JKSCI

한국컴퓨터정보학회논문지
Journal of The Korea Society of Computer and Information
Vol. 29 No. 2, pp. 127-136, February 2024
https://doi.org/10.9708/jksci.2024.29.02.127

Hierarchical NFT using Parent-Child Structure

JongWook Bae*, Nitin Bhagat*, Su-Hyun Lee*

*Postdoctoral Researcher, Dept. of Computer Engineering, Changwon National University, Changwon, Korea
*Student, Dept. of Computer Engineering, Changwon National University, Changwon, Korea
*Professor, Dept. of Computer Engineering, Changwon National University, Changwon, Korea

[Abstract]

This paper presents a novel method for minting hierarchical Non-Fungible Tokens(NFTs) via a parent-child structure. In contrast to existing NFT structures, our proposed model enables an NFT to act as a parent, creating child NFTs and distributing ownership stakes among them. These child NFTs are recursively structured, allowing them to generate their own descendants. The existing structure of NFTs does not inherently allow for fractional ownership. However, our proposed hierarchical model provides a feasible solution to this restriction. By dividing an NFT into multiple child NFTs, each with its own unique identity, we facilitate the detailed division of an asset, thereby making fractional ownership possible. In conclusion, the hierarchical NFT model proposed in this paper offers a promising solution to the challenges of fractional ownership in the digital asset arena. By enabling the detailed division of NFTs through a parent-child structure, we anticipate a future where digital assets can be owned and traded more flexibly and transparently.

▸ **Key words**: Hierarchical Non-Fungible Tokens, Parent-Child Structure, Klaytn, multiple child NFTs

[요　약]

본 논문은 부모-자식 구조를 통해 계층적 NFT(Non-Fungible Token)를 발행하는 새로운 방법을 제시한다. 기존 NFT 구조와 달리 우리가 제안한 모델은 NFT가 부모 NFT 역할을 하여 자식 NFT를 생성하고 이들 사이에 소유권 지분을 분배할 수 있도록 한다. 자식 NFT는 재귀적으로 구조화되어 자체 자손을 생성할 수 있다. NFT의 기존 구조는 본질적으로 부분 소유권을 허용하지 않는다. 그러나 우리가 제안한 계층적 모델은 이러한 제한에 대한 실행 가능한 솔루션을 제공한다. NFT를 각각 고유한 ID를 가진 여러 자식 NFT로 분할함으로써 자산의 분할을 용이하게 하여 부분 소유권을 가능하게 한다. 결론적으로, 본 논문에서 제안된 계층적 NFT 모델은 디지털 자산 분야에서 부분 소유권 문제에 대한 솔루션을 제공한다. 부모-자식 구조를 통해 NFT의 세부적인 분할을 가능하게 함으로써 디지털 자산을 보다 유연하고 투명하게 소유하고 거래할 수 있는 미래를 기대한다.

▸ **주제어**: 계층형 NFT, 부모-자식 구조, 클래이튼, 다중 자식 NFT

• First Author: JongWook Bae, Corresponding Author: Su-Hyun Lee
*JongWook Bae (bae@cwnu.ac.kr), Dept. of Computer Engineering, Changwon National University
*Nitin Bhagat (bhagatnitin312@gmail.com), Dept. of Computer Engineering, Changwon National University
*Su-Hyun Lee (sleepl@changwon.ac.kr), Dept. of Computer Engineering, Changwon National University
• Received: 2024. 01. 29, Revised: 2024. 02. 14, Accepted: 2024. 02. 19.

I. Introduction

The introduction of Non-Fungible Tokens (NFTs) has revolutionized the way we perceive digital assets, providing a unique identity to each asset on the blockchain. While NFTs have been largely utilized in the domain of digital art, their potential use-cases extend far beyond. This paper explores the innovative concept of hierarchical NFT generation, a novel approach that facilitates fractional ownership and trading of assets[1-2].

Fractional ownership, enabled by dividing an asset into multiple unique entities, provides an opportunity for multiple individuals to hold a stake in a single high-value asset. This concept has been prevalent in traditional sectors like real estate and business, and now, the advent of NFTs brings this concept into the digital realm. However, the current structure of NFTs does not inherently support fractional ownership[3]. Herein lies the significance of hierarchical NFT generation.

Hierarchical NFTs involve structuring NFTs in a hierarchical manner, where each NFT can be divided into multiple sub-NFTs, each with its own unique identity[4]. We implement a hierarchical structure of NFT by storing child and parent IDs and dividing ownership. This structure allows for the granular division of an asset, enabling fractional ownership. Moreover, each fraction can be independently traded, bringing in a new dimension to the NFT marketplace.

This paper, explores the methodology of creating hierarchical NFTs and the mechanisms for fractional ownership and trading. It further explores the potential benefits, challenges, and implications of this concept in various sectors. Chapter 2 describes the theoretical background, explains the hierarchical NFT method proposed in Chapter 3 and 4, and concludes in Chapter 5.

