Musereum

Artem Aler Dan

20 ноября 2017 г.

Оглавление

Глава 1

Техническое описание платформы

1.1 Обзор

Задачи протокола «Musereum» необходимо разделить на составляющие:

- Обеспечение единого состояния системы без необходимости централизации вычислений для обеспечения отказоустойчивости;
- Децентрализованное хранилище большого объема данных без возможности цензуры путем блокировки единого (централизованного) источника данных;
- Предоставление доказательства целостности системы для аудирования исторических изменений;
- Интерфейс внесения изменений в будущее состояние системы.

Ведение и изменение единого состояния системы «Musereum» без необходимости централизации вычислений гарантируется использованием концепции «блокчейн» с использованием виртуальной машины Ethereum (EVM) *(Подробнее в Блокчейн)*. Для хранение большого объема данных (аудио-треки, метаданные, текстовое и графическое описание) утилизируется технология «IPFS» *(Подробнее в Децентрализованное хранилище.)*

1.1.1 Соглашение о наименовании

Здесь и далее по тексту все сущности с определенным содержанием или интерфейсом будут закреплять за собой краткое математическое наименование с общим правилом: *заглавная латинская буква, но не N, B, Z*, например:

$$Y \text{ is Notary} \tag{1.1}$$

Все глобально определенные контракты будут закреплять за собой строчный греческий символ, например, контракт управления списком нотариусов – ν и специальную запись ассоциированную с конкретным функционалом контракта:

$$\nu \text{ is Notary Registry Contract} \tag{1.2}$$

$$\nu_{vote} \text{ is vote contract method} \tag{1.3}$$

Множества всех известных сущностей определяются как *принятое наименование сущности в black-board записи, но не* $\mathbb{N}, \mathbb{B}, \mathbb{Z}$, например: список нотариусов:

$$\mathbb{Y} \text{ is set of all known Notaries} \tag{1.4}$$

$\mathbb{N}$ – множество всех целых чисел в диапазоне $(0; 2^{256} - 1)$. $\mathbb{N}_1$ – подмножество $\mathbb{N}$ без 0.

$\mathbb{Z}$ – множество всех целых чисел в диапазоне $(-2^{255}; +2^{255} - 1)$.

$\mathbb{B}$ – множество всех байтов $(0; 255)$.

$\mathbb{X}^i$ – множество размерности i где каждый элемент множества пренадлежит $\mathbb{X}$, например:

$$\mathbb{B}^{32} \text{ is a set of } \mathbb{B} \text{ with size } 32 \tag{1.5}$$

Получение элемента множества может быть записано двумя способами: подстрочным индексом или в квадратных скобках:

$$\begin{aligned} \mathbb{Y}_2 &= \mathbb{Y}[2] \\ \mathbb{Y}_n &= \mathbb{Y}[n] \end{aligned} \tag{1.6}$$

Внешние данные представлены в формулах как элементы множества Γ с удобно читаемыми наименованиями (или сокращениями если это оговорено), например: глобальное время в UNIX формате:

$$\Gamma_{time} \text{ is world time variable} \tag{1.7}$$

Все временные переменные в формулах записываются как строчная латинская буква, например:

$$\begin{aligned} &\text{Let } a \text{ is a second notary in list of notaries} \\ &a = \mathbb{Y}[2] \end{aligned} \tag{1.8}$$

1.1.2 Математические функции и символы

Во всех формулах используется $|\mathbb{Y}|$ как операция взятия количества элементов множества $\mathbb{Y}$.

Для сокращения и/или удобства прочтения математической нотации вводятся следующие соглашения и функции:

Поиск индекса в множестве – $\mathcal{I}$:

Let a is a set of values:

$$a = 100, 200, 300$$

$$f(v, x) = \{i \mid i \in \mathbb{N}, i > |v|, v_i = x\} \tag{1.9}$$

$$\mathcal{I}(v, x) = \begin{cases} min(f(v, x)), & \text{if } |f(v, x)| > 0 \\ -1, & \text{overwise} \end{cases} \tag{1.10}$$

$$\tag{1.11}$$

$$r1 = \mathcal{I}(a, 200) = 1$$

$$r2 = \mathcal{I}(a, 400) = -1 \tag{1.12}$$

Определения именованного кортежа – $\mathcal{O}$:

Let a is a set of names:

$$a = \{name, age, height\}$$

and let b is a set of indexes:

$$b = \{0, 1, 2\} = \{x \mid x \in \mathbb{N}, x < |a|\} \tag{1.13}$$

finaly r is a target named tuple, where

$$r = \mathcal{O}(a) = \mathcal{O}(name, age, height) \tag{1.14}$$

$$r = \{x_i \mid i \in a, x_i = 0\} \tag{1.15}$$

Определения кортежа события – $\mathcal{E}$:
События специальный именнованный кортеж, содержащий в себе кроме целевых данных, общие для любого события: `event`, `txHash`, `txIndex`, `blockHash`, `blockHeight`, `address`.

Let a is a set of default event fields:

$$a = \{event, txHash, txIndex, blockHash, blockHeight, address\}$$

Let b is a set of extra event fields:

