ οντοτήτων με χρήση κόμβων και ακμών. Οι γράφοι εφαρμόζονται σε πολλά πεδία, από τη χαρτογράφηση δικτύων έως την αναπαράσταση συναλλαγών.

2.2.1 Εφαρμογές στην Επιστήμη των Υπολογιστών

Στην Επιστήμη των Υπολογιστών(computer science), οι γράφοι χρησιμοποιούνται από αλγορίθμους αναζήτησης Bread-First Search(αναζήτηση κατά πλάτος), DFS (αναζήτηση κατά βάθος) για την βελτιστοποίηση και ανάλυση διάφορων δικτύων. Ο DFS στην προκειμένη περίπτωση, προσφέρει τρόπο αναζήτησης κόμβων και μονοπατιών με πολυπλοκότητα $O(V + E)$.

2.2.2 Γράφοι και Blockchain

Οι ανταλλαγές NFTs μπορούν να μοντελοποιηθούν ως γράφοι, με τα NFTs ως κόμβους και τις επιθυμίες ανταλλαγών ως ακμές. Η ανάλυση αυτών των γραφημάτων αποκαλύπτει μοτίβα, κύκλους και συγκεντρώσεις δραστηριότητας. Η χρήση λίστας γειτνίασης διευκολύνει την αναπαράσταση και υλοποίηση DFS για την εξερεύνηση του γράφου.

Ορισμός .1 (Λίστα γειτνίασης)**.** Σε έναν **κατευθυνόμενο γράφο** $G = (V, E)$, η λίστα γειτνίασης (adjacency list) αντιστοιχίζει κάθε κόμβο $u \in V$ σε μία λίστα κόμβων v τέτοια ώστε να υπάρχει κατευθυνόμενη ακμή από τον u προς τον v, δηλαδή $(u, v) \in E$.

Κεφάλαιο 3

Εννοιολογική προσέγγιση του Barterplace

3.1 Ο γράφος

Στην καρδιά του Barterplace βρίσκεται μια δομή κατευθυνόμενου γράφου, η οποία αναπαριστά τις προθέσεις ανταλλαγής NFTs. Ο γράφος υλοποιείται και διατηρείται από ένα έξυπνο συμβόλαιο(smart contract), προσφέροντας τη δυνατότητα για ασφαλείς και πολυμερείς ανταλλαγές NFTs χωρίς την ανάγκη για ανταλλαγές αποκλειστικά 1 προς 1.

- **Κόμβοι ως NFTs:** Κάθε κόμβος αντιστοιχεί σε ένα μοναδικό NFT με όλα τα απαραίτητα metadata.

- **Ακμές ως προθέσεις ανταλλαγής:** Κάθε ακμή συμβολίζει την επιθυμία ενός κατόχου να ανταλλάξει το NFT του με ένα άλλο.

- **Κύκλοι ως έγκυρες ανταλλαγές:** Ένας κύκλος στον γράφο σηματοδοτεί μια αλληλουχία αμοιβαίων ανταλλαγών που μπορούν να πραγματοποιηθούν ταυτόχρονα.

3.2 Προσθήκη πρόθεσης ανταλλαγής

Παράδειγμα 1: Alice → Bob

Η Alice, κάτοχος του NFT *duck*, επιθυμεί να το ανταλλάξει με το NFT *dog* του Bob. Η πρόθεσή της αποτυπώνεται ως μια ακμή από το $\delta\upsilon\varsigma\kappa$ προς το $\delta o\gamma$.

Σχήμα 3.1: Αναπαράσταση πρόθεσης ανταλλαγής μεταξύ δύο NFTs.

Η κάθε ακμή συνοδεύεται από metadata του ιδιοκτήτη, πράγμα που την κάνει εύκολα προσπελάσιμη από την χρήστη και δίνεται η δυνατότητα ακύρωσής της.

3.3 Εντοπισμός κύκλου ανταλλαγής

Παράδειγμα 2: Bob → Alice

Ο Bob προσθέτει με τη σειρά του πρόθεση ανταλλαγής του dog για το duck. Δημιουργείται κύκλος, επιτρέποντας την άμεση εκτέλεση της ανταλλαγής.

Σχήμα 3.2: Ολοκληρωμένος κύκλος ανταλλαγής μεταξύ Alice και Bob.

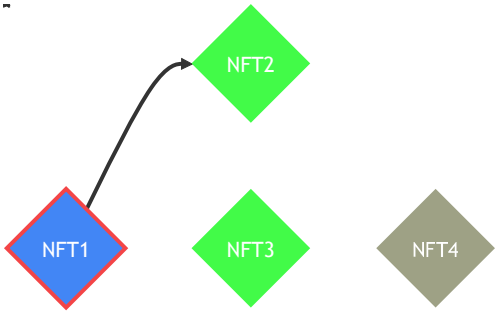
Ο κύκλος ανιχνεύεται με έναν τροποποιημένο αλγόριθμο DFS που ξεκινά από τον προορισμό της τελευταίας ακμής που προστέθηκε και ακολουθεί ακμές προς τα πίσω μέχρι να βρεθεί μονοπάτι στην αρχή της τελευταίας ακμής.