II. Theoretical Background

Non-Fungible Tokens (NFTs) represent a novel application of blockchain technology, embodying unique digital assets with immutable and verifiable ownership. Unlike fungible tokens like Bitcoin or Ethereum, each NFT is distinct and carries unique information and value. They are built on Ethereum's ERC-721 and ERC-1155 standards, which enable the creation and exchange of these unique tokens on the blockchain. ERC-721 is a standard for Non-Fungible Tokens, while ERC-1155 accommodates both Fungible and Non-Fungible Tokens[5]. NFTs have been increasingly recognized in various digital platforms and sectors, including digital art, gaming, and virtual real estate, due to their capacity to provide digital scarcity, provenance, and ownership of unique digital assets[6].

The choice to utilize the Klaytn Improvement Proposal-17 (KIP-17) standard as the foundational framework for our research is grounded in its distinct advantages over other blockchain protocols. KIP-17, developed by Ground X, a subsidiary of South Korea's internet Kakao, offers a more scalable, efficient, and user-friendly platform for the creation and transaction of NFTs. Unlike other standards, such as Ethereum's ERC-721 and ERC-1155, KIP-17 addresses several challenges, notably high gas fees and network congestion, thereby offering a more viable platform for broader NFT adoption. Our study, therefore, seeks to explore the uncharted territories of NFTs, leveraging the unique capabilities of KIP-17[7].

Many NFTs can be difficult to purchase due to their high cost, which could lead to lower liquidity of NFTs. Furthermore, when a single high-value asset is represented as an NFT, the risk associated with the fluctuation of that asset's value increases. In order to address these issues, a method of fractional ownership of NFTs is considered. By dividing the NFT into smaller units, it is possible to invest at a lower price. This approach will enhance the liquidity of NFTs and diversify investment risk[8-9].

However, this method also presents challenges. With the fractionalization of NFT ownership, the management of each fraction's ownership rights can become more complex. Furthermore, accurately measuring the value of each fraction and conducting trades based on this can be difficult. To address these issues, the concept of hierarchical NFTs was introduced. Hierarchical NFTs operate by creating multiple layers of NFTs, with each layer's NFT interconnected with others. This allows for the clear definition and management of each NFT's ownership rights and value, effectively addressing the issues of trading and liquidity associated with fractionalized NFTs.

Gebreab and colleagues proposed an NFT-based solution for managing refurbished medical devices and addressing concerns about quality and safety. Their approach leverages dynamically configurable NFTs in a parent-child hierarchy to track modifications stages, and non-transferable NFTs to ensure trustworthiness. The proposed solution has limitations as it relies on soulbound tokens to authenticate modification activities, which can be susceptible to fraudulent activities[10]. Noh and Shin proposed a method of managing copyright works in a virtual space using hierarchical NFTs. In this method, the virtual performance is organized as the Root Node, and each copyright work is set up as a Leaf Node, allowing for the management of copyrighted works through a hierarchically structured NFT all at once. However, this method utilizes the interface provided by ERC-6150 to maintain the hierarchical relationship[11].

The hierarchical NFT we propose begins with the creation of an NFT for a basic asset. This NFT then performs the role of a parent, creating child NFTs and distributing its ownership stakes to them. These child NFTs, in turn, possess a recursive structure that allows them to generate their own offspring. Using this hierarchical NFT structure, digital asset ownership and liquidity can be granularized. It also allows for effective management of complex ownership structures. This contributes to clearly defining the value of digital assets, transparently managing ownership, and maximizing the efficiency of blockchain technology.

III. Desing of the Proposed Scheme

1. Block diagram of our platform

The overarching architecture of our system is shown in Fig. 1. Users can upload their digital content using the platform's dedicated upload interface and provide relevant details about the content[12].

Upon receiving the user's input, our server transfers the content to the InterPlanetary File System(IPFS) and creates a metadata file containing the Content Identifier(CID) and user-provided details. The metadata file is also uploaded to IPFS, generating a hash address that becomes the Uniform Resource Identifier(URI) of the NFT, providing a unique digital footprint for each asset. This token URI is used by our smart contract to generate the NFT.

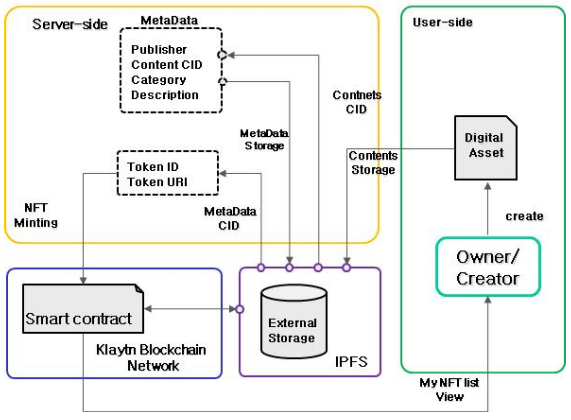


Fig. 1. Block diagram of NFT

2. Parent-child structure

Our platform is based on a smart contract named 'CWNFT'. This contract inherits from KIP17 and uses Counters to track token IDs and store tokenURIs. It includes a generator function for minting KIP17 tokens and methods for querying and managing NFTs providing a complete framework for creating, tracking, and managing NFTs.