$$b = \{asset, index\}$$

finaly e is a target event tuple, where

$$r = \mathcal{E}(b) \tag{1.16}$$

$$r = \mathcal{O}(\{asset, index\} \cup a) \tag{1.17}$$

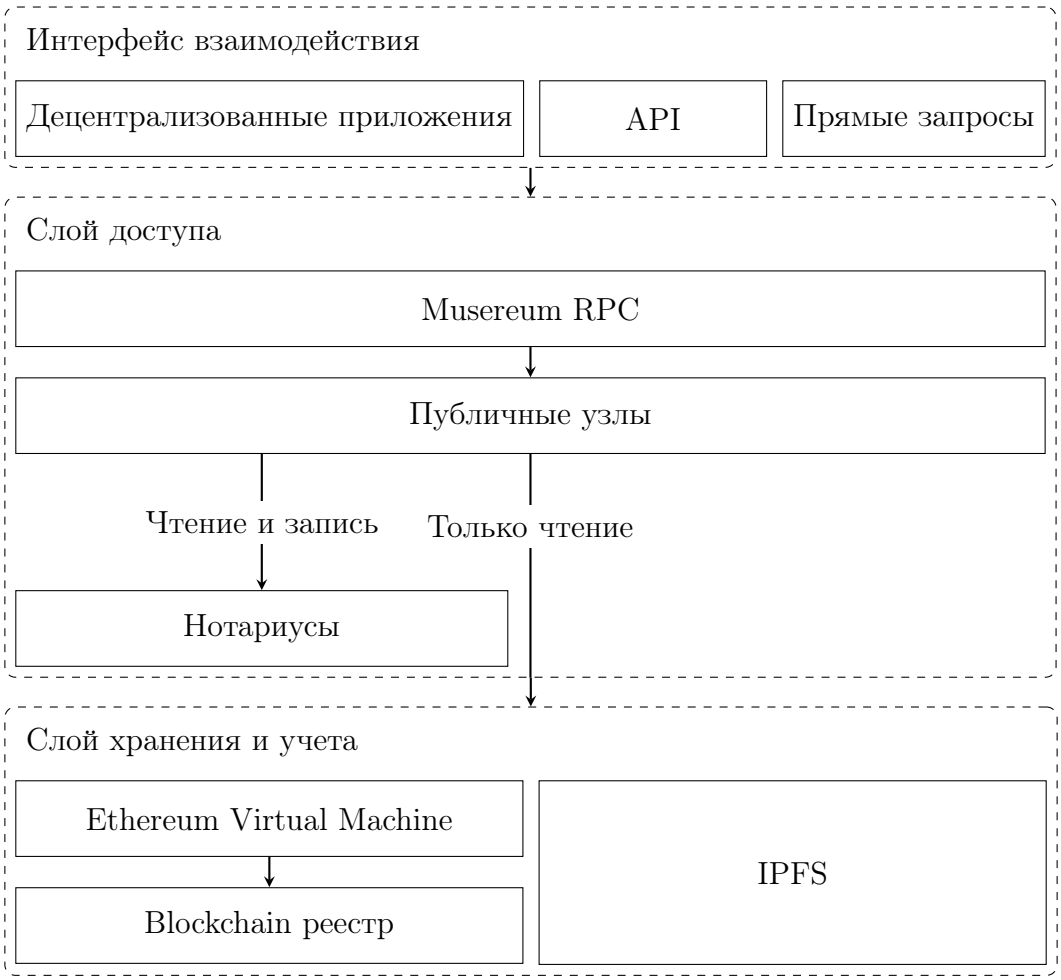
1.2 Архитектура платформы

Основная задача платформы предоставить пользователям возможность для:

1. Чтения и интерпретации данных
2. Внесения изменений
3. Поиска данных
4. Аудита и получения доказательства отсутствия фальсификации данных

Для предоставления всех вышеуказанных возможностей платформа разделена на логические слои:

- Хранение и учет
  - Blockchain
  - Децентрализованное хранилище
- Слой доступа на чтение и запись
  - Публичные узлы
  - Узлы нотариусы
- Интерфейс взаимодействия
  - Децентрализованные приложения
  - Application Programming Interface



1.2.1 Слой хранения и учета

Базовым слоем платформы является слой хранения и учета. Задача слоя предоставить возможность ведения децентрализованного реестра состояний («блокчейн») и децентрализованное хранения большого объема данных.

Каждый элемент слоя: блокчейн, хранилище, виртуальная машина, распределены по сети (узлам) и каждый желающий может создать свою, локальную копию.

1.2.2 Слой доступа

Задачи слоя доступа: предоставление, по запросу, информации из базового слоя и получение запросов об изменении.

Создание узла для организации дополнительной точки доступа никак не ограничивается платформой и доступно любому желающему. Данная особенность выгодно отличает платформу Musereum от других, централизованных музыкальных платформ:

- Гарантирует 100% up-time сети
- Устойчивость сети к цензуре
- Возможность заинтересованным сторонам создать собственную, локальную копию сети, а не доверять полученной информации от третьих лиц

1.2.3 Слой интерфейсов взаимодействия

Верхним слоем платформы является слой интерфейсов взаимодействия. Задача слоя получать пользовательский ввод и формировать из него понятные для протокола запросы. Запросы делятся на два типа: чтение и запись.

Платформа не накладывает каких-либо ограничений на чтение данных из блокчейна и децентрализованного хранилища. Единственное требование – формирования запроса в соответствии с соглашением об API.

Для записи (изменения состояния, сохранении данных) от клиента требуется предоставление валидной подписи запроса основанной на его уникальном приватном ключе и соблюдение других требований к записи (наличие прав, денежных средств для оплаты работы протокола и другие).

Для создания запроса не требуется организации собственной точки доступа (синхронизации узла с сетью) и в качестве клиента может выступать server-less приложение на web-технологиях *(подробнее в Децентрализованные приложения)*

1.3 Блокчейн

Протокол «Musereum» открытый, публичный, но требующий разрешения на производство блоков основанный на протоколе «Ethereum». Консенсус по единому состоянию сети в децентрализованной структуре производителей блоков достигается использованием алгоритма Proof-of-Authority (далее PoA). Для достижения консенсуса через PoA требуется:

1. Общеизвестный список нотариусов ($\mathbb{Y}$) с разрешением на производство блоков;
2. Контракт управления списком нотариусов (ν).

Изменение списка нотариусов (исключение, назначение нотариусов) изменяется контрактом управления по результатам голосования текущих нотариусов (большинством).

Для первичного назначения составляется список из 12 нотариусов.

Задача нотариуса:

1. проверить поступившие транзакции участников системы,
2. собрать в единый блок изменения общего состояния,
3. заверять своей уникальной подписью собранный блок.

