COLLECTIVE INVESTMENT SCHEME FUND ADMINISTRATION AND MANAGEMENT USING MODULAR BLOCKCHAINS AND TOKENIZATION

Authors: Steven Garner, Nigel Carter

Date: February 2023

Keywords: AI, Tokenization, Modular Blockchain, Accounting, Cap table, KYC, Reporting, Administration, Custody

Abstract

The automation of fund administration processes using ai, tokenization and modular blockchain technology for the purpose of reducing costs, eliminating reconciliations and providing immediate reporting and transparency for all stakeholders.

The current problem

Fund administration processes are complex, time-consuming, and prone to errors. They involve multiple parties, including administrators, investors, advisers, partners, directors, custodians, and administrators, each of whom has to perform various management tasks such as subscriptions, redemptions, cash movements, asset valuations, accounting, and NAV reporting. These processes are critical to the smooth functioning of a fund with timely complete and accurate account reporting, but they are also resource-intensive and can create a significant burden for all parties involved.

Managers of collective investment schemes or funds, however structured, usually delegate the operational processes to experts, but this can often create reconciliation and gap issues.

The proposed solutions

In recent years, blockchain technology has emerged as a potential solution for simplifying fund administration processes. Atomic settlement is a goal sought by many of the leading fund administration organizations. By using a modular blockchain fund administrators can streamline operations, reduce fund expenses, and increase

transparency. In this white paper, we will explore how fund administration processes can be operated on a modular blockchain, focusing on subscriptions, redemptions, cash movements, asset valuations, accounting, and NAV reporting.

A modular blockchain architecture separates the data and the processing elements. The protocol separates data, transaction processing and consensus processing functions into modules that can come from different parties and then links them all together.

A fund administration service could benefit from utilizing a modular blockchain and tokens to increase the speed and accuracy of cash movements, bank account reconciliations, subscriptions, redemptions, accounting, NAV reporting, cap table administration, customer onboarding with KYC and AML checks. The use of a blockchain and tokens can help streamline these processes by providing a secure, immutable ledger for all transactions that is accessible to all parties. Additionally, tokens can be used to automate certain processes and reduce manual labor associated with reconciliations and other tasks. By utilizing a secure token system and blockchain technology, fund administrators can also reduce costs associated with compliance and regulatory checks, as well as reduce the risk of fraud and errors. Furthermore, a modular blockchain and tokens can provide greater transparency into transactions, allowing fund administrators to quickly and accurately provide real-time NAV reports, cap table updates and customer onboarding. In sum, utilizing a modular blockchain and tokens can help fund administrators make their processes more timely, less error prone and more cost effective.

Modular Blockchain for Fund Administration Processes

A modular blockchain is a type of blockchain that allows for the creation of individual modules or smart contracts, each of which can perform specific functions. This modular approach to blockchain technology is ideal for fund administration processes, as it allows for the creation of smart contracts that can automate and streamline various tasks.

Subscriptions and Redemptions

Smart contracts can be used to automate the subscriptions and redemptions process. When an investor wants to subscribe to a fund, they can initiate the transaction through the blockchain, which will then trigger the smart contract to verify the investor's identity, perform KYC checks, and execute the transaction.

Similarly, when an investor wants to redeem their shares, the smart contract can perform the necessary checks and execute the transaction automatically. This process reduces the need for manual intervention, reducing the risk of errors and increasing efficiency.

Subscription Example

1. Receive the subscription payment: The administrator should receive the payment in the form of a wire transfer or check in US dollars (USD).

2. Record the transaction: The administrator should enter the subscription payment details into the ledger and update the books.

3. Convert the funds to the fund's base currency: The administrator should convert the USD payment to the fund's base currency, typically the Euro.

4. Record the converted funds: The administrator should record the converted funds in the ledger and update the books.

5. Reconcile the funds: The administrator should compare the amount of the subscription payment to the amount of the converted funds to make sure everything matches up.

