MCFT: Multi-Class Fungible Token

Albert Chon
Department of Computer Science
Stanford University
achon@stanford.edu

July 2018

Abstract

We describe a new token standard that enables the creation of multiple classes of fungible tokens (MCFTs) within a single smart contract. This standard provides the basic functionality to create, track and transfer ownership of MCFTs. We describe a new model of token issuance wherein an initial coin offering (ICO) can be modeled as a creation of a new class within the MCFT contract. We then describe a novel method of decentralized exchange that is enabled by the MCFT standard and describe the potential application of MCFTs in the music and energy industries.

Contents

1 Introduction

1.1 Existing Token Standards

1.1.1 ERC-20

On November 19th 2015, the twentieth Ethereum Request for Comment (ERC) was proposed which defined a new smart contract standard that allowed for the creation of a single class of fungible tokens. Since its creation, the ERC-20 standard has become the industry standard for token issuance, as billions of dollars worth of Ethereum have been raised through new tokens created through this standard (e.g. Filecoin, EOS, TRX, etc). Each token created through this standard has the capability to define its own unique functionality suited to its own use case (e.g. payments for file transfer, computation, etc) but are united in that they all conform to a standard API which allow for developers to accurately predict interaction between tokens within the larger ecosystem. The standard ERC-20 interface is shown below.

```
contract ERC20Interface {
    function totalSupply() public constant returns (uint);
    function balanceOf(address tokenOwner) public constant returns (uint balance);
    function allowance(address tokenOwner, address spender) public constant returns (uint remaining);
    function transfer(address to, uint tokens) public returns (bool success);
    function approve(address spender, uint tokens) public returns (bool success);
    function transferFrom(address from, address to, uint tokens) public returns (bool success);

    event Transfer(address indexed from, address indexed to, uint tokens);
    event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
}
```

The above framework has been used by countless blockchain projects for issuing tokens.

1.1.2 ERC-223

On March 5th 2017, the ERC-223 standard was proposed to resolve the issue of ERC-20 tokens being lost forever when they are sent to a contract address that had not anticipated receiving the tokens. Millions of dollars worth of ERC-20 tokens had been lost since ERC-20 contracts were unable to recognize incoming transactions with a different token, and hence were unable to support usage of such tokens. In an ERC-223 contract, however, the contract will reject transactions with tokens it cannot recognize, thus preventing such losses.

The ERC-223 standard is a superset of the ERC-20 standard and adds the following functions in its interface.

```
    function transfer(address _to, uint _value, bytes _data);
    function tokenFallback(address _from, uint _value, bytes _data);
    function transfer(address to, uint value, bytes data);

    event Transfer(address indexed from, address indexed to, uint value, bytes data);
```

1.1.3 ERC-721

On September 20th 2017, the ERC-721 standard was proposed, defining a new smart contract standard to allow for the creation and issuance of non-fungible tokens. Unlike fungible tokens which are interchangeable amongst each other, non-fungible tokens are unique and can create verifiable digital scarcity. Despite how recently this standard was introduced, numerous games such as CryptoKitties, CryptoFighters and CryptoCup have used ERC-721 tokens to represent virtual collectibles whose value is derived from its scarcity. Applying NFTs to the real world is now currently an active area of research, as scarce, digital tokens can be used to represent physical property such as houses and artwork on the blockchain. Like the ERC-20 token standard, all ERC-721 contracts conform to the same required interface shown below.

```
contract ERC721Interface {
    function balanceOf(address _owner) external view returns (uint256);
    function ownerOf(uint256 _tokenId) external view returns (address);
    function safeTransferFrom(address _from, address _to, uint256 _tokenId, bytes data) external payable;
    function safeTransferFrom(address _from, address _to, uint256 _tokenId) external payable;
    function transferFrom(address _from, address _to, uint256 _tokenId) external payable;
    function approve(address _approved, uint256 _tokenId) external payable;
    function setApprovalForAll(address _operator, bool _approved) external;
    function getApproved(uint256 _tokenId) external view returns (address);
    function isApprovedForAll(address _owner, address _operator) external view returns (bool);

    event Transfer(address indexed _from, address indexed _to, uint256 indexed _tokenId);
    event Approval(address indexed _owner, address indexed _approved, uint256 indexed _tokenId);
    event ApprovalForAll(address indexed _owner, address indexed _operator, bool _approved);
}
```

1.1.4 ERC-998

On April 15th 2018, the ERC-998 standard was proposed to allow a non-fungible token to own another non-fungible ERC-721 or standard fungible ERC-20 tokens. Hence, fungible and nonfungible tokens could be composed as a single token which could then be bought, sold and transferred. Taking tokens from CryptoKitties as an example, one could create a token composition of a cryptokitty which own a scratching post and a feeding dish which could (in theory) be contain fungible “chow” tokens. The ERC-998 token standard is still under active review by the Ethereum community which has not agreed on a standard interface as of the time of writing.