Для соблюдения консенсуса, в протокол вводится внешняя константа Γ_{step}, определяющая количество секунд в одном временном шаге или время между блоками. Musereum определяет константу Γ_{step} как 5, или 1 блок раз в пять секунд.

$$\Gamma_{step} = 5 \tag{1.18}$$

По соглашению с алгоритмом консенсуса PoA, нотариус наделен правом создать один блок (K) за a временных штампов Γ_{time}. Где a равно количеству нотариусов:

$$a = |\mathbb{Y}| \tag{1.19}$$

Выбор индекса нотариуса i из множества $\mathbb{Y}$ для создания блока на штамп времени b происходит по формуле:

$$b = \frac{\Gamma_{time}}{\Gamma_{step}}, \quad b \in \mathbb{N} \tag{1.20}$$

$$i = b \ mod \ a \tag{1.21}$$

1.3.1 Выбор версии

В случаях если нотариусы не могут прийти к общему (единому) состоянию системы и происходит «форк» (от англ. вилка) – появляются две или более цепочки блоков ($\mathbb{K}$), сеть определяет вес ($\beta_{score}(\mathbb{K}_c)$) каждой из цепочек исходя из количества нотариусов участвующих в создании блоков:

$$h = |\mathbb{K}_c| \quad h \text{ is length of blockchain} \tag{1.22}$$

$$\beta_{score}(\mathbb{K}_c) = \Gamma_{u128max} * h - m \tag{1.23}$$

Сеть всегда выбирает цепочку с большим весом по β_{score}.

1.3.2 Финализация блока

По консенсусу необратимым считается блок в цепочке после которого более 50% нотариусов создали 2 или более блока.

Таким образом минимальное время полного подтверждения блока (C) будет:

$$C = 2 * \Gamma_{step} * |\mathbb{Y}| \tag{1.24}$$

Или 120 секунд при изначальных настройках протокола: 12 нотариусов и блок раз в пять секунд

1.3.3 Свидетельство сделки

Для совершения любого действия требующего изменения состояния системы от пользователя требуется формирования транзакции (T) подписанной уникальной крипто-подписью (T_{sign}).

Наличие крипто-подписи гарантирует сети, что транзакция была создана владельцем учетной записи и является изъявлением воли владельца.

Нотариус очередь которого формировать блок проверяет валидность транзакции и исполняет связанный код:

1. Отправитель имеет право на формирование транзакции к данному адресу – $\rho_{allowTx}(T_{from}, T_{to})$ (подробнее в Пра-

вила взаимодействия с EVM)

2. Транзакция соответствует формальным признакам – $\rho_{initialValid}(T)$
3. Проверка подписи владельца учетной записи – $\rho_{checkSign}(T)$
4. Наличие на счету ETM токенов для оплаты аванса работы вычислительной машины – $\rho_{upFront}(T)$
5. Выполнение связанного кода EVM не вызвало исключения – $\rho_{exception}(\rho_{execute}(T))$

$$\begin{aligned}\rho_{success}(T) = \quad & \rho_{allowTx}(T_{from}, T_{to}) > 0 \wedge \\ & \rho_{initialValid}(T) > 0 \wedge \\ & \rho_{checkSign}(T) > 0 \wedge \\ & \rho_{upFront}(T) > 0 \wedge \\ & \rho_{exception}(\rho_{execute}(T)) = 0\end{aligned} \tag{1.25}$$

Убедившись в валидности транзакции нотариус упаковывает транзакцию в блоки оповещает сеть о блоке и изменении состояния сети ($\mathbb{W}_h$).

$$\mathbb{W}_h = \rho_{state}(\mathbb{W}_{h-1}, \mathbb{K}_h) \tag{1.26}$$

Другие нотариусы получают блок, проверяют ($\rho_{success}(T)$) и принимают решение о принятии его в цепочку. Создавая блок над $\mathbb{K}_h$ нотариус подтверждает его валидность и всех вложенных в него транзакций.

Таким образом количество заверяющих подписей у транзакции может быть от 1 до $|\mathbb{Y}|$.

Минимальное время получения $|\mathbb{Y}|$ подписей:

$$a = \Gamma_{step} * |\mathbb{Y}| \tag{1.27}$$

1.3.4 Вознаграждение за производство блоков

Вознаграждение за блок (R) состоит из двух частей: новой эмиссии и комиссий за совершенные действия.

В платформе Musereum эмиссия новых токенов ETM происходит в процессе создания блока. Размер эмиссии фиксирован – 3 ETM за каждый блок.

Размер комиссии за совершенные действия высчитывается для каждой транзакции отдельно. Отправитель указывает в транзакции стоимость единицы сложности ($T_{gasPrice}$) которую он готов потратить в качестве компенсации работы сети.

Финальная комиссия высчитывается по формуле:

$$x = T_{gasPrice} * \rho_{dificulty}(\rho_{execute}(T)) \tag{1.28}$$

Общее вознаграждение за блок:

Let a is a list of transactions of block $\mathbb{K}_h$

$$\begin{aligned} a &= \mathbb{K}[h]_{transactions} \\ \mathbb{R}_h &= 3 + \sum_{i=0}^{|a|} a_i \end{aligned} \tag{1.29}$$

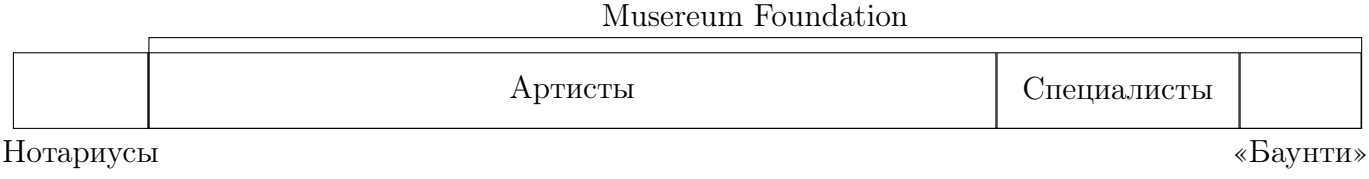
Распределение вознаграждения

В отличие от протокола Ethereum и Bitcoin, в сети Musereum нету майнеров и основную ценность создают артисты, выбирающие Musereum платформой для выпуска и дистрибуции своих произведений.

Musereum высоко ценит вклад артистов в создание ценности и направляет большую часть награды за блок на субсидирование артистов.

Общая схема распределения:

- 90% – Musereum Foundation:
  - 70% – программы субсидирования артистов, например, Pay-per-Play
  - 20% – программа субсидирования аффилированных специалистов: регистраторы музыкальных активов, аудиторы смарт-контрактов
  - 10% – программы вознаграждения: Fans Bounty, Clipmaker Bounty и т.д.
- 10% – Производители блоков



1.3.5 Виртуальная машина Ethereum

1.3.6 Смарт-контракты Ethereum

1.3.7 Правила Musereum по взаимодействию с EVM

Для повышения безопасности участников платформы работа со смарт-контрактами EVM в Musereum отличается от таковой в родительской сети Ethereum.