3.4 Επιλεγμένα παραδείγματα σύνθετων ανταλλαγών

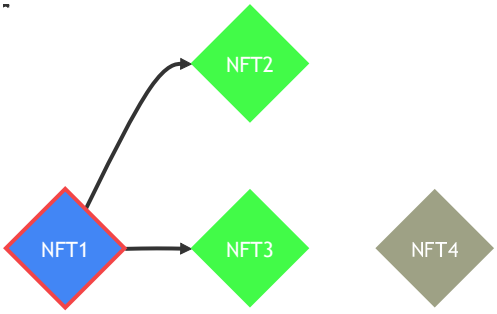
Προσθήκη προθέσεων ανταλλαγής μεταξύ 3 NFTs

Οι κάτοχοι των NFTs 1 (μπλε), 2 και 3 (πράσινος), 4 (καφέ) δηλώνουν διαδοχικές προθέσεις ανταλλαγής:

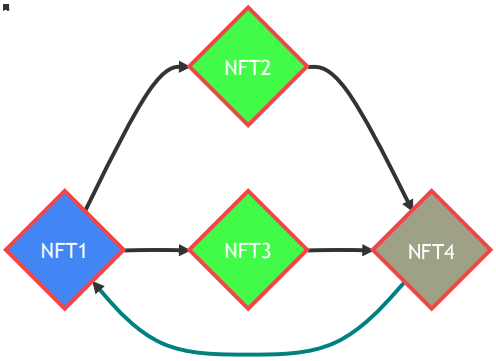
- NFT 1 → NFT 2
- NFT 1 → NFT 3
- NFT 2 → NFT 4
- NFT 3 → NFT 4
- NFT 4 → NFT 1 (κλείνει τον κύκλο)



(α′) Ανταλλαγή NFT 1 → NFT 2



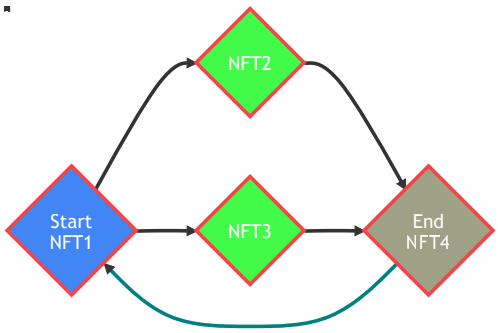
(β′) Ανταλλαγή NFT 1 → NFT 3



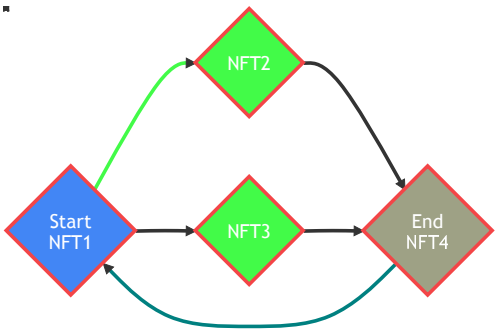
Σχήμα 3.4: Ανταλλαγή NFT 4 → NFT 1 και δημιουργία κύκλου.

Ανίχνευση κύκλου με DFS

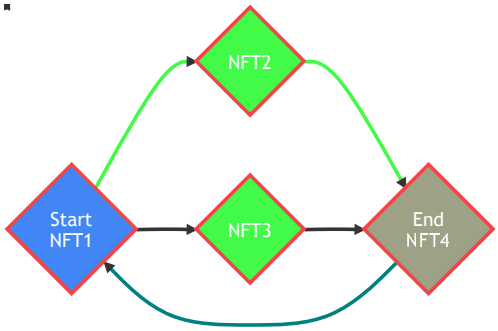
Ο DFS ξεκινά από το NFT 1 και ακολουθεί την πορεία NFT 1 → NFT 2 → NFT 4. Η εύρεση αυτής της διαδρομής υποδεικνύει κύκλο ανταλλαγής.



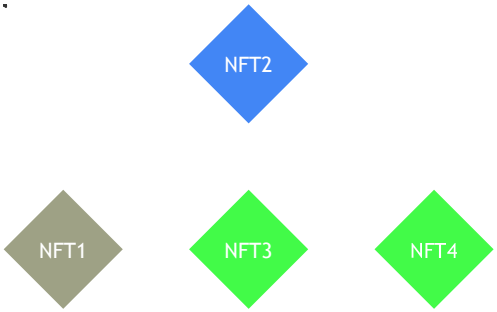
(α′) Εκκίνηση DFS από NFT 1



(β′) Ακμή NFT 1 → NFT 2



(α′) Ακμή NFT 2 → NFT 4, εύρεση κύκλου



(β′) Ολοκλήρωση μεταβίβασης NFTs

Η πορεία που εντοπίζεται από τον DFS είναι:

$$P = (\text{ΝΦΤ } 1, \text{ΝΦΤ } 2, \text{ΝΦΤ } 4)$$

και αποτελεί αναγαία και ικανή συνθήκη για την εκτέλεση των μεταβιβάσεων.

Κεφάλαιο 4

Υλοποίηση του έργου

4.1 Ethereum και Solidity

Το project βασίζεται στο Ethereum και σε έξυπνα συμβόλαια γραμμένα σε Solidity. Το Ethereum παρέχει την κατάλληλη υποδομή για την υλοποίηση αποκεντρωμένων εφαρμογών μέσω του Ethereum Virtual Machine (EVM).