6. Transfer the funds to the fund's bank account: The administrator should transfer the converted funds to the fund's bank account.

7. Notify the investor: The administrator should notify the investor that the subscription payment has been processed and the funds have been transferred.

Smart Contract Example

```
pragma solidity ^0.8.0;

contract Subscription {

    string public name;

    uint public subscriptionAmount;

    string public currency;

    string public consideration;

    uint public timestamp;

    constructor(string memory _name, uint _subscriptionAmount, string memory
_currency, string memory _consideration) {

        name = _name;

        subscriptionAmount = _subscriptionAmount;

        currency = _currency;

        consideration = _consideration;

        timestamp = block.timestamp;
```

```
    }
    function getDate() public view returns (uint) {
        return timestamp;
    }
    function getSubscriptionAmount() public view returns (uint) {
        return subscriptionAmount;
    }
    function getName() public view returns (string memory) {
        return name;
    }
    function getCurrency() public view returns (string memory) {
        return currency;
    }
    function getConsideration() public view returns (string memory) {
        return consideration;
    }
}
```

In this smart contract, the Subscription contract is created with the name, subscriptionAmount, currency, and consideration passed as parameters to the constructor. The timestamp is automatically set to the current block timestamp using block.timestamp.

The smart contract provides several functions to retrieve the stored data:

getDate(): returns the timestamp of the subscription

getSubscriptionAmount(): returns the subscription amount

getName(): returns the name of the subscriber

getCurrency(): returns the currency of the subscription amount

getConsideration(): returns the consideration for the subscription

When the contract is deployed, the data can be written to the blockchain by calling the constructor with the appropriate parameters. The data can be retrieved by calling the appropriate function.

Note that this is just a basic example, and depending on the specific requirements of the use case, you may need to add additional functionality or modify the existing code. Additionally, this smart contract assumes that the subscription amount is denominated in a single currency. If the use case requires support for multiple currencies, it may need to be modified. The cap-table smart contract could be

```
pragma solidity ^0.8.0;

contract CapTable {

    struct Shareholder {

        string name;

        uint shares;

        uint256 minimumTimePeriod;

    }

    struct Asset {

        string name;

        string assetType;

        uint quantity;

        uint nominalValue;

        string consideration;

        string currency;

    }

    mapping (address => Shareholder) public shareholders;

    Asset public asset;

    uint public totalShares;

    constructor(string memory _name, string memory _assetType, uint _quantity, uint
_nominalValue, string memory _consideration, string memory _currency, uint256
_minimumTimePeriod) {

        asset = Asset(_name, _assetType, _quantity, _nominalValue, _consideration,
_currency);
```

```
        totalShares = _quantity;
        shareholders[msg.sender] = Shareholder("Buyer", 100, _minimumTimePeriod);
    }
    function buyShares(uint _shares) public payable {
        require(msg.value == (asset.nominalValue * _shares), "Insufficient funds");
        require(_shares <= (totalShares - shareholders[msg.sender].shares), "Not enough
shares available");
        shareholders[msg.sender].shares += _shares;
        totalShares -= _shares;
    }
}
```

Redemption example

1. Receive the redemption request: The administrator should receive the redemption request in writing from the investor.

2. Calculate the redemption amount: The administrator should calculate the redemption amount, taking into account any applicable fees.

3. Record the redemption request: The administrator should enter the redemption request details into the ledger and update the books.

4. Convert the funds to the fund's base currency: The administrator should convert the redemption amount to the fund's base currency, typically the Euro.

5. Record the converted funds: The administrator should record the converted funds in the ledger and update the books.

6. Reconcile the funds: The administrator should compare the amount of the redemption request to the amount of the converted funds to make sure everything matches up.

7. Transfer the funds to the investor’s bank account: The administrator should transfer the converted funds to the investor’s bank account.

8. Notify the investor: The administrator should notify the investor that the redemption request has been processed and the funds have been transferred.