Загрузить смарт-контракт в сеть может любой участник сети, но такой смарт-контракт получается статус $P_{development}$ и взаимодействовать с ним могут только аффилированные адреса учетных записей сети. Изначально это адрес пользовательской учетной записи с которой был загружен контракт.

Правила взаимодействия со смарт-контрактами регулируются смарт-контрактом разрешений: ρ.

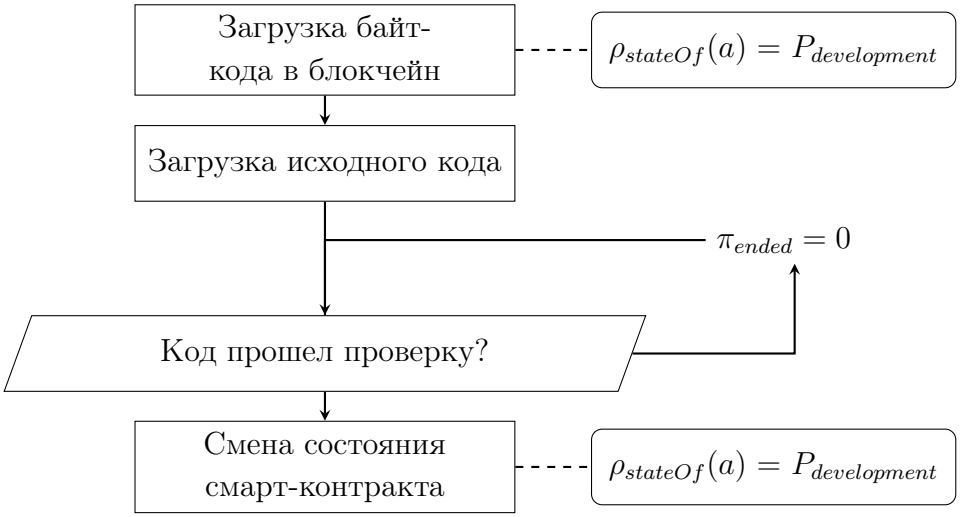
Любой вызов смарт-контракта изначально запрашивает разрешение на совершение данного вызова – $\rho_{allowTx}(T_{from}, T_{to})$

$$a = \rho_{stateOf}(T_{to}) \tag{1.30}$$

$$b = \rho_{affilateWith}(T_{to}) \tag{1.31}$$

$$\rho_{allowTx}(T_{from}, T_{to}) = \begin{cases} 1, & \text{if } a = P_{production} \\ 1, & \text{if } a = P_{development} \wedge T_{from} \in b \\ 0, & \text{overwise} \end{cases} \tag{1.32}$$

Для получения статуса $P_{production}$ автор смарт-контракта обязан предоставить на проверку исходный код смарт-контракта и пройти процедуру проверки *(подробнее в Реестр смарт-контрактов)*



1.4 Децентрализованное хранилище

1.4.1 Обзор технологии

1.4.2 Соглашение о хранении данных

1.4.3 Система удобочитаемых наименований

1.5 Децентрализованные приложения

Децентрализованные приложения Musereum реализуют код для слоя учета и хранения и код для интерфейса взаимодействия. Протокол позволяет взаимодействовать приложениям между собой.

1.5.1 Governance dApp

Децентрализованное приложение управления списком нотариусов и проверки их прав на производство блоков.

1.5.2 Contract Registry

Децентрализованное приложение для регистрации смарт-контрактов платформы и их учета.

Смарт-контракт приложения позволяет получить:

1. Список всех зарегистрированных смарт-контрактов
2. Получить состояние контракта ассоциированное с адресом:
   (a) `Unknown` – неизвестное состояние (по умолчанию для неизвестных для Contract Registry адресов)
   (b) `Development` – смарт-контракт зарегистрирован, но находится в разработке (взаимодействие доступно только аффилированным пользователям)
   (c) `Production` – смарт-контракт зарегистрирован и доступен для взаимодействия всем желающим
   (d) `Halt` – смарт-контракт зарегистрирован, но отключен (по решению автора)
   (e) `Suspend` – смарт-контракт зарегистрирован, но отключен (по решению аудиторов)
3. Получить ссылку на репозиторий с исходным кодом
4. Получить ссылку на ABI JSON для

взаимодействия со смарт-контрактом

5. Получить список аффилированных аудиторов
6. Получить экземпляр голосования *(подробнее в VotingdApp)* для принятия решения о смене статуса

Исходный код и проверка смарт-контракта

Валидация смарт-контрактов происходит в трех направлениях:

- Валидация исходного кода на:
  1. Наличие уязвимостей
  2. Соответствие заявленной бизнес-логики
  3. Отсутствие алгоритмических ошибок
- Соответствие праву юрисдикций в которых работает платформа Musereum
- Соответствие концепции Musereum

Для одобрения смарт-контракта необходимо единогласное согласие от всех аффилированных аудиторов по трем направлениям.

Отслеживать ход валидации можно в Voting dApp в соответсвующей `ContractValidationBallot`.

1.5.3 Musereum Name System (MNS)

Реестр ассоциированных имен к адресам в платформе Musereum? требуется для предоставления доступа к актуальным версиям приложений протокола без необходимости запоминать внутренний адрес (20 символьный hex-адрес).

Запись об имени может создать любой пользователь сети для произвольного адреса. Количество имен для адреса никак не ограничено.

Для защиты от нецелевого использования имен, за регистрацию нового имени взимается плата в ETM равная текущей стоимости.

Запись об имени является производным от MNS смарт-контрактом: `NameRecordContract`.