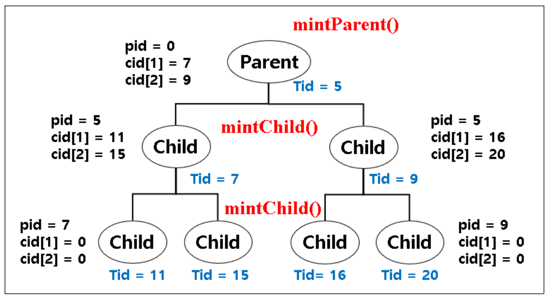


Fig. 2. Content and metadata upload result

Our platform has an innovative feature called the 'Heiric' structure, which implement a hierarchical structure in NFTs. In this structure, 'pid' represents the parent ID, mapping to the token ID of the current NFT's parent, while 'cid' stores the token IDs of the children NFTs of the current NFT. Additionally, 'share' signifies the ownership proportion associated with the relevant NFT. This component is essential for establishing and maintaining hierarchical relationships among NFTs within our platform.

```
contract CWNFT is KIP17Enumerable {
    Counters.Counter private _tokenIds;
    mapping(uint =>string) public tokenURIs ;
    mapping(uint256 => uint256) public nftPrices;
    uint256 [] public onSaleNfts;

    struct Heiric {
        uint256 pid;
        uint256[] cid;
        uint8 share ;
    }
    Heiric[] heiric ;
    constructor() KIP17("CWNU_NFT", "CWNU"){    }
    function tokenURI(uint _tokenId) {    }
    function mintParent(string memory _tokenURI) {    }
    function mintChild(uint parentId, uint8 _share) {    }
}
```

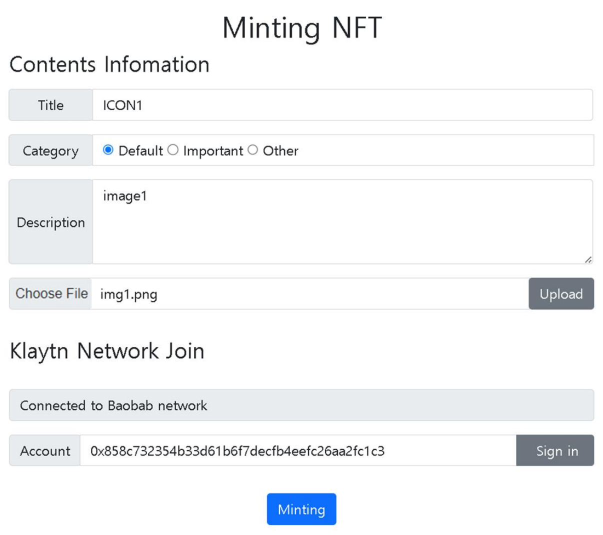
The minting of NFTs, both parent and child, is carried out using the 'mintParent()' and 'mintChild()' functions. When a parent NFT is minted using the 'mintParent()' function, it is assigned 100 percent ownership share. When minting child NFTs, the 'mintChild()' function is used, taking the parent ID and the ownership share as arguments.

IV. Implementation of the Proposed Scheme

1. Minting a parent NFT

As illustrated in <Fig. 3>, our platform's minting page simplifies the NFT creation process for users. The user-friendly interface enhances the NFT creation experience, making it more accessible and easy to use.

Users initiate the process by selecting their digital content using the [Choose File] button and then upload it to IPFS through Infura by clicking the [Upload] button. This can only be done if the user has registered the project ID and secret key for IPFS in their INFURA account[13].


Minting NFT

Contents Infomation

Title	ICON1
Category	[✓ Default] ○ Important ○ Other
Description	image1
Choose File	img1.png

Klaytn Network Join

Connected to Baobab network

Account	0x858c732354b33d61b6f7decfb4eefc26aa2fc1c3

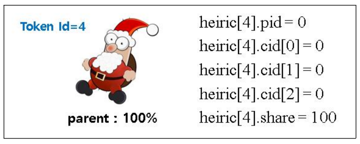
Fig. 3. Content and metadata upload result