1.2 Use Cases and Implications

From just the ERC-20 and ERC-721 standards alone, we have seen that billions of dollars worth of economic value being created. Hence, there are huge implications associated with creating a new, widely used token standard. As this is the case, providing new functionality through a new token standard can open up numerous possibilities and novel types of digital value creation, representation and transfer in our economy.

2 Multi-Class Fungible Token (MCFT) Standard

2.1 EIP 1178

2.1.1 High-Level Explanation

Imagine several ERC-20 tokens that are related to each other in some way. As an ERC-20 token, one would, of course, be able to query certain properties about each token, perform actions such as transfers, and define custom properties/functionalities about the relation of one token to another. Now, instead of having separate ERC-20 tokens interacting with each other, consider a single smart contract that encapsulates this concept through the creation of several classes of tokens within the same contract. This is the basis of the ERC-1178 standard which allows for greater intraoperability and more complex token interactions through creating several tokens on the same smart contract.

2.1.2 Motivation

Currently, there is no standard to support tokens that have multiple classes. In the real world, there are many situations in which defining distinct classes of the same token would be fitting (e.g. distinguishing between preferred, common, and restricted shares of a company). Yet, such nuance cannot be supported in today's token standards. An ERC-20 token contract defines tokens that are all of one class while an ERC-721 token contract creates a class (defined by token id) for each individual token. The ERC-1178 token standard proposes a new standard for creating multiple classes of tokens within one token contract.

2.1.3 Specification

The following is the Solidity code that corresponds to the proposed MCFT token standard.

```
contract ERC1178 {
    // Required Functions
    function implementsERC1178() public pure returns (bool);
    function totalSupply() public view returns (uint256);
    function individualSupply(uint256 classId) public view returns (uint256);
    function balanceOf(address owner, uint256 classId) public view returns (uint256);
    function classesOwned(address owner) public view returns (uint256[]);
    function transfer(address to, uint256 classId, uint256 quantity) public;
    function approve(address to, uint256 classId, uint256 quantity) public;
    function transferFrom(address from, address to, uint256 classId) public;

    // Optional Functions
    function name() public pure returns (string);
    function className(uint256 classId) public view returns (string);
    function symbol() public pure returns (string);

    // Required Events
    event Transfer(address indexed from, address indexed to, uint256 indexed classId, uint256 quantity);
    event Approval(address indexed owner, address indexed approved, uint256 indexed classId, uint256 quantity);
}
```

Notice there are functions above that are not part of pre-existing token standards. The following should delineate each function and variables used in more detail. The full example smart contract Solidity code is included in the appendix.

```
  function individualSupply(uint256 classId) public view returns (uint256) {
    return classIdToSupply[classId];
  }

  function balanceOf(address owner, uint256 classId) public view returns (uint256) {
    return ownerToClassToBalance[owner][classId];
  }

  // class of 0 is meaningless and should be ignored.
  function classesOwned(address owner) public view returns (uint256[]){
    uint256[] memory tempClasses = new uint256[](currentClass - 1);
    uint256 count = 0;
    for (uint256 i = 1; i < currentClass; i++){
      if (ownerToClassToBalance[owner][i] != 0){
        if (ownerToClassToBalance[owner][i] != 0){
          tempClasses[count] = i;
        }
        count += 1;
      }
    }
    uint256[] memory classes = new uint256[](count);
```

```
  for (i = 0; i < count; i++){
    classes[i] = tempClasses[i];
  }
  return classes;
}

function transfer(address to, uint256 classId, uint256 quantity) public {
  require(ownerToClassToBalance[msg.sender][classId] >= quantity);
  ownerToClassToBalance[msg.sender][classId] -= quantity;
  ownerToClassToBalance[to][classId] += quantity;
  Transactor memory zeroApproval;
  zeroApproval = Transactor(0x0, 0);
  approvals[msg.sender][classId] = zeroApproval;
}

function approve(address to, uint256 classId, uint256 quantity) public {
  require(ownerToClassToBalance[msg.sender][classId] >= quantity);
  Transactor memory takerApproval;
  takerApproval = Transactor(to, quantity);
  approvals[msg.sender][classId] = takerApproval;
  emit Approval(msg.sender, to, classId, quantity);
}

function approveForToken(uint256 classIdHeld, uint256 quantityHeld,
  uint256 classIdWanted, uint256 quantityWanted) public {
  require(ownerToClassToBalance[msg.sender][classIdHeld] >= quantityHeld);
  TokenExchangeRate memory tokenExchangeApproval;
  tokenExchangeApproval = TokenExchangeRate(quantityHeld, quantityWanted);
  exchangeRates[msg.sender][classIdHeld][classIdWanted] = tokenExchangeApproval;
}
```