Смарт-контракт содержит информацию о владельце имени и ассоциированном адресе в сети и предоставляет возможности:

Смены владельцем ассоциированного адреса – Смены владельцем владельца (уступление прав) – MNS накладывает ограничения на доступные имена: Длина имени не может превышать 20 символов (или менее при non-ASCII символах) Длина имени не может быть менее 4-х символов

Интерфейс взаимодействия

`mapping (address => address) names` – (η_{names})
Ассоциативный список адрессов к адрессам смарт-контрактов регистрации имени

`uint price` – (η_{price})
Стоимость регистрации имени в токенах ETM

`event NewName(address indexed name, byte20 ascii)` – ($\eta_{NewName}$)
Журнал событий регистрации новых имен

`event AssociateWith(address indexed name, address indexed target)` – ($\eta_{AssociateWith}$)
Журнал событий ассоциирования адреса со смарт-контрактом регистрации имени

`function buyName(byte20 acsii) payable` – (η_{buy})
Метод покупки имени. Возвращает `address` если покупка прошла успешно и `throw` при ошибке.

$$\eta_{buy}(s) = \begin{cases} throw, & \text{if } s \in \eta_{NewName} \\ throw, & \text{if } |s| < 4 \\ throw, & \text{if } T_{value} < \eta_{price} \\ address, & \text{overwise} \end{cases} \tag{1.33}$$

Смарт-контракт NameRecord – (η')

`address owner` – (η'_{owner})
Текущий владелец имени

`byte20 ascii` – (η'_{ascii})
Зарегистрированное имя

`address target` – (η'_{target})
Текущая ассоциация с адресом на платформе Musereum

`function associate(address target)` – ($\eta'_{associate}$)
Изменить ассоциацию имени на новый адрес. Возвращает `true` если прошло успешно и `throw` при ошибке.

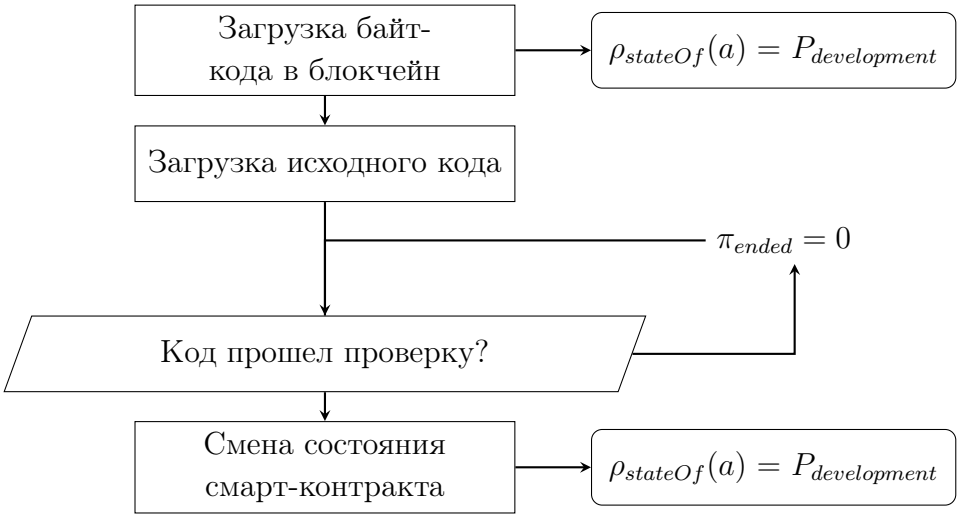
$$\eta'_{associate}(a) = \begin{cases} throw, & \text{if } T_{from} \neq \eta'_{owner} \\ throw, & \text{if } a = 0 \\ true, & \text{overwise} \end{cases} \tag{1.34}$$

1.5.4 Voting dApp

Приложение децентрализованного голосования состоит из связанных элементов:

- `BallotsManager` (ϕ)
  реестр голосований
- `Ballot` (π)
  абстрактный смарт-контракт голосования
- `VotingRightsToken` (ϕ^{τ})
  специализированный токен выдаваемый участникам голосования для доказательства своих прав на участие в голосовании
- Интерфейса взаимодействия с голосованиями

Для создания голосования инициатор предложения должен сформировать валидный `Ballot` – смарт-контракт ведущий учет голосов и реализующий `applyProposal` метод для изменения состояния в соответствии с предложением.



Смарт-контракт Ballot

Абстрактный смарт-контракт описывающий требования к финальной реализации предложений для голосования.

`address votingToken` – (π_{token})
Ассоциированный с голосованием токен наделяющий держателя правом голоса

`function vote(bool)` – (π_{vote})
Внешний метод для записи решения, вызывающего метод.

`function applyProposal()` – (π_{apply})
Абстрактный метод применения предложения по факту успешного голосования. Реа-

лизуется в дочерних смарт-контрактах.

`uint agreeVotesCount` – (π_{agree})
Количество токенов переданных для принятия решения.

`uint rejectVotesCount` – (π_{reject})
Количество токенов переданных для отклонения решения.

`uint endTime` – (π_{end})
Время завершения голосования.

`function isEnded()` - (π_{ended})
Возвращает 1 если время для голосования закончилось и 0 в других случаях.

$$\pi_{ended} = \begin{cases} 1, & \text{if } \pi_{end} > \Gamma_{time} \\ 0, & \text{overwise} \end{cases} \tag{1.35}$$

`function voteMajorityRule()` – (π_{rule})
Результат функции $f(x) = 1/x$ где x это отношение голосов за принятие к общему количеству голосов достаточное для принятие решения:

$$\pi_{rule} = \begin{cases} \infty, & \text{if } x = 0 \\ \frac{1}{x}, & \text{overwise} \end{cases} \tag{1.36}$$

`function compare()` – ($\pi_{compare}$)
Возвращает 1 если количество голосов за принятие решение удовлетворяет условия голосования для принятия решения.

$$\pi_{compare} = \begin{cases} 1, & \text{if } \pi_{agree} * \pi_{rule} > \pi_{agree} + \pi_{reject} \\ 0, & \text{overwise} \end{cases} \tag{1.37}$$

`function isSuccess()` – ($\pi_{success}$)
Возвращает 1 если голосование успешно завершено и 0 в других случаях.

$$\pi_{success} = \begin{cases} 1, & \text{if } \pi_{ended} \wedge \pi_{compare} \\ 0, & \text{overwise} \end{cases} \tag{1.38}$$

1.5.5 Musical Assets Registry

Приложение регистрации музыкальных активов на платформе Musereum. Приложение состоит из:

1. `MusicalAssetsRegistry` (MAR – ψ)
   Корневой смарт-контракт приложения. Выполняет функции фабрики и реестра.
2. `AssetContract` – (α)
   Производимые в MAR смарт-контракты музыкальных активов.
3. Ассоциированные с `AssetContract`'ами смарт-контракты децентрализованных лейблов
4. `AssetRegistryBallot` – ($\pi^{regAsset}$)
5. `AssetUnregistryBallot` – ($\pi^{unregAsset}$)
6. Интерфейса взаимодействия – веб-приложения в рамках Musereum Wallet

Смарт-контракт MusicalAssetsRegistry

Задачи смарт-контракты: учет зарегистрированных музыкальных активов, создание заявок на регистрацию музыкального актива через голосование в Voting dApp.