Cash Movements

Smart contracts can also be used to manage cash movements within the fund. When an investor subscribes to a fund, the smart contract can automatically trigger the transfer of funds from the investor's account to the fund's account. When an investor redeems their shares, the smart contract can transfer the necessary funds back to the investor's account.

Custody of assets

A Custodian works closely with an Administrator and is responsible for the safekeeping and accounting of the fund's assets, and for ensuring the fund is in compliance with applicable laws, regulations, and standards as in the case of European depositaries and AIFMS. The Custodian can also act as an insurance against breach or loss and provide a bankruptcy remote offering. They may also provide reporting services to investors and other stakeholders.

The actual process of custody for digital assets on a blockchain involves creating a wallet address on the blockchain and transferring the digital assets to that address. The asset is then secured by the blockchain’s distributed ledger and can be accessed only by those with the private key associated with the wallet address. The asset can then be further secured by setting up access rules, such as 2-factor authentication, and multisig wallets, which require multiple signatures to authorize a transaction.

Asset Valuations

Asset valuations are a critical component of fund administration processes. Traditionally, asset valuations are performed manually by fund administrators, which can be time-consuming and prone to errors. However, by using a modular blockchain, smart contracts can automate the asset valuation process and ensure an immutable and transparent proof of valuation.

For example, a smart contract can be created to automatically pull data from various sources, including exchanges, pricing feeds, and other data providers, to determine the current value of the fund's assets. This process can be performed on a regular basis, ensuring that the fund's asset valuations are always up-to-date and accurate.

Scenario valuations can also be created using ai and corpus data to provide stakeholders with immediate and transparent reporting time stamped as a proof of record on the blockchain.

Smart Contract Template Example

```
// SPDX-License-Identifier: MIT

pragma solidity ^0.8.0;
```

```
import "https://github.com/smartcontractkit/chainlink/blob/develop/
evm-contracts/src/v0.8/interfaces/AggregatorV3Interface.sol";

contract AssetValuation {


    struct Source {

        uint256 weight;

        AggregatorV3Interface aggregator;

    }

    mapping(string => Source[]) sourcesByType;

    function addSource(string memory _type, uint256 _weight, address _aggregator)
public {

        sourcesByType[_type].push(Source(_weight,
AggregatorV3Interface(_aggregator)));

    }

    function getAssetValue(string memory _type, uint256 _date, uint256 _factor, string
memory _json, uint256 _scenario) public view returns (uint256) {

        uint256 totalValue = 0;

        Source[] memory sources = sourcesByType[_type];

        for (uint i = 0; i < sources.length; i++) {

            (, int256 price, , uint256 timestamp, ) = sources[i].aggregator.latestRoundData();

            if (timestamp < _date) {

                continue;

            }

            totalValue += uint256(price) * sources[i].weight;

        }

        return totalValue * _factor * _scenario;

    }

}
```

Key parts of this contract:

AggregatorV3Interface: This is an interface provided by the Chainlink project that allows us to retrieve the latest price data from various data sources. We import this interface in our contract.

addSource: This function allows us to add a data source for a particular asset type. The weight parameter allows us to assign a weight to each source, which will be used to calculate the overall value of the asset.

getAssetValue: This is the main function that calculates the asset value based on the inputs provided. It first retrieves all the sources for the given asset type and filters out any sources that have a timestamp earlier than the input date. It then calculates the overall value of the asset by multiplying the prices of each source by its weight, and then multiplies the result by the valuation factor and the scenario.