2.2 Advantages

The three benefits of the ERC-1178 token standard are greater intraoperability, security, and efficiency.

2.2.1 Intraoperability

Rather than interoperability (which in the case of the Ethereum blockchain typically refers to the the ability to exchange and make use of tokens across multiple smart contracts), we emphasize the greater intraoperability afforded by this new token standard. By essentially allowing multiple tokens to be under the same smart contract, developers as well as users can enjoy greater simplicity and ease of use while interacting with MCFTs. Because the data associated with each token class resides in the same contract, developers can much more easily leverage direct access to internal data structures to build more sophisticated applications than what was possible before simply with ERC-20 contracts.

2.2.2 Security

An additional key advantage with MCFTs is that because the tokens are defined in a single contract, external function calls do not need to be made to interact with multiple tokens, thus significantly reducing the potential attack vector surface.

2.2.3 Efficiency

MCFTs provide superior efficiency in two fronts. First, along with enhanced security, less gas is consumed from not having to make external function calls. Second, less gas is required for the creation of the contract and of the tokens themselves. As a matter of fact, the original inspiration for the creation of the ERC-1178 standard was that the costs of trying to model multi-class tokens using different classes of fungible ERC 721 tokens (an oxymoron) was far too great. Using the maximum gas amount one can send with a transaction on MetaMask, we were only able to create around 46 ERC 721 tokens in our contract (with its corresponding internal data structures maintaining the state of each token class) before exhausting all gas. This experience motivated the creation of the MCFT token standard to lower the gas fees for users and developers, thus making participation in a smart contract-enabled ecosystem more efficient and democratic.

3 Primary Use Case: Artíste

3.1 Problem

There are currently many inefficiencies in the creative industry. New artists especially face many challenges in not only producing their art (music, painting, film, dance, etc) but also earning money and gaining a following. It can be especially challenging for artists to gain recognition and convince others to invest in their talent and future.

3.2 Solution

We propose Artíste - a crowdfunding platform and tokenized ecosystem that allows artists to distribute their art, raise funds, build a following, and gain publicity and support from their fans.

3.2.1 Crowdfunding

Artíste is built on the ERC-1178 standard and allows artists to raise money by issuing their personal class of Artíste tokens to their supporters.

3.2.2 Tokenized Ecosystem

In turn, these tokens can then be redeemed for goods and services (e.g. concert tickets, song mentions, autographs, etc.) from the artist on the Artíste platform. The artist can then sell the tokens they receive, thus incentivizing the artist to participate and exchange value on the platform. Because there is a finite supply of tokens for each artist and since fans can sell their artist tokens to each other, as the artist grows more popular the tokens appreciate in value. Since as the artist grows in popularity, the value of the goods and services provided by the artist grow in value, supporters are rewarded for holding their artist tokens and are incentivized to invest in the artist's future.

3.2.3 Decentralized Exchange

Artíste also provides an decentralized exchange which allows users to trade their artist tokens of one class for another. To do this, a user first posts a bid defining the exchange rate that they would like to exchange their token for. The user does not need to know apriori who they want to exchange with; they simply need to know which the class and quantity of token they would like to post and obtain in exchange. Then other users can view these bids on the Artíste marketplace and choose to fulfill the bids and complete the exchange using any quantity that satisfies the exchange rate. We are currently working on extending this implementation to support ERC-20 and ERC-223 tokens as well.