Голосование проходит среди авторизованных системой регистраторов.

Реестр полностью публичный и предоставляет возможности для аудита данных всем участникам сети.

Получение исторических данных возможно через создание запроса к контракту за журналом событий:

`event NewAsset(address indexed asset, uint indexed index)` – $(\psi_{NewAsset})$
Возвращает журнал событий добавления музыкальных активов в реестр с присвоенными им индексами в реестре. *Событие создается в момент создание заявки на регистрацию, а не при прохождение регистрации.*

$$\psi_{NewAsset} = \mathbb{E}[asset, index], \quad a \in \mathbb{A} \wedge i \in \mathbb{N} \tag{1.39}$$

`event RegistryAsset(address indexed asset, uint indexed index)` – $(\psi_{RegistryAsset})$
Возвращает подмножество журнала событий $\psi_{NewAsset}$ ассоциированных к адресам активов успешно прошедших голосование за добавление в публичный реестр.

$$\psi_{RegistryAsset} \subset \psi_{NewAsset} \tag{1.40}$$

`event UnregistryAsset(address indexed asset, uint indexed index)` – $(\psi_{UnregistryAsset})$
Возвращает подмножество $\psi_{RegistryAsset}$ для музыкальных активов удаленных из публичного реестра Musereum по решению сообщества.

$$\psi_{UnregistryAsset} \subset \psi_{RegistryAsset} \subset \psi_{NewAsset} \tag{1.41}$$

`mapping (address => uint) assets` – $(\psi_{assets}(a))$
Ассоциативный словарь добавленных в реестр музыкальных активов.

$$\psi_{assets}(a) = \begin{cases} i, & \text{if } a \in \mathcal{S}(\psi_{NewAsset}, [asset]) \\ 0, & \text{overwise} \end{cases}, \quad a \in \mathbb{A} \wedge i \in \mathbb{N}_1 \tag{1.42}$$

`address[] public assetsIndex` – $(\psi_{index}(i))$
Список всех добавленных в реестр музыкальных активов. Индекс актива в списке равен индексу в ассоциативном словаре минус 1.

$$i = \psi_{assets}(a), \quad a \in \mathbb{A} \wedge i \in \mathbb{N} \tag{1.43}$$

$$\psi_{index}(i-1) \equiv a, \quad \text{if } i > 0 \tag{1.44}$$

Смарт-контракт AssetContract

Запись в блокчейне о зарегистрированном музыкальном активе. Содержит всю необходимую информацию для идентификации музыкального актива и определения природы актива.

`string name` – (α_{name})
Ассоциированное с активом имя

`Multihash meta` – (α_{meta})
Хеш ассоциированной с активом мета-информацией (хранится в децентрализованном хранилище)

$$\alpha_{meta} = \mathcal{O}[hash, hashFunction, size], \tag{1.45}$$

$$\alpha_{meta}[hash] \in \mathbb{B}_{32} \wedge \alpha_{meta}[hashFunction] \in \mathbb{B} \wedge \alpha_{meta}[size] \in \mathbb{B} \tag{1.46}$$

`event SetAssetType(uint8 indexed typeId)` – ($\alpha_{SetAssetType}$)
Возвращает журнал событий определения типа актива, содержащее индекс перечисления типом активов (`AssetTypes` – α^{types}).

$$\alpha^{types} \in \mathbb{B} \tag{1.47}$$

$$\alpha_{SetAssetType} = \mathbb{E}[typeId], \quad i \in \alpha^{types} \tag{1.48}$$

`mapping (uint8 => bool) type` – ($\alpha_{type}(i)$)
Ассоциативный словарь типов актива.

$$\alpha_{type}(i) = \begin{cases} 1, & \text{if } i \in \alpha_{SetAssetType} \\ 0, & \text{overwise} \end{cases}, \quad i \in \alpha^{types} \tag{1.49}$$

Смарт-контракт AssetRegistryBallot

Реализация Ballot смарт-контракта для проведения голосования за одобрение актива в реестре Musereum.

Смарт-контракт AssetUnregistryBallot

Реализация Ballot смарт-контракта для проведения голосования об исключении актива из реестре Musereum.

1.5.6 Decentralized Autonomous Labels

Децентрализованный автономный лейбл – это смарт-контракт управления ассоциированным с ним музыкальным активом. В возможности DAL входит:

1. Определение устава лейбла регулирующего правила принятия решений;
2. Выпуск и продажа лицензий на коммерческое использование связанного музыкального актива;
3. Аккумуляция и распределение дохода музыкального актива,
4. Распределение и учет прав на музыкальный актив с цифровым доказательство – токеном.

Управление музыкальным активом происходит по принципу демократического голосования держателями токенов.

Возможности DAL реализует набор смарт-контрактов:

- `DALRegistry` – (δ^R)
  Реестр зарегистрированных децентрализованных автономных лейблов
- `DALContract` – (δ)
  Смарт-контракт децентрализованного автономного лейбла
- `DALToken` – (τ^δ)
  Смарт-контракт токена подтверждающего права держателя на участие в децентрализованном лейбле
- Набор смарт-контрактов голосования реализации Ballot:
  - `CharterChangeVotingRulesBallot`
    Предложение по голосованию за смену правил голосования о внесении изменений в устав
  - `CharterChangeProposalBallot`
    Предложение по внесению изменений в устав
  - `AddLicenseBallot`
    Предложение по созданию новой лицензии
  - `CloseLicenseBallot`
    Предложение о прекращении продажи лицензии
  - `ReplaceLicenseBallot`
    Предложение о смене/внесению изменений в лицензию
- `DALAssetLicense` – (θ)
  Смарт-контракт лицензии на коммерческое использование
- `DALAssetLicenseRule` – (θ')
  Производный от лицензии смарт-контракт ограничения использования

Смарт-контракт DALRegistry

Реестр зарегистрированных автономных лейблов необходим для учета и доступа к лейблу управляющему ассоциированным музыкальным активом.