4.1.1 Ethereum Virtual Machine (EVM)

4.1.1.1 Αρχιτεκτονική

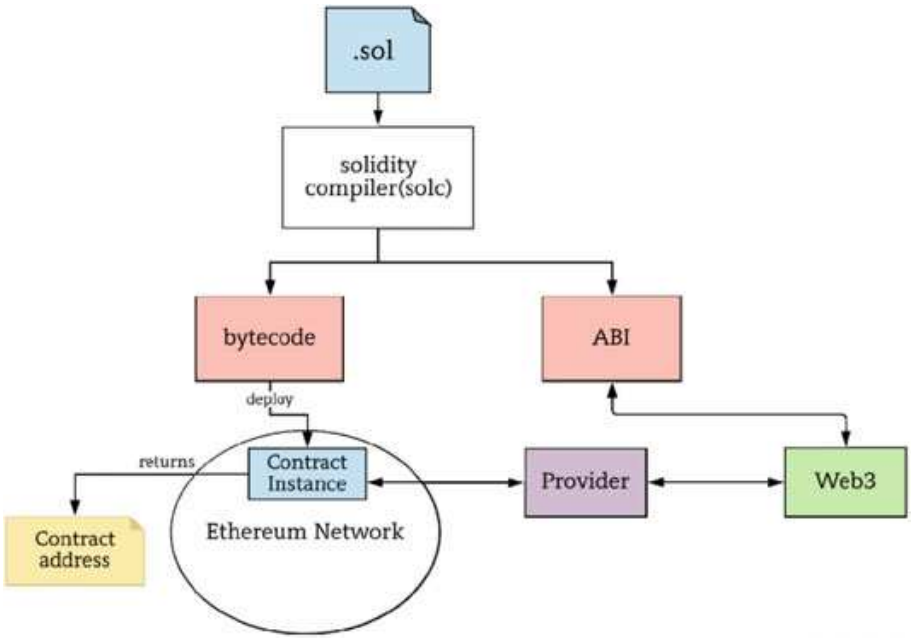
Το EVM είναι μία Turing-complete υπολογιστική μηχανή, βασισμένη σε στοίβα, η οποία εκτελεί bytecode που παράγεται από κώδικα γραμμένο σε Solidity. Η εκτέλεση είναι ντετερμινιστική σε όλους τους κόμβους του δικτύου, εξασφαλίζοντας τη συναίνεση.

4.1.1.2 Διαχείριση πόρων και Gas

Το EVM αξιοποιεί το μηχανισμό gas για τον έλεγχο της απαιτούμενων πόρων κατά την εκτέλεση. Κάθε εντολή έχει ένα κόστος και ο χρήστης ορίζει το gas limit σε κάθε συνναλαγή. Το αχρησιμοποίητο gas επιστρέφεται ενώ το χρησιμοποιημένο καταναλώνεται από τους επαληθευτές του δικτύου.

4.1.1.3 Ανάπτυξη και εκτέλεση

Τα έξυπνα συμβόλαια αναπτύσσονται μέσω συναλλαγών και αποκτούν μόνιμη διεύθυνση στο blockchain. Είναι μη τροποποιήσιμα μετά το deployment τους.



Σχήμα 4.1: Ροή ανάπτυξης smart contract στο Ethereum.

4.1.2 Εργαλεία ανάπτυξης

Στη διαδικασία ανάπτυξης του Barterplace χρησιμοποιήθηκαν:

- Remix IDE για δοκιμές του κώδικα και κλήση έξυπνων συμβολαίων
- Hardhat για scripts, testing και δελποψμεντς
- Ethers.js για αλληλεπίδραση προγραμματιστικά με τα έξυπνα συμβόλαια

4.2 IPFS

4.2.1 Τι είναι το IPFS

Το IPFS (InterPlanetary File System) είναι ένα αποκεντρωμένο σύστημα αποθήκευσης αρχείων. Κάθε αρχείο έχει μοναδικό hash που χρησιμεύει ως αναφορά.

4.2.2 Deployment

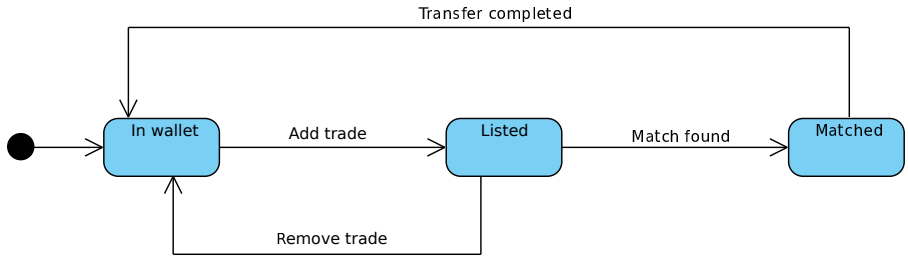
Τα metadata των NFTs αποθηκεύονται συνήθως εκτός του blockchain, όπως στο IPFS. Ωστόσο, η αναγνώριση και επιβεβαίωση της ιδιοκτησίας γίνεται μέσω του ίδιου του blockchain.

4.3 Έξυπνο συμβόλαιο Βαρτερπλαςε

Το συμβόλαιο του Barterplace υποστηρίζει αποθήκευση/διαχείριση NFTs και προθέσεων ανταλλαγής.