4 Future Works and Potential Applications

4.1 NRG Exchange

The situation is the following:

We are at the cusp of a radical transformation of our existing fossil fuel-based infrastructure to a renewable energy-based infrastructure. For example, Sweden and Denmark currently run on 50% solar, wind, and hydro power and

have committed to 100% renewable energy by 2040. This "new industrial revolution" will fundamentally transform the way we store, trade, and use energy, and will be supported by a new industry of machine-to-machine interactions and autonomous networks.

Currently, energy markets are inefficient and centralized where the largest energy companies have oligopoly pricing on the energy markets. There is no free market for individuals to sell their energy resources, so individuals who have surplus energy resources (e.g. from harvesting energy from solar panels) cannot sell them to other users - only back to their energy company at the rate they set. Not only is this system inefficient, it also introduces a single centralized point of failure where the individual is dependent on one centralized entity. Hence, we propose creating an efficient trustless marketplace for energy resources. Using MCFTs, multiple energy resources (e.g. wind, geothermal, solar, etc.) can tokenized as its own class under the ERC-1178 standard. With such tokenization, users can enjoy greater liquidity and interoperability as they can post their energy tokens in an open marketplace for energy resources.

5 Acknowledgements

We would like to thank Kevin Choi for the substantial amount of help he gave us in the ideation and initial creation of this whitepaper.

6 Disclaimer

6.1 Notes

This document is only used to introduce the concept of MCFTs to the global community, and any part thereof and any copy thereof must not be taken or transmitted to any country where distribution or dissemination of this document is prohibited or restricted.

This document is only used for the purpose of conveying information and does not constitute related opinions for buying or selling MCFT tokens, shares or securities. Any similar proposals or offers shall be performed under the trustworthy clauses, applicable securities laws and other relevant laws. The information or analysis above does not constitute an investment decision or specific proposal.

This document does not constitute any investment advice, investment intentions or solicitation of investment in the form of securities. This document

does not constitute and construe to provide any buying or selling behaviors, any buying or selling invitation, any forms of security behaviors. It is not any forms of contract or commitment.

No information in this document should be considered to be business, legal, financial or tax advice regarding MCFTs. Each potential participant should consult its own legal, financial, tax or other professional adviser.

References

[1] Nakamoto, Satoshi. *Bitcoin: A peer-to-peer electronic cash system.* 2008. https://bitcoin.org/bitcoin.pdf

[2] Szabo, Nick. *Smart contracts: building blocks for digital markets.* 1996. ht tp://www.alamut.com/subj/economics/nick_szabo/smartContracts.html