Valuing a digital asset:

AssetValuation assetValuation = new AssetValuation();

assetValuation.addSource
("ETH", 50, 0x5f4eC3Df9cbd43714FE2740f5E3616155c5b8419); // Ethereum price feed

assetValuation.addSource
("BTC", 50, 0xF4030086522a5bEEa4988F8cA5B36dbC97BeE88c); // Bitcoin price feed

uint256 assetValue = assetValuation.getAssetValue
("ETH", 1645185600, 2, "{\"exchange\":\"Coinbase\",\"pair\":\"ETH/USD\"}", 1);

In this example, we create a new instance of the AssetValuation contract and add two sources for the asset type "ETH" and "BTC", both with a weight of 50. We then call the getAssetValue function with the following parameters:

type: "ETH"

date: 1645185600 (which represents February 18, 2022)

factor: 2

json: "{"exchange":"Coinbase","pair":"ETH/USD"}"

scenario: mid-market

Accounting

Smart contracts can also be used to automate the accounting process. When a transaction occurs within the fund, the smart contract can automatically update the fund's accounting records, including entries for subscriptions, redemptions, cash movements, and asset valuations. This process can reduce the need for manual

intervention, reducing the risk of errors and increasing efficiency. Using double entry accounting contract techniques, the need for reconciliations is no longer required. Statutory or regulatory reporting is immediate without need for preparation elsewhere and this will speed up any audits.

Smart Contract Example

Here is an example of a smart contract that creates a double entry onto a blockchain for an investor buying 5 membership points, crediting equity with USD 100 and crediting bank with USD 100:

```
pragma solidity ^0.8.0;

contract DoubleEntry {
    struct Entry {
        address account;
        int256 amount;
        string entryType;
    }
    Entry[] public entries;
    constructor() {
        entries.push(Entry(msg.sender, -100, "Equity"));
        entries.push(Entry(msg.sender, 100, "Bank"));
    }
    function buyMembershipPoints(int256 _amount) public {
        entries.push(Entry(msg.sender, _amount, "Equity"));
        entries.push(Entry(msg.sender, -_amount, "Bank"));
    }
}
```

In this smart contract, the DoubleEntry contract is created with two initial entries, one crediting equity with USD 100 and the other crediting bank with USD 100. These entries represent the initial investment from the investor.

The smart contract provides a function buyMembershipPoints() to allow the investor to purchase additional membership points. The function takes the amount to purchase as a parameter and creates two new entries to represent the transaction. The first entry

credits equity with the purchase amount and the second entry debits 'bank' with the same amount. This creates a double entry, ensuring that the accounting is always balanced.

The Entry struct contains information about the account (represented by an Ethereum address), the amount (represented as a signed integer), and the type of entry (either "Equity" or "Bank"). The entries array contains all the entries in the ledger. Note that this is just a basic example, and depending on the specific requirements of the use case, you may need to add additional functionality or modify the existing code. Additionally, this smart contract assumes that there is only one investor and that all entries are denominated in USD. If the use case requires support for multiple investors or multiple currencies, the contract will need to be refactored accordingly.

The production of templated account reports will be immediately available as all the accounting entries will be balanced and accorded the correct transaction time stamp and reporting periods. Consolidation will also be immediately available and accessible as with all the transactions by api into spreadsheets or other packages for further analysis such as stress testing.

Here is a template for a statement of financial position (balance sheet) for a fund, based on the International Financial Reporting Standards (IFRS) produced entirely from a modular blockchain produced Trial Balance:

[Name of Fund] Statement of Financial Position As of [Date] in Currency [Currency] in [Thousands, Millions etc]

Assets		Liabilities and Equity	
Current Assets		Current Liabilities	
Cash and Cash Equivalents	[Amount]	Accounts Payable	[Amount]
Marketable Securities	[Amount]	Accrued Expenses	[Amount]
Receivables	[Amount]	Short-Term Borrowings	[Amount]
Other Current Assets	[Amount]	Other Current Liabilities	[Amount]
Total Current Assets	[Amount]	Total Current Liabilities	[Amount]
Non-Current Assets		Non-Current Liabilities	
Property, Plant, and Equipment	[Amount]	Long-Term Borrowings	[Amount]
Intangible Assets	[Amount]	Other Non-Current Liabilities	[Amount]
Investments in Associates and Subsidiaries	[Amount]	Deferred Tax Liabilities	[Amount]
Other Non-Current Assets	[Amount]	Total Non-Current Liabilities	[Amount]

