Техническое описание программного обеспечения “Блокчейн-платформа IZZZIO версия 1.2.0”

ООО “Изио”

Москва, 2020

Содержание документа:

1. Введение

Данный документ содержит техническое описание программного комплекса “Блокчейн-платформа IZZZIO версия 1.2.0”.

2. Наименование и назначение программного обеспечения

Блокчейн-платформа IZZZIO версия 1.2.0 (далее – блокчейн-платформа IZZZIO, Платформа IZZZIO, Система, ПО) - это программный комплекс, предназначенный для хранения, передачи и обработки любых данных, а также, для автоматизации бизнес-процессов с помощью применения технологии распределенного реестра.

3. Характеристика программного обеспечения

Платформа IZZZIO построена на базе технологии распределенного реестра.
Использование платформы обеспечивает согласованность и доступность данных, а также устойчивость к злонамеренному изменению данных (навязыванию ложных данных).

3.1 Архитектура блокчейн-платформы IZZZIO

Основной механизм хранения информации в блокчейн-платформе IZZZIO - ее хранение в виде направленной цепи блоков данных со следующими свойствами:

- каждый последующий блок криптографически зацеплен с предыдущим блоком;
- цепь распределенного реестра не может иметь ветвлений.