After the user upload their data, they can log in to their Kaikas wallet and click on the [Minting] button. This action triggers the smart contract within our system. The 'mintParent()' function combines the unique token ID and the TokenURI to mint an NFT. This function also increments the token ID counter, assigns the tokenURI to the corresponding token ID, and mints the NFT using the _mint() function.

```
function mintParent(string memory _tokenURI)
                                payable public returns(uint256) {
    _tokenIds.increment();
    uint256 tokenId = _tokenIds.current();
    tokenURIs[tokenId] = _tokenURI;
    _mint(msg.sender, tokenId);

    uint256[] memory cid = new uint256[](3);
    Heiric memory room = Heiric(0, cid, 100);
    heiric.push(room);
    return id;
}
```

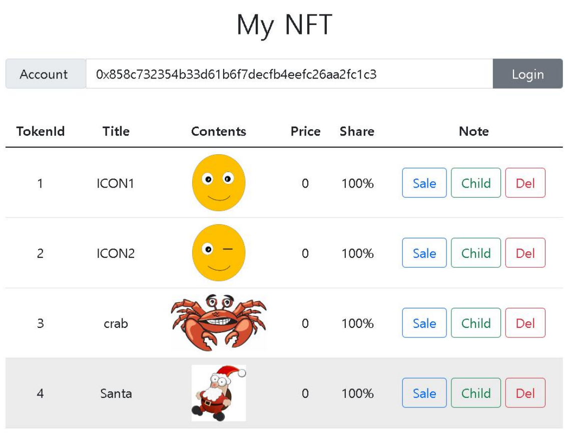
To manage NFTs hierarchically, the 'Heiric' structure is introduced. When a parent NFT is initially minted by the ‘mintParent()’ function, it does not have a parent or child. Therefore, 'pid' is assigned a value of zero, and 'cid' is kept as an initialized array. Since a parent NFT, at the point of creation, holds the entire ownership share, 'share' is consequently set at 100 percent. Here, the number of 'cid' elements is set to 3 to limit the number of children to 2, and to store the number of children in element 0. <Fig. 4> shows an example of the attribute values in the ‘hieric’ structure of token ID 4.



heiric[4].pid = 0
heiric[4].cid[0] = 0
heiric[4].cid[1] = 0
heiric[4].cid[2] = 0
heiric[4].share = 100

Fig. 4. Example of ‘heiric’ structure attribute values

After the NFT minting process is complete, users can view and manage their minted NFTs on the "My NFT" page as shown in <Fig. 5>. This feature allows users to monitor and control their NFTs within our system.



My NFT

Account 0x858c732354b33d61b6f7decfb4eefc26aa2fc1c3 Login

TokenId	Title	Contents	Price	Share	Note
1	ICON1		0	100%	Sale Child Del
2	ICON2		0	100%	Sale Child Del
3	crab		0	100%	Sale Child Del
4	Santa		0	100%	Sale Child Del

Fig. 5. Listed of created NFTs

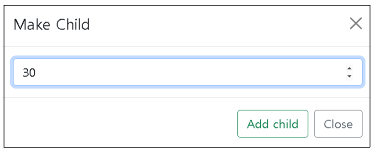
The ‘getNFTs()’ function retrieves a list of NFTs owned by a specific user. It fetches the ID, token URI, price, and ownership share for each token owned by the user's address and stores them in the 'NFTs' structure. Here, 'ownership share' accesses the 'hieric' structure with the token ID and retrieves the value of the internal variable 'share'.

```
function getNFTs(address _owner) view virtual
                                public returns (NFTs[] memory) {
    uint256 total = balanceOf(_owner);
    NFTs[] memory nfts = new NFTs[](total);

    for(uint256 i = 0; i < total; i++) {
        uint256 id = tokenOfOwnerByIndex(_owner, i);
        string memory uri = tokenURI(id);
        nfts[i] = NFTs(id, uri, nftPrices[id], heiric[id].share);
    }
    return nfts;
}
```

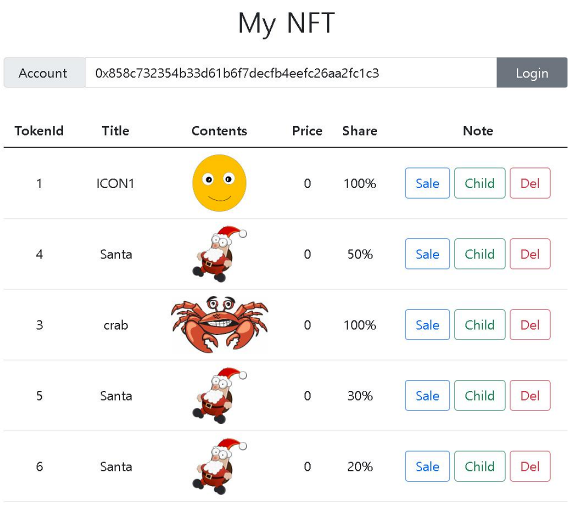
2. Sharing child from a parent NFT

When you select an NFT on the "My NFT" page and click the [Child] button, a pop-up window as shown in <Fig. 6> will appear. Here, you can input the desired ownership share. Clicking the [Add child] button will create a child NFT, allowing for the division of ownership shares.