`mapping (address => address) assetLabels` – $(\delta^R_{labels}(a))$
Ассоциативный словарь созданных децентрализованных лейблов.

```
event CreateLabel(address indexed asset,
                  address indexed label,
                  uint indexed index)
```
– $(\delta^R_{CreateLabel})$

Возвращает множество событий создания децентрализованных лейблов. Событие включает в себя: адрес созданного лейбла, адресом ассоциированного музыкального актива и присвоенный лейблу индекс в реестре.

$$\begin{aligned} \delta^R_{CreateLabel} &\in \mathcal{E}(asset, label, index) \\ asset \in \mathcal{S}(\psi_{RegistryAsset}, [asset]) &\subset \mathbb{A}^{\alpha} \wedge \\ label &\in \mathbb{A}^{\delta} \wedge \\ index &\in \mathbb{N}_1 \end{aligned} \tag{1.50}$$

```
event CreateLabelBallot(address indexed label,
                        address indexed ballot)
```
– $(\delta^R_{CreateLabelBallot})$

Возвращает множество событий создания голосований децентрализованными лейблами.

Событие включает в себя: адреса децентрализованного лейбла и адрес созданного голосования.

$$\begin{aligned} \delta^R_{CreateLabelBallot} &\in \mathbb{E}[label, ballot] \\ label \in \mathcal{S}(\delta^R_{CreateLabel}, [label]) &\subset \mathbb{A}^{\delta} \wedge \\ ballot \in \mathcal{S}(\phi_{CreatedBallot}, [ballot]) &\subset \mathbb{A}^{\pi} \end{aligned} \tag{1.51}$$

```
event FinishLabelVoting(address indexed label,
                        address indexed ballot,
                        bool indexed result)
```
– $(\delta^R_{FinishLabelVoting})$

Возвращает подмножество $\delta^R_{CreateBallot}$ для завершенных голосований с явным указанием успеха или провала предложения.

$$\begin{aligned} f_Z(\mathbb{E}') &= \{\mathbb{E}'_i : \ \forall \ \mathbb{E}'_i[ballot][ended] > 0\} \\ \delta^R_{FinishLabelVoting} &= f_Z(\delta^R_{CreateLabelBallot}) \end{aligned} \tag{1.52}$$

`event NewLabelLicense(address indexed label, address indexed license)` – $(\delta^R_{NewLicense})$
Возвращает множество событий созданных децентрализованным лейблом лицензий.

`event RemoveLabelLicense(address indexed label, address indexed license)` – $(\delta^R_{RemoveLicense})$

Возвращает подмножество $\delta^R_{NewLicense}$ для снятых с продажи лицензий.

Смарт-контракт DALContract

Смарт-контракт управления музыкальным активом. Управление активом и связанными сущностями происходит путем вынесения предложения на голосование с предварительным распределением `VotingRightsToken` всем участникам сообщества в соответствии с балансом связанного токена `DALToken`.

Децентрализованный лейбл создается автоматически по факту регистрации связанного актива в реестре `MusicalAssetsRegistry`.

При создании децентрализованного лейбла выпускается 1,000,000,000 токенов подтверждения прав, для упрощения расчетов в связанных пользовательских интерфейсах фактическое число делится на 10^7.

Таким образом единоличный контроль лейблом эквивалентен 100.0000000 токенам, а минимальный голос в компании – 0.0000001%.

Права держателей токенов `DALToken`:

1. Участие в распределении роялти
2. Участие в голосовании за принятие решений
3. Уступление токенов другим участником платформы через обмен на иные токены или ETM

Смарт-контракт DALToken

Токен децентрализованного лейбла является ERC-20 совместимым токеном.

Для обеспечения работы системы распределения роялти в код стандартного ERC-20 токена добавленны необходимые изменения:

1. Сохраняется one-to-one связь с ассоциированным лейблом в поле `address label`
2. Внесены требуемые для оптимизации алгоритма распределения вызовы `withdrawRevenueFor()` в `transfer(...)` и `transferFrom(...)` функции для отправителя и получателя
3. Реализован метод `makeVoteToken()` для создания `VotingRightsToken`'а соответствующего текущим балансам

Распределение роялти

Роялти – полученные в ходе коммерческой активности связанного музыкального актива единицы ETM на баланс связанного децентрализованного автономного лейбла.

Каждый участник лейбла в любой момент времени может реализовать свое право на получение доли роялти в соответствии с текущим количество токенов.

Распределение роялти	
$a = T_{from}$	Initial withdrawal transaction
$d = \delta(T_{to})$	DAL instance
$r = \tau^{\delta}(d_{token})$	Rights tokens of corresponding DAL
$h = \mathbb{K}_n$	Height of latest block
$b_h = r_{balanceOf}(a)$	Balance of DAL rights tokens at block h
$t_h = r_{totalSupply}$	Total supply of rights token (typicaly is 10^9)
$s_h = \frac{b_h}{t_h}$	Stake of a at moment h
$I_h = x$	I_h Income of a DAL at moment h
$T_h = \sum_{i=1}^{h} I_i$	Total lifetime income of DAL
$W_h = s * (T_h - \sum_{i=1}^{h-1} W_i)$	Available withdrawal amount at moment h

(1.53)

Оптимизация алгоритма распределения

В целях оптимизации и устранения необходимости считать W_h для каждого уникального h, в смарт-контракте вводится вспомогательный ассоциативный список `incomeAtLatestWithdraw(address => uint)`, сохраняющий значение в момент последнего вывода средств для ассоциированного адреса.

Такой подход позволяет производить расчет роялти исключительно в момент когда этого требует бизнес-логика и только для связанных с ней держателей.