4.3.1 Καταστάσεις NFT

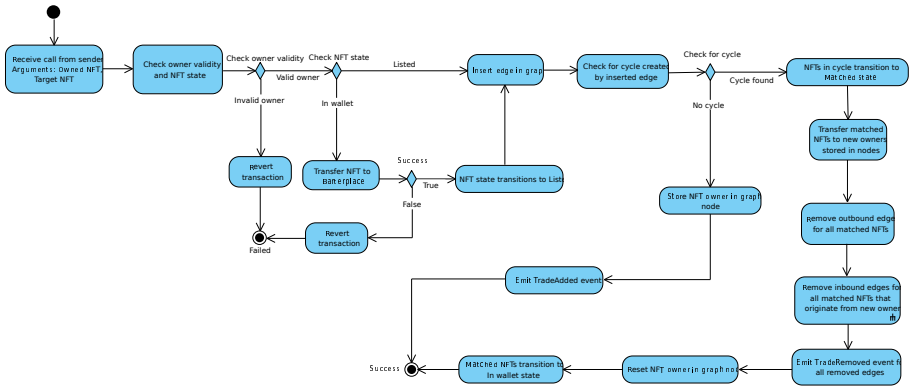
Τα NFTs μπορούν να είναι σε κατάσταση διαθέσιμη, υπό ανταλλαγή ή κλειδωμένη.



Σχήμα 4.2: Διάγραμμα με τις δυνατές καταστάσεις ενός ΝΦΤ (διαθέσιμο, καταχωρισμένο, υπό αντιστοίχιση).

4.3.2 Μέθοδος Προσθήκης Ανταλλαγής

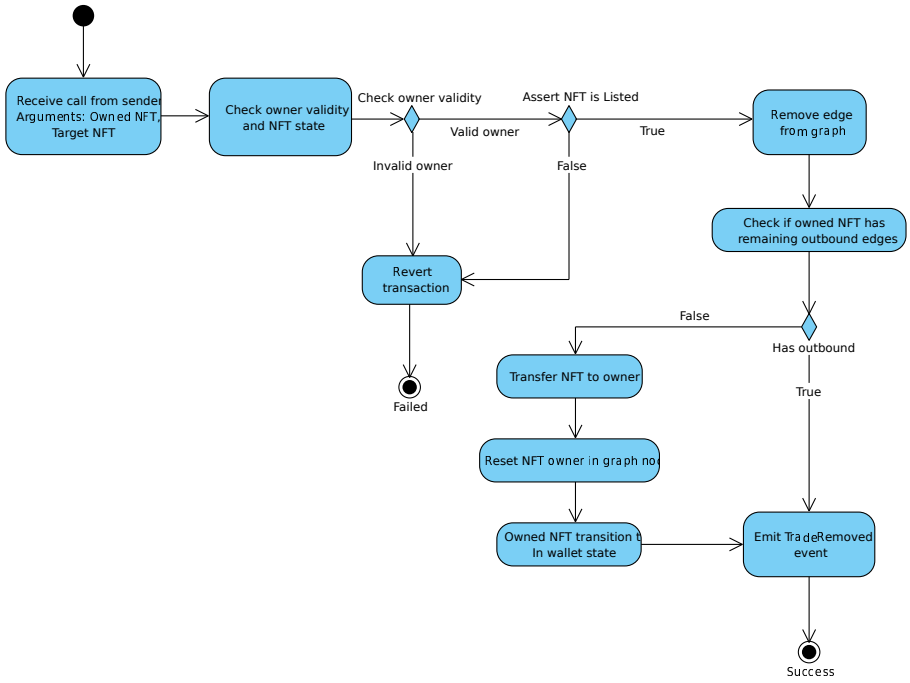
Η προσθήκη πρόθεσης ανταλλαγής προσθέτει μια ακμή στο γράφο.



Σχήμα 4.3: Διάγραμμα για την προσθήκη ανταλλαγής.

4.3.3 Μέθοδος Αφαίρεσης Ανταλλαγής

Η αφαίρεση πρόθεσης επαναφέρει την κατάσταση του NFT.