Make Child

30

Fig. 6. Popup for division of ownership share


My NFT

Account 0x858c732354b33d61b6f7decfb4eefc26aa2fc1c3 Login

TokenId	Title	Contents	Price	Share	Note
1	ICON1		0	100%	Sale Child Del
4	Santa		0	50%	Sale Child Del
3	crab		0	100%	Sale Child Del
5	Santa		0	30%	Sale Child Del
6	Santa		0	20%	Sale Child Del

Fig. 7. Example of child minting on My NFT page

<Fig. 7> shows an example of minting two child NFT in Santa NFT and dividing the ownership shares into 30% and 20%, respectively. Token IDs 4, 5, and 6 are all Santa NFTs, with ownership stakes of 50%, 30%, and 20%, respectively.

```
function mintChild(uint parentId, uint8 _share)
                        payable public returns (uint256) {
    require(ownerOf(parentId) == msg.sender,
                        "Caller is not nft token owner.");
    tokenIds.increment();
    uint256 childId = tokenIds.current();
    tokenURIs[childId] = tokenURI(parentId);
    _mint(msg.sender, childId);

    uint256 i = ++heiric[parentId].cid[0] ;
    heiric[parentId].cid[i] = childId ;
    heiric[parentId].share -= _share;

    uint256[] memory cid = new uint256[](3);
    Heiric memory room = Heiric(parentId, cid, _share);
    heiric.push(room);
    return childId;
}
```

This is achieved using the ‘mintChild()’, which increments the 'tokenIds' counter and assigns its current value to 'childId'. This 'childId' is then associated with the same token URI as the parent token. The new child token is minted with the ‘_mint()’ function, assigning ownership to the message sender.

The next steps involves updating the 'heiric' structure to add the new child token ID to the 'cid' array and deducting the '_share' value from the parent token's owership share. Here, 'cid[0]' represents the number of child NFTs, which increases each time a new child token is minted, allowing for easy tracking of the total number of child NFTs associated with a specific parent NFT.

In conclusion, the function creates a new instance of the 'Heiric' struct with the parent token's ID, a new 'cid' array, and the '_share' value. This instance is then appended to the 'heiric' array to store the hierarchy. The function finally returns the 'childId', signifying the successful minting of the child token.

3. Hierarchical structure of NFT

<Fig. 8> illustrates the hierarchical structure of the NFT created in the example given in Fig. 5, and exemplifies the values stored in the attributes of the 'heiric' structure. The parent, Santa, was minted fourth, therefore having a token ID of 4 and is stored in the fourth index of the 'heiric' array. Two children were minted, generating token IDs 5 and 6, correspondingly creating the 5th and 6th indices in the 'heiric' array. Furthermore, the values 5 and 6 are stored in 'cid[1]' and 'cid[2]' of 'heiric[4]', which represent Santa's children. Since the children were assigned ownership shares of 30% and 20% respectively, Santa retains only 50% of the ownership share. The 'pid' of 'heiric[5]' and 'heiric[6]' stores 4, which is the token ID of the parent, Santa, and the ownership shares stored are 30% and 20% respectively. Finally, as no children have been generated yet, the 'cid' array holds a value of 0.

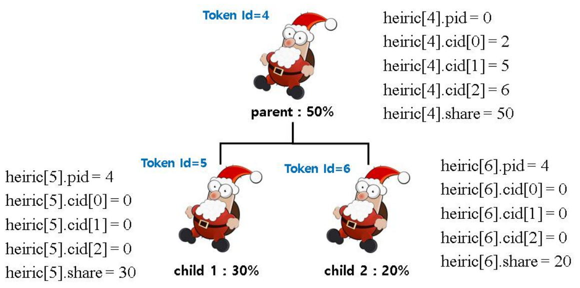


Fig. 8. Example of NFT hierarchy and the ‘hieric’ structure values

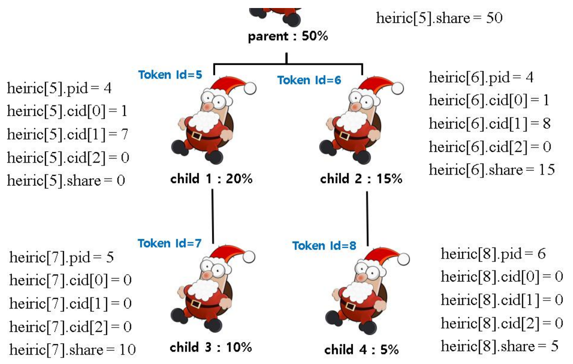


Fig. 9. Changes in hierarchy and structure values

If token IDs 5 and 6 mint child and share ownership shares of 10% and 5%, respectively, the hierarchy and attribute values will change as shown in <Fig. 9>.