При уступлении прав третьему держателю (перевод токенов) автоматически запускается алгоритм расчета доступного роялти для текущего и будущего держателей.

Голосование за принятие решений

Устал децентрализованного лейбла

Продажа лицензий и смарт-контракт DALAssetLicense

Лицензирование “Musereum” включает в себя возможность проведения следующих типов лицензирования/

Продажа лицензий на коммерческое использование связанного музыкального актива основная задача децентрализованного лейбла, а получение роялти за продажу основная мотивация существования.

Все лицензии выпускаются в виде экземпляра смарт-контракта `DALAssetLicense` через интерфей усправления `DALContract`.

Задача корневого контракта `DALAssetLicense` определить природу лицензии, правила приобретения и ограничения на характер использования. `DALAssetLicense` является контейнером и предоставляет держателю неограниченную лицензию на коммерческое использование, ограничения же описываются в прозводных смарт-контрактах: `DALAssetLicenseRule`.

DALAssetLicenseRule

Абстрактный смарт-контракт, определяющий ограничение родительской лицензии, существует 7 базовых типов ограничений:

1. `EnumerateRule` – органичение перечислением
2. `RangeRule` – ограничение диапазоном
3. `ValueRule` – ограничение значением
4. `AddressRule` – ограничение адресом(ами)
5. `TextRule` – описательное ограничение
6. `AllOfRule` – группирующее ограничение по типу: все из группы
7. `AnyOfRule` – группирующее ограничение по типу: любое из группы

Финальная реализация ограничения может быть выбрана из предзаготовленных или создана под конкретную лицензию децентрализованным лейблом.

Сравнение ограничений

$$a = \theta'(original), \quad b = \theta'(target)$$

$$\theta'_{fitWith}(b) = \begin{cases} 1, & \text{if } a_{type} = b_{type} \ \wedge \ f_z(a,b) \\ 0, & \text{overwise} \end{cases} \tag{1.54}$$

`EnumerateRule`

$$f_z(a,b) = \begin{cases} a_{items} \subset b_{items}, & \text{if } a_{all} \\ |\{i : \forall\ a_{items}[i] \in b_{items}\}| > 0, & \text{overwise} \end{cases} \tag{1.55}$$

`RangeRule`

$$f_z(a,b) = a_{min} > b_{min} \ \wedge \ b_{max} > a_{max} \tag{1.56}$$

`ValueRule`

$$f_z(a,b) = \begin{cases} a_{value} = b_{value}, & \text{if } a_{equal} \\ a_{value} < b_{value}, & \text{if } a_{less} \\ a_{value} > b_{value}, & \text{if } a_{more} \end{cases} \tag{1.57}$$

`AddressRule`

$$f_z(a,b) = a_{addresses} \in b_{addresses} \tag{1.58}$$

`AllOfRule`

$$f_z(a,b) = ||\{\top : a_{items}[i]_{fitWith}(b) = \top\}|| = ||a|| \tag{1.59}$$

`AnyOfRule`

$$f_z(a,b) = ||\{\top : a_{items}[i]_{fitWith}(b) = \top\}|| > 0 \tag{1.60}$$

Таблица 1.1: Предзаготовленные типовые ограничения лицензий

Наименование	Описание
	`EnumerateRule`
`LocationRule`	Ограничение по географии использования и производные: `CountryRule` и `CityRule`
`ChannelRule`	Ограничение по канналу воспроизведения
`MediaRule`	Ограничение по типу носителя
`GenreRule`	Ограничение по жанру носителя (используется для кино, театра и т.д.)
`UsageRule`	Ограничение по типу использования
	`RangeRule`
`PlayTimesRule`	Ограничение количеству/длительности воспроизведения
`LifetimeRule`	Ограничение срока действия лицензии
`NumberOfCopyRule`	Ограничение по количеству копий
`PowerOfRule`	Ограничение по бюджету проекта и(или) доходу компании приобретателя
	`ValueRule`
`PriceRule`	Стоимость лицензии

Устав децентрализованного лейбла

Любой децентрализованный лейбл

1.5.7 Soundchain

Soundchain является первым проектом направленным на освоение Musereum Foundation. Основная задача Soundchain начисление роялти децентрализованным лейблам за прослушивание музыкальных активов.

Работу Soundchain определяет множество смарт-контрактов и интерфейсов взаимодействия, рассмотрим смарт-контракт PayPerPlay через который:

- учитывается каждое прослушивание музыкального актива
- генерируется список выплат роялти из выделенного объема Musereum Foundation

Let n is a moment of time and

Let r is a money available to pay-per-play for

Set of plays a , where a_i is a tuple:

$$a = \{a_i \mid \forall i \in a, a_i \equiv \mathcal{O}[listener, asset, times, sign]\} \tag{1.61}$$

$$b = \{b_i \mid \forall i \in a, b_i = f_{pay}(a_i), b_i \in (0; 1)\} \tag{1.62}$$

$$c = \sum_{i=0}^{|b|} b_i, \quad d = \frac{r}{c} \tag{1.63}$$

Finaly p is a set of payments:

$$p = \{p_i \mid \forall i \in b, p_i = d * b_i\}, \quad r = \sum_{i=0}^{|p|} p_i \tag{1.64}$$

Остается определить, что такое функция f_{pay}. Результат функции зависит от двух переменных: коэффициента слушателя и коэффициента лейбла:

$$f_{pay}(a) = min(f_{listenerCoef}(a)), \tag{1.65}$$

$$f_{labelCoef}(\delta^R_{labels}(a_{asset})) \tag{1.66}$$

Коэффициет слушателя определяет насколько платформа доверяет данному слушателю:

1. Персонификация учетной записи: ФИО, География;
2. Подтверждение телефонного номера (Proof-of-SMS);
3. Подтверждение географического происхождения: верификация документов;
4. Размер материальной заинтересованности в будущем платформы: количество ETM, количество и объем удерживаеных токенов децентрализованных ограницаций.

Коэфициент лейбла учитвает в подсчете:

1. Совокупный доход с продажи лицензий ассоциированного актива;
2. Доход с продажи лицензий за последнюю неделю;
3. Искусственный коэфициент оценки лейбла SoundChain'ом (дополнительно депонирование средств, сжигание средств и т.д.)

(1.67)