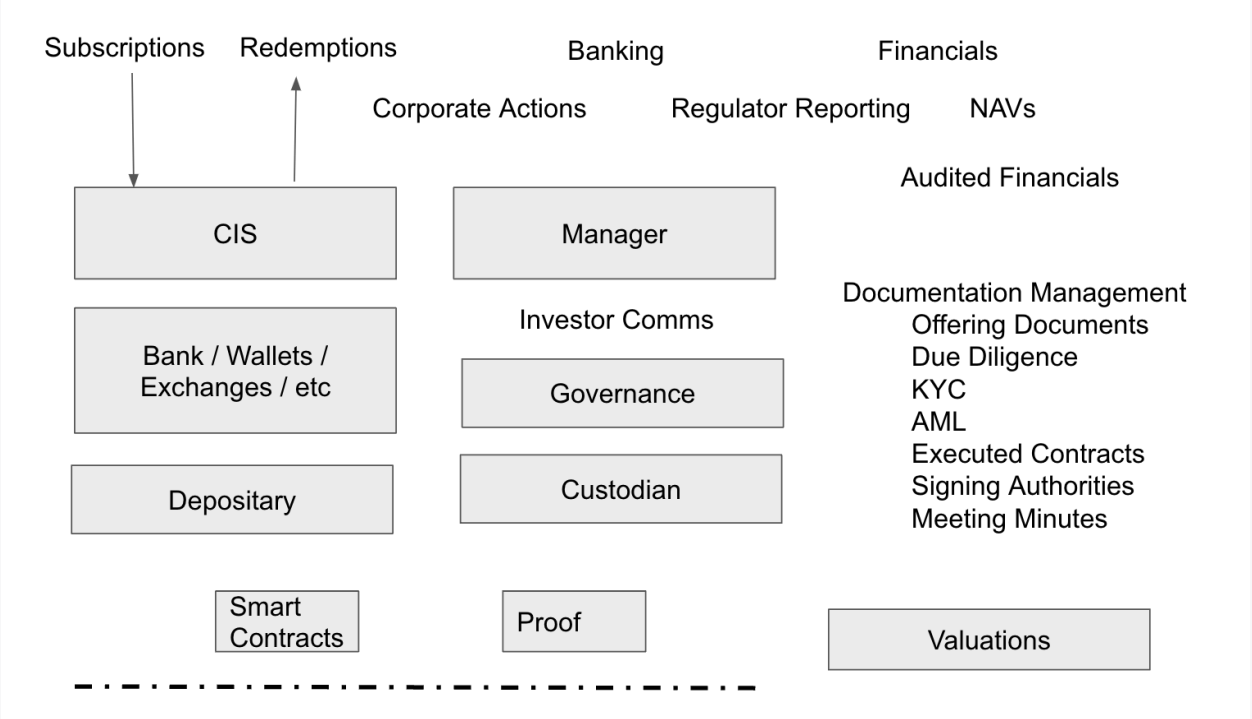
Total Non-Current Assets	[Amount]		
		Equity	
Total Assets	[Amount]	Share Capital	[Amount]
		Retained Earnings	[Amount]
		Other Reserves	[Amount]
		Total Equity	[Amount]
Total Liabilities and Equity	[Amount]	Total Liabilities and Equity	[Amount]
Notes:			
The assets section of the balance sheet is divided into current assets and non-current assets. Current assets are those that are expected to be realized within 12 months or the normal operating cycle, whichever is longer. Non-current assets are those that are expected to be held for more than 12 months or the normal operating cycle, whichever is longer.			
The liabilities section of the balance sheet is also divided into current liabilities and non-current liabilities. Current liabilities are those that are expected to be settled within 12 months or the normal operating cycle, whichever is longer. Non-current liabilities are those that are expected to be settled after 12 months or the normal operating cycle, whichever is longer.			
Equity represents the residual interest in the assets of the fund after deducting liabilities. It includes share capital, retained earnings, and other reserves.			