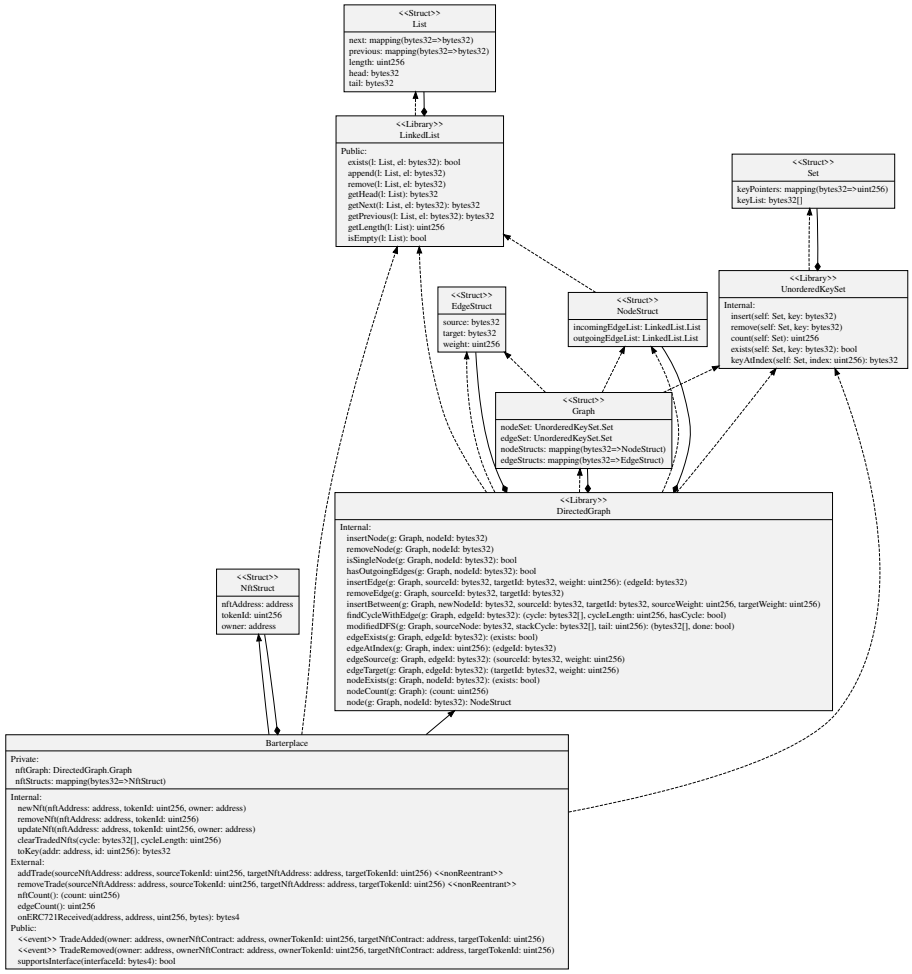


Σχήμα 4.4: Διάγραμμα για την αφαίρεση ανταλλαγής.

4.3.4 Ανίχνευση κύκλου

Ο αλγόριθμος DFS χρησιμοποιείται για τον εντοπισμό κύκλων ανταλλαγής. Όταν εντοπιστεί κύκλος, πραγματοποιείται ταυτόχρονη μεταβίβαση ιδιοκτησίας.

4.3.5 Διάγραμμα κλάσεων



Σχήμα 4.5: UML διάγραμμα κλάσεων smart contract.

4.4 Off-chain βάση δεδομένων

Απαιτείται μια εξωτερική βάση δεδομένων για indexing και queries για την κατάσταση του γράφου, τα οποία δεν είναι εφικτό να εκτελούνται on-chain.

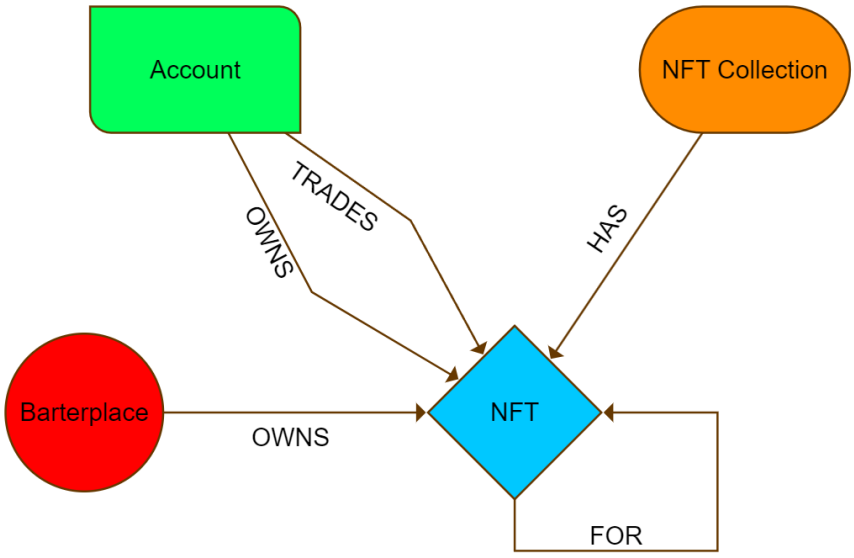
4.4.1 Graph Βάση Δεδομένων

Μοντελοποίηση των NFTs και των ανταλλαγών ως γράφος. Τα βασικά πλεονεκτήματα περιλαμβάνουν:

- Ταχύτατα queries για υποστήριξη front-end
- Ευέλικτο και προσαρμόσιμο σχήμα βάσης

- Υψηλή απόδοση στην αναπαράσταση και επεξεργασία σχεσιακών δεδομένων

4.4.2 Σχήμα



Σχήμα 4.6: Γραφική αναπαράσταση βάσης δεδομένων NFTs.

4.4.3 API

REST endpoints για εγγραφή, ανάκτηση και διαχείριση δεδομένων NFT και ανταλλαγών:

1. **Λίστα όλων των collections**:
   GET /collections
   Επιστρέφει λίστα με όλα τα NFT collections.

   Παράμετροι:
   limit - int (προαιρετική παράμετρος): Μέγιστος αριθμός αποτελεσμάτων
   cursor - string (προαιρετική παράμετρος): Χρησιμοποιείται για pagination

2. **Λίστα όλων των NFTs ανά collection**:
   GET /collections/{contract_address}
   Επιστρέφει λίστα με όλα τα NFTs που ανήκουν σε συγκεκριμένη collection.

   Παράμετροι:
   contract_address - string: Διεύθυνση του collection contract
   include_metadata - boolean (προαιρετική παράμετρος): Επιστροφή μεταδεδομένων. Προεπιλογή: *false*
   limit - int
   cursor - string

3. **Λίστα όλων των NFTs που ανήκουν σε account**:
   GET /nfts/account/{account_address}
   Επιστρέφει λίστα με NFTs που ανήκουν σε συγκεκριμένο account.