4. Register for sale

When you click the [Sale] button, the "Register for Sale" form will appear as shown in <Fig. 10>. In the dialog box that follows, enter the amount you want to sell. Click the [Register] button to execute the subsequent code.

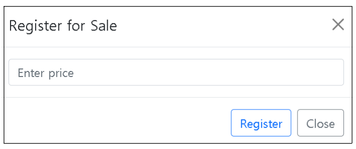
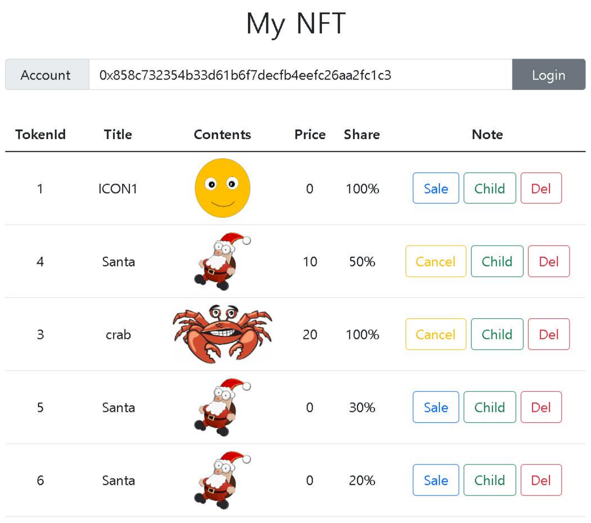


Fig. 10. Sale registration pop-up

The function 'setSaleNFT()' takes two parameters: '_id' representing the token ID of the NFT, and '_price' representing the intended selling price. It initiates by identifying the owner of the NFT using the 'ownerOf()' function. Then, it validates several conditions: 1) the caller of the function (msg.sender) must be the owner of the NFT, 2) the NFT must not already be listed for sale, and 3) the owner must have authorized the sale of the token. If all conditions are met, the NFT is listed for sale with the price set to '_price', and its token ID is added to the 'saleNfts' array. This function ensures the legitimate and secure listing of NFTs for sale.

```
function setSaleNft(uint256 _id, uint256 _price) public {
    address nftOwner = ownerOf(_id);
    require(nftOwner== msg.sender, "not nft token owner");
    require(nftPrices[_id]==0, "nft is already on sale.");
    require(isApprovedForAll(nftOwner, address(this)),
                            "nft owner did not approve token.");
    nftPrices[_id] = _price;
    saleNfts.push(_id);
}
```

After entering the desired sale amount shown in <Fig. 10> and completing the sale registration, the sale amount is displayed. The [Sale] button then transforms into the [Cancel] button, as shown in <Fig. 11>, allowing the user to cancel the sale registration if they choose to do so at any point.


My NFT

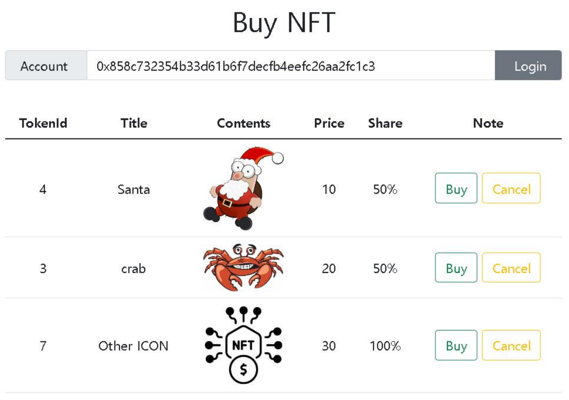
Account 0x858c732354b33d61b6f7decfb4eefc26aa2fc1c3 Login

TokenId	Title	Contents	Price	Share	Note
1	ICON1		0	100%	Sale Child Del
4	Santa		10	50%	Cancel Child Del
3	crab		20	100%	Cancel Child Del
5	Santa		0	30%	Sale Child Del
6	Santa		0	20%	Sale Child Del

Fig. 11. Sales registration completed

5. Buy NFT

Subsequently, the "Buy NFT" interface, as shown in <Fig. 12>, displays NFTs available for purchase by their owners. Users can browse and buy their preferred NFTs through this interface.