[3] https://github.com/ethereum/eips/issues/20

[4] https://github.com/ethereum/EIPs/issues/223

[5] https://github.com/ethereum/EIPs/issues/998

[6] https://github.com/achon22/EIPs/blob/master/EIPS/eip-1178.md

7 Appendix

```
contract ArtisteTokenContract is AccessControl, ERC1178 {
  using SafeMath for uint256;
  address public Owner;
  uint256 public tokenCount;
  uint256 currentClass;
  uint256 minTokenPrice;
  uint256 minCount;
  struct Transactor {
    address actor;
    uint256 amount;
  }
  struct TokenExchangeRate {
    uint256 heldAmount;
    uint256 takeAmount;
  }
  mapping(uint256 => uint256) public classIdToSupply;
  mapping(address => mapping(uint256 => uint256)) ownerToClassToBalance;
  mapping(address => mapping(uint256 => Transactor)) approvals;
  mapping(uint256 => string) public classNames;
  // owner's address to classIdHeld => classIdWanted => TokenExchangeRate
  mapping(address => mapping(uint256 => mapping(uint256 => TokenExchangeRate))) exchangeRates;

  // Constructor
  constructor () public {
    Owner = msg.sender;
    currentClass = 1;
    tokenCount = 0;
    minCount = 1000;
    minTokenPrice = 200000000000000;
  }

  function implementsERC1178() public pure returns (bool) {
    return true;
  }

  function totalSupply() public view returns (uint256) {
    return tokenCount;
  }

  function individualSupply(uint256 classId) public view returns (uint256) {
    return classIdToSupply[classId];
  }

  function balanceOf(address owner, uint256 classId) public view returns (uint256) {
    return ownerToClassToBalance[owner][classId];
  }

  // class of 0 is meaningless and should be ignored.
  function classesOwned(address owner) public view returns (uint256[]){
    uint256[] memory tempClasses = new uint256[](tokenCount);
    uint256 count = 0;
    for (uint256 i = 1; i < currentClass; i++){
      if (ownerToClassToBalance[owner][i] != 0){
        tempClasses[count] = ownerToClassToBalance[owner][i];
        count += 1;
      }
    }
    uint256[] memory classes = new uint256[](count);
```

```
  for (i = 0; i < count; i++){
    classes[i] = tempClasses[i];
  }
  return classes;
}

function transfer(address to, uint256 classId, uint256 quantity) public {
  require(ownerToClassToBalance[msg.sender][classId] >= quantity);
  ownerToClassToBalance[msg.sender][classId] -= quantity;
  ownerToClassToBalance[to][classId] += quantity;
  Transactor memory zeroApproval;
  zeroApproval = Transactor(0x0, 0);
  approvals[msg.sender][classId] = zeroApproval;
}

function approve(address to, uint256 classId, uint256 quantity) public {
  require(ownerToClassToBalance[msg.sender][classId] >= quantity);
  Transactor memory takerApproval;
  takerApproval = Transactor(to, quantity);
  approvals[msg.sender][classId] = takerApproval;
  emit Approval(msg.sender, to, classId, quantity);
}

function approveForToken(uint256 classIdHeld, uint256 quantityHeld,
  uint256 classIdWanted, uint256 quantityWanted) public {
  require(ownerToClassToBalance[msg.sender][classIdHeld] >= quantityHeld);
  TokenExchangeRate memory tokenExchangeApproval;
  tokenExchangeApproval = TokenExchangeRate(quantityHeld, quantityWanted);
  exchangeRates[msg.sender][classIdHeld][classIdWanted] = tokenExchangeApproval;
}

// A = msg.sender and B = to
// A wants to exchange his quantityHeld amount of token classIdHeld
// in exchange for person to's quantityWanted amount of tokens of classIdWanted
function exchange(address to, uint256 classIdPosted, uint256 quantityPosted,
  uint256 classIdWanted, uint256 quantityWanted) public {
    // check if capital existence requirements are met by both parties
    require(ownerToClassToBalance[msg.sender][classIdPosted] >= quantityPosted);
    require(ownerToClassToBalance[to][classIdWanted] >= quantityWanted);
    // check if approvals are met
    require(approvals[msg.sender][classIdPosted].actor == address(this) &&
      approvals[msg.sender][classIdPosted].amount >= quantityPosted);
    require(approvals[to][classIdWanted].actor == address(this) &&
      approvals[to][classIdWanted].amount >= quantityWanted);
    // check if exchange rate is acceptable
    TokenExchangeRate storage rate = exchangeRates[to][classIdWanted][classIdPosted];
    require(SafeMath.mul(rate.takeAmount, quantityWanted) < SafeMath.mul(rate.heldAmount, quantityPosted));
    // update balances
    ownerToClassToBalance[msg.sender][classIdPosted] -= quantityPosted;
    ownerToClassToBalance[to][classIdPosted] += quantityPosted;
    ownerToClassToBalance[msg.sender][classIdWanted] += quantityWanted;
    ownerToClassToBalance[to][classIdWanted] -= quantityWanted;
    // update approvals and
    approvals[msg.sender][classIdPosted].amount -= quantityPosted;
    approvals[to][classIdWanted].amount -= quantityWanted;
  }

function transferFrom(address from, address to, uint256 classId) public {
  Transactor storage takerApproval = approvals[from][classId];
  uint256 quantity = takerApproval.amount;
```

```
    require(takerApproval.actor == to && quantity >= ownerToClassToBalance[from][classId]);
    ownerToClassToBalance[from][classId] -= quantity;
    ownerToClassToBalance[to][classId] += quantity;
    Transactor memory zeroApproval;
    zeroApproval = Transactor(0x0, 0);
    approvals[from][classId] = zeroApproval;
  }

  function name() public pure returns (string) {
    return "Artiste Token";
  }

  function className(uint256 classId) public view returns (string){
    return classNames[classId];
  }

  function symbol() public pure returns (string) {
    return "ARTE";
  }

  // Artists call this function to create their own token offering
  function registerArtist(string artistName, uint256 count) public payable returns (bool){
    require(msg.value >= count * minTokenPrice && count >= minCount);
    ownerToClassToBalance[msg.sender][currentClass] = count;
    classNames[count] = artistName;
    currentClass += 1;
    return true;
  }
}
```