NAV Reporting

NAV reporting is a critical component of fund administration processes. However, the process can be time-consuming and prone to errors, particularly when performed manually. By using a modular blockchain, smart contracts can automate the NAV reporting process.

For example, a smart contract can be created to automatically calculate the fund's net asset value (NAV) based on the current asset valuations and accounting records. The NAV can then be reported to investors and other stakeholders automatically, reducing the need for manual intervention.

Typical Structure



A GP/LP structure is a common governance structure used in private equity and venture capital funds, where the General Partner (GP) manages the fund and makes investment decisions on behalf of the Limited Partners (LPs) who provide the capital. The structure combines the tax benefits of a limited partnership with the legal benefits of a general partnership. It allows the investors to enjoy the limited liability of a limited partnership, while also providing the flexibility of a general partnership. This structure also allows investors to share profits and losses in proportion to their percentage of ownership. Furthermore, it allows the investors to create their own management structure and specify their own roles in the venture. This structure is often used for real estate investments, but can be used in a variety of other areas. Here is a typical governance structure for a GP/LP structure:

1. General Partner (GP): The GP is the manager of the fund and is responsible for making investment decisions and managing the portfolio. The GP typically has a fiduciary duty to act in the best interests of the LPs.
2. Limited Partners (LPs): The LPs are the investors in the fund and provide the capital for the GP to invest. They have limited liability and are not involved in the day-to-day management of the fund.
3. Advisory Board: The Advisory Board is a group of individuals who provide advice and guidance to the GP on investment decisions and fund strategy. The Advisory Board is typically appointed by the GP and consists of individuals with relevant experience in the industry.
4. Management Company: The Management Company is a separate entity that provides management and administrative services to the fund. The Management

Company is typically owned by the GP and is responsible for managing the fund's operations.

5. Investment Committee: The Investment Committee is a group of individuals who review and approve investment decisions made by the GP. The Investment Committee is typically made up of senior members of the GP and may also include representatives from the LPs.
6. Fund Administrator: The Fund Administrator is responsible for managing the administrative tasks of the fund, such as accounting, reporting, and compliance. The Fund Administrator is typically appointed by the GP and may be a third-party service provider.
7. Legal Counsel: Legal Counsel provides legal advice and support to the GP and the fund. They are responsible for ensuring that the fund is in compliance with all relevant laws and regulations.
8. The governance structure of a GP/LP structure is designed to ensure that the fund is managed in the best interests of the LPs and that the GP is held accountable for its decisions. The roles and responsibilities of each party are clearly defined to ensure that the fund operates efficiently and effectively.

Corporate actions such as income or capital distributions can be accounted for and recorded on the blockchain as creditors and reconciled to cash movements.

Conclusion

In conclusion, fund administration processes can be operated on a modular blockchain, leveraging the benefits of smart contracts to automate and streamline various tasks. This approach can reduce the need for manual intervention, reduce the risk of errors, decrease the costs of operating, increase efficiency and transparency. It would be recommended that the process is run in parallel with existing legacy systems and processes.

References

Polygon Avail: Unlocking the Modular Blockchain Future
https://polygon.technology/blog/polygon-avail-unlocking-the-modular-blockchain-future Polygon Labs April 2022

Central Bank of Ireland: Fund Management Companies – Guidance
https://www.centralbank.ie/docs/default-source/regulation/industry-market-sectors/funds/ucits/guidance/fund-mancos-guidance.pdf 2016

Accdefi: double entry contract accounting on a distributed accounting ledger
https://accdefi.com/#whitepaper May 2021

PWC: 2021 Manual of accounting series
https://www.pwc.co.uk/services/risk/insights/manual-of-accounting.html

Morning Star: Should You Worry About a Fund Firm Failing? https://www.morningstar.co.uk/uk/news/203912/should-you-worry-about-a-fund-firm-failing.aspx Aug 2020