   Παράμετροι:
   account_address - string: Διεύθυνση wallet
   include_metadata - boolean (προαιρετική)
   limit - int
   cursor - string

4. **Ιστορικό μεταβιβάσεων NFT**:
   GET /nfts/{contract_address}/{token_id}/history
   Επιστρέφει το ιστορικό μεταβιβάσεων ενός συγκεκριμένου NFT.

   Παράμετροι:
   contract_address - string
   token_id - string
   limit - int
   cursor - string

5. **Ενεργές ανταλλαγές NFT στο Barterplace**:
   GET /nfts/{contract_address}/{token_id}/trades
   Επιστρέφει τις διαθέσιμες εξερχόμενες αχμές για το συγκεκριμένο NFT.

   Παράμετροι:
   contract_address - string
   token_id - string
   include_metadata - boolean
   limit - int
   cursor - string

6. **Ανάκτηση metadata NFT**:
   GET /nfts/{contract_address}/{token_id}/metadata
   Επιστρέφει τα on-chain metadata ενός συγκεκριμένου NFT.

   Παράμετροι:
   contract_address - string
   token_id - string

7. **Ανάκτηση ιδιοκτήτη NFT**:
   GET /nfts/{contract_address}/{token_id}/owner
   Επιστρέφει τον ιδιοκτήτη ενός συγκεκριμένου NFT.

   Παράμετροι:
   contract_address - string
   token_id - string

4.4.4 NFT Discovery Service (Ανακάλυψη NFTs)

Υπηρεσία σε Node.js που συγχρονίζει τη βάση με τα δεδομένα στο blockchain.
Ροή λειτουργίας

Βήμα 1: Εκκίνηση της υπηρεσίας

Αρχικοποίηση της υπηρεσίας NFT discovery.

Βήμα 2: Φόρτωση ρυθμίσεων

Η υπηρεσία φορτώνει τις απαραίτητες ρυθμίσεις, όπως παραμέτρους δικτύου, διεύθυνση Barterplace και API keys.

Βήμα 3: Σύνδεση με το Blockchain

Η υπηρεσία δημιουργεί σύνδεση με το Blockchain για να λαμβάνει εισερχόμενα blocks.

Βήμα 4: Ανίχνευση νέου block

Το Blockchain δημοσιεύει νέο block που περιέχει συναλλαγές σχετικές με NFTs.

Βήμα 5: Ανάλυση και φιλτράρισμα δεδομένων του block

Η υπηρεσία NFT discovery εξάγει τις σχετικές συναλλαγές και φιλτράρει γεγονότα των smart contracts.

Βήμα 6: Επεξεργασία γεγονότων smart contract

Το σύστημα παρακολουθεί γεγονότα όπως:

- `TradeAdded` (ένα NFT trade ξεκίνησε)
- `TradeRemoved` (μια ανταλλαγή ακυρώθηκε)

Βήμα 7: Ενημέρωση της βάσης δεδομένων

Τα εξαγόμενα δεδομένα χρησιμοποιούνται για την ενημέρωση της off-chain βάσης δεδομένων, διατηρώντας τα αρχεία NFTs επικαιροποιημένα.

Βήμα 8: Έλεγχος συγχρονισμού

Αν η βάση δεδομένων είναι μη ενημερωμένη, η υπηρεσία ενεργοποιεί ανάκτηση ιστορικών δεδομένων για την ανανέωση των παρελθοντικών συναλλαγών.

Βήμα 9: Κατάσταση ετοιμότητας

Όταν ολοκληρωθούν όλα τα προηγούμενα βήματα, η υπηρεσία εισέρχεται σε ενεργή κατάσταση ακρόασης, επεξεργαζόμενη συνεχώς νέα blocks.

UML διαγράμματα ακολουθίας

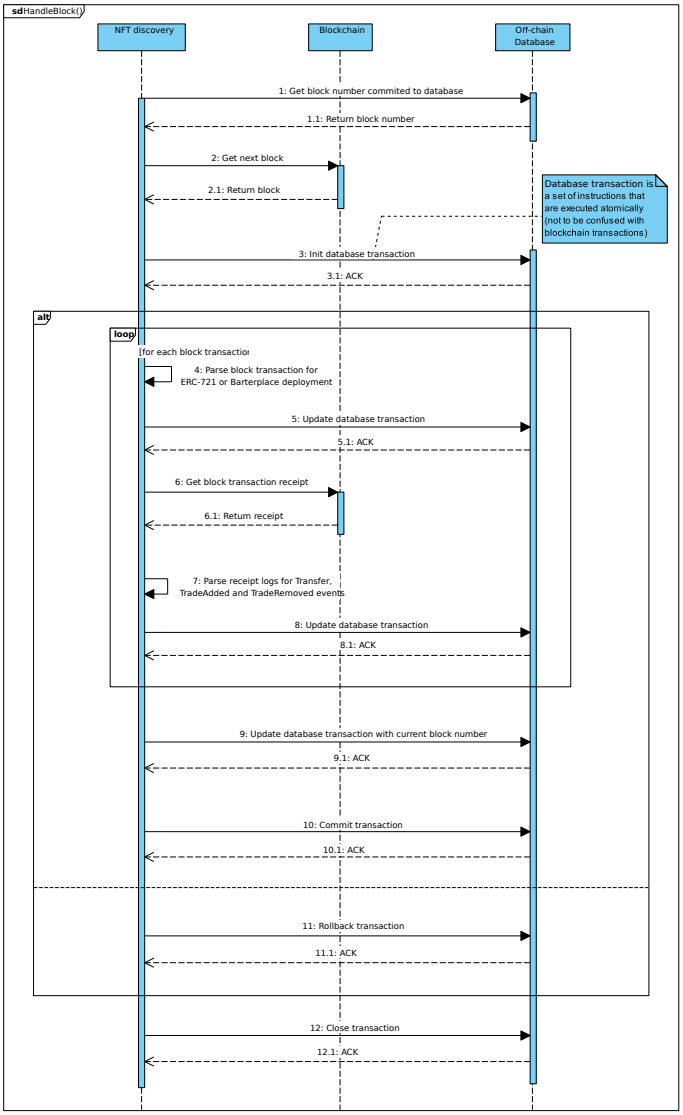


Figure 4.7: Ακολουθία λειτουργίας HandleBlock.

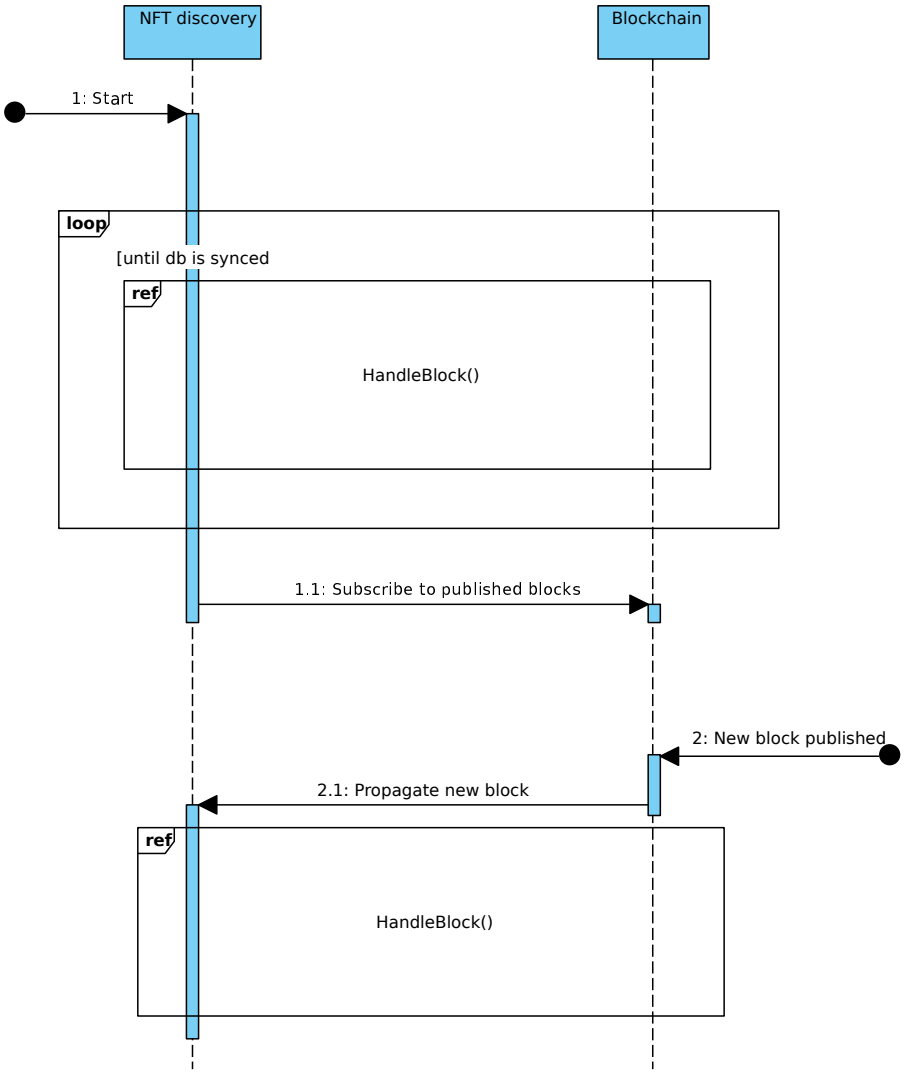


Figure 4.8: Ακολουθία αρχικοποίησης υπηρεσίας NFT Discovery.

Κεφάλαιο 5

Συζήτηση και αξιολόγηση

Η υλοποίηση του συστήματος ανταλλαγών NFTs μέσω του Barterplace ανέδειξε τόσο τις δυνατότητες όσο και τους περιορισμούς της προσέγγισης. Παρότι το σύστημα λειτουργεί ως απόδειξη σκοπιμότητας, απαιτείται βελτιστοποίηση ώστε να επιτύχει κλιμακούμενη απόδοση σε συνθήκες πραγματικού κόσμου. Σε αυτό το κεφάλαιο αναλύονται το κόστος gas, οι περιορισμοί του EVM και προτάσεις όπως off-chain υπολογισμούς.

5.1 Ανάλυση κόστους gas

Δοκιμές με 1.000 χρήστες που κατείχαν από 5 NFTs και εκτελούσαν τυχαίες ανταλλαγές μεταξύ τους αποκάλυψαν σημαντικές πληροφορίες για την απόδοση του έξυπνου σομβολαίου.