Buy NFT

Account | 0x858c732354b33d61b6f7decfb4eefc26aa2fc1c3 | Login

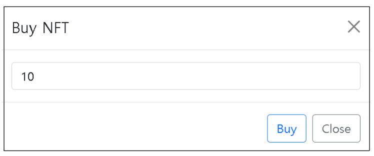
TokenId	Title	Contents	Price	Share	Note
4	Santa		10	50%	Buy Cancel
3	crab		20	50%	Buy Cancel
7	Other ICON		30	100%	Buy Cancel

Fig. 12. Buy NFT Page

The 'getSaleNfts()' function retrieves details of all NFTs currently listed for sale. It iterates over each element in the 'SaleNft' array, fetching the corresponding NFT's details such as token URI, price, and the share of the 'heiric'. These details are stored as an NFT object in the 'onSaleNfts' array. The function provides a detailed snapshot of all NFTs available for purchase at that moment.

```
function getSaleNfts() public view returns(Nfts[] memory) {
    Nfts[] memory onSaleNfts = new Nfts[](SaleNfts.length);
    for(uint i = 0; i < onSaleNft.length; i ++){
        uint id = SaleNft[i];
        onSaleNfts[i] = Nfts(id, tokenURI(id),
                             nftPrices[id],  heiric[id].share );
    }
    return onSaleNfts;
}
```

When the user clicks the [Buy] button on the purchase page a popup for purchasing the NFT will appear, as shown in <Fig. 13>.



Buy NFT

10

Buy Close

Fig. 13. Purchase pop-up

```
function buyNft(uint256 _id) public payable {
    address owner = ownerOf(_id);
    require(owner != msg.sender, "caller is nft owner.");
    require(isApprovedForAll(owner, address(this)),
              "nft owner did not approve token.");

    payable(owner).transfer(msg.value);
    KIP17(address(this)).safeTransferFrom(
                          nftOwner, msg.sender, _id);
    removeSaleNft(_id);
}
```

The 'buyNft()' function accepts the TokenId of the NFT as an argument. It first identifies the current owner of the NFT using the 'ownerOf' function. Then, it checks if the function caller is not the current owner of the NFT and if the token has been approved for sale by the owner. If these conditions are met, the function transfers the payment from the buyer to the current NFT owner. The NFT is then transferred from the owner to the buyer using the 'safeTransferFrom()' function of the KIP17 protocol. Finally, the 'removeSaleNft()' function is called to remove the NFT from the sale list, completing the purchase process.

The 'removeSaleNft()' function removes a specific NFT by its Token id. It first sets the price of the NFT to 0 to mark it for removal. Then, it iterates over the 'SaleNfts' array, which contains the unique identifiers of all NFTs currently on sale. When it encounters an NFT with a price of zero, it replaces that element with the last element in the array and then removes the last element, effectively eliminating the NFT from the sale list.

```
function removeSaleNft(uint256 _id) private {
    nftPrices[_id] = 0;
    for(uint256 i = 0; i<SaleNfts.length; i ++){
        if (nftPrices[SaleNfts[i]] == 0){
            SaleNfts[i] = SaleNfts[SaleNfts.length -1] ;
            SaleNfts.pop();
        }
    }
}
```

When you click the [Cancel] button in <Fig. 11>, the 'removeSaleNft()' function is called to remove the NFT from the sale list.

6. Deletion of NFT

```
function burn(uint256 _id) public {
    require(ownerOf(_id) == msg.sender,
              "msg.sender is not the owner of the token");
    require(heiric[_id].cid[0] == 0, "You have children");

    uint256 pid = heiric[_id].pid;
    if (pid != 0){
        heiric[pid].share += heiric[_id].share ;
        heiric[pid].cid[_id] = 0 ;
        heiric[pid].cid[0]-- ;
    }
    else heiric[_id].share = 0 ;
    _burn(_id);
    removeSaleNft(_id);
}
```

When you click the [Del] button in <Fig. 11>, the 'burn()' function is called. The 'burn()' function is a public function that allows the owner of a NFT to permanently remove the NFT from existence. It first checks if the caller of the function is the owner of the token and if the token has any child. If the token has a parent (pid != 0), the function redistributes the owership share of the token to be burned to its parent and updates the parent's child. If the token does not have a parent, it simply sets the owership share of the token to 0. After these steps, it calls the '_burn()' function to destroy the token and the 'removeSaleNft()' function to remove the token ID from the sale list.

V. Conclusions

In this paper, we proposed a new method for creating hierarchical Non-Fungible Tokens(NFTs) using a parent-child structure. This innovative model allows an NFT to act as a parent, creating child NFTs and distributing ownership stakes to them. The child NFTs have a recursive structure that allows them to generate their own offspring. This was successfully implemented by storing parent and child IDs in the 'hieric' structure and dividing ownership shares during the NFT minting process.

The current NFT structure does not inherently support fractional ownership, but our proposed hierarchical model presents a possible solution to this limitation. By dividing an NFT into multiple child NFTs, each with its own unique identity, we enable the granular division of an asset, facilitating fractional ownership.

Our hierarchical NFT model significantly contributes to transparent and efficient valuation of digital assets. It allows for clear delineation and management of ownership, maximizing the efficiency of blockchain technology.

In conclusion, the hierarchical NFT model we propose offers a promising solution to the challenges of fractional ownership in the digital asset space. By enabling the granular division of NFTs through a parent-child structure, we anticipate a future where digital assets can be owned and traded more flexibly and transparently.

The hierarchical NFTs we propose have a more complex structure than traditional NFTs, leading to increased costs due to the necessity of storing additional ownership share data. Additionally, as they are not based on a standardized protocol for the hierarchization of NFTs, difficulties are encountered in ensuring compatibility across various platforms. In the future, we aim to create a standardized protocol specifically for hierarchical NFTs.

ACKNOWLEDGEMENT

This research was funded by convergence research financial program for instructors, graduate students and professors in 2023.

REFERENCES

[1] L. Kugler, "Non-fungible tokens and the future of art," Communications of the ACM, Vol. 64, No. 9, pp. 19-20, Sep. 2021. DOI: 10.1145/3474355

[2] K. Shah, U. Khokhariya, and S. Patel, "Smart contract-based dynamic non-fungible tokens generation system", Apr. 2023. DOI: 10.21203/rs.3.rs-2796956/v1

[3] A. J. Abualhamayl, M. A. Almalki, and F. Al-Doghman, etc., "Towards Fractional NFTs for Joint Ownership and Provenance in Real Estate," 2023 IEEE International Conference on e-Business Engineering, pp. 143-148, Sydney, 20233 DOI: 10.1109/ICEBE59045.2023.00022

[4] H. R. Hasan, M. Madine, I. Yaqoob, K. Salah, R. Jayaraman, and D. Boscovic, "Using NFTs for ownership management of digital twins and for proof of delivery of their physical assets," Future Generation Computer Systems, Vol. 146, pp. 1-17, Sep. 2023. DOI: 10.1016/j.future.2023.03.047

[5] O. Marin, T. Cioara, L. Toderean, D. Mitrea, and I. Anghel, "Review of Blockchain Tokens Creation and Valuation," Future Internet, Vol. 15, No. 382, Nov. 2023. DOI: 10.3390/fi15120382

[6] LimeChain, ERC-721 vs ERC-1155, https://limechain.tech/blog/erc-721-vs-erc-1155-ethereum-token-standards/

[7] Klaytn API service, KIP-17 API, https://www.klaytnapi.com/en/resource/openapi/kip17/reference/overview/

[8] Q. Wang, R. Li, Q. Wang, and S. Chen, "Non-fungible token (nft): overview, evaluation, opportunities and challenges," ArXiv preprint arXiv:2105.07447, May 2021. DOI: 10.48550/arxiv.2105.07447

[9] E. Sung, O. Kwon, and K. Sohn, "Nft luxury brand marketing in the metaverse: leveraging blockchain-certified nfts to drive consumer behavior," Psychology & Marketing, Vol. 40, No. 11, pp. 2306-2325, Jun. 2023. DOI: 10.1002/mar.21854

[10] S. A. Gebreab, K. Salah, R. Jayaraman and J. Zemerly, "Trusted Traceability and Certification of Refurbished Medical Devices Using Dynamic Composable NFTs," IEEE Access, Vol. 11, pp. 30373-30389, Mar. 2023, DOI: 10.1109/ACCESS.2023.3261555

[11] C. H. Roh 1 and D. M. Shin, "A Study on the Technology of Managing Joint Copyrights in Hierarchical NFT-Based Large-Scale Virtual Performances," Journal of Software Assessment and Valuation, Vol. 19, No. 4, pp. 11-21, Dec. 2023 DOI: 10.29056/jsav.2023.12.02

[12] N. Bhagat, J. W. Bae, and S. H. Lee, "A user friendly NFT platform for Digital Assets," Proceedings of KSCI Conference 2023, Vol. 31, No. 2, pp. 447-450, 2023

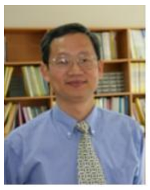
[13] INFURA, INFURA IPFS, https://www.infura.io/product/ipfs/

Authors

JongWook Bae received the B.S. in Computer Science from Changwon University, Korea in 2001, followed by his Master's and Doctoral degrees in the same department in 2005 and 2012, respectively.

He currently is a lecturer in the Department of Computer Science at Changwon National University. He is interested in image processing and blockchain.

Nitin Bhagat received the B.E and M.E degrees in Computer Engineering from Pokhara University, Nepal, in 2004 and 2017, respectively. He joined the faculty of Computer Engineering as a Lecturer at Purbanchal University, Biratnagar, Nepal, in 2009. He is currently a Research scholar in the blockchain technology at Changwon National University in Korea. He is interested in blockchain technology, NFT web platform design and evaluation.



Su-Hyun Lee received the B.S. in Computer Science from Kwangwoon University, Korea in 1987. He received the M.S. and Ph.D. degrees in Computer Science from Korea Advanced Institute of Science and Technology(KAIST), Korea, in 1989, 1994, respectively. Dr. Lee is a Professor in the Department of Computer Engineering, Changwon National University since 1996. He is interested in computer algorithm, programming languages, compiler, and blockchain.