Κόστος Gas ανά μέθοδο				
Μέθοδος	**Ελάχιστο**	**Μέγιστο**	**Μέσος όρος**	**# Κλήσεις**
addTrade	375199	4762760	565738	5000
removeTrade	127983	137983	85434	52

Πίνακας 5.1: Ανάλυση κόστους gas κατά τη χρήση

Κόστος deployment συμβολαίων		
Συμβόλαιο	**Μέσος όρος**	**% του block limit**
Barterplace	2,931,923	9.8%
LinkedList	669,509	2.2%
SimpleCollection	2,535,147	8.5%

Πίνακας 5.2: Κατανάλωση gas για το deployment συμβολαίων

Το Ethereum block limit είναι 30 εκατομμύρια gas. Οι παραπάνω μετρήσεις δείχνουν ότι οι βασικές ενέργειες παραμένουν εντός ορίων, αλλά η λειτουργία `addTrade`(προσθήκη ανταλλαγής) είναι ιδιαίτερα απαιτητική.

5.2 Στρατηγικές βελτιστοποίησης

Για την καλύτερη απόδοση, μπορούν να εφαρμοστούν τεχνικές όπως:

- Μείωση writes σε storage και χρήση `calldata` αντί `memory`.
- Χρήση mappings αντί για arrays.
- Μαζική εκτέλεση(batching) συναλλαγών.
- Χρήση proxy συμβολαίων για μελλοντικές αναβαθμίσεις.

Οι παραπάνω πρακτικές μπορούν να βοηθήσουν στη μείωση της κατανάλωσης gas και να ενισχύσουν την επεκτασιμότητα.

5.2.1 Off-chain υπολογισμός

Η μεταφορά απαιτητικών υπολογισμών εκτός blockchain είναι μια αποτελεσματική στρατηγική. Ο υπολογισμός διαδρομών ανταλλαγών μπορεί να γίνεται εκτός αλυσίδας, με την τελική εκτέλεση να καταγράφεται on-chain και οι ανταλλαγές να πραγματοποιούνται από το έξυπνο συμβόλαιο.

Ένα έξυπνο συμβόλαιο με μηχανισμό staking χρησιμοποιείται για την εξασφάλιση ορθότητας:

- Οι χρήστες καταθέτουν εγγύηση.
- Εάν ο κύκλος είναι έγκυρος, η εκτέλεση γίνεται.
- Αν υποβληθεί λανθασμένη συναλλαγή, μπορεί να αποδειχθεί με fraud proof.
- Το συμβόλαιο καταστρέφεται και η εγγύηση αποδίδεται στον καταγγέλλοντα.

Αυτό μειώνει το κόστος gas και αποτρέπει κακόβουλη συμπεριφορά.

Κεφάλαιο 6

Συμπεράσματα

Η παρούσα διπλωματική εργασία παρουσίασε το Barterplace, ένα αποκεντρωμένο σύστημα ανταλλαγής NFTs που βασίζεται σε τεχνολογία blockchain και θεωρία γράφων. Αντί να στηρίζεται στην παραδοσιακή αγοραπωλησία, το Barterplace εισάγει ένα μοντέλο πολυμερούς ανταλλαγής μέσω κατευθυνόμενων γράφων.

Κάθε πρόθεση ανταλλαγής αναπαρίσταται ως ακμή σε γράφο, και η ανίχνευση κύκλων μέσω του αλγορίθμου DFS επιτρέπει την αυτόματη και δίκαιη εκτέλεση συναλλαγών. Η υλοποίηση έγινε με smart contracts στην πλατφόρμα Ethereum, διασφαλίζοντας διαφάνεια και ασφάλεια, ενώ το IPFS αξιοποιήθηκε για την αποκεντρωμένη αποθήκευση μεταδεδομένων των NFTs.

Κύρια οφέλη της προσέγγισης

1. **Ανταλλαγές χωρίς εμπιστοσύνη (trustless):** Η ανάγκη για μεσάζοντες εξαλείφεται, μειώνοντας τα κόστη και αυξάνοντας την ασφάλεια.
2. **Βελτιωμένη ρευστότητα:** Οι πολυμερείς ανταλλαγές αυξάνουν τις πιθανότητες επιτυχούς μεταβίβασης.
3. **Αυτόματη και δίκαιη εκτέλεση:** Οι έ μά (smart contracts) εφαρμόζουν τους κανόνες χωρίς ανθρώπινη παρέμβαση.

Κατευθύνσεις για μελλοντική έρευνα

Παρόλο που η υλοποίηση επιβεβαίωσε τη σκοπιμότητα του συστήματος, παραμένουν προκλήσεις, κυρίως όσον αφορά την επεκτασιμότητα. Το υψηλό κόστος gas για σύνθετες συναλλαγές περιορίζει τη χρήση του σε μεγάλης κλίμακας περιβάλλοντα.

Πιθανές βελτιώσεις περιλαμβάνουν:

- Βελτιστοποίηση της λογικής του smart contract
- Αξιοποίηση off-chain υπολογισμού για trade paths

Η υιοθέτηση αυτών των τεχνικών μπορεί να οδηγήσει σε μια πιο αποδοτική, κλιμακούμενη και ρεαλιστική πλατφόρμα ανταλλαγής ψηφιακών περιουσιακών στοιχείων στο πλαίσιο του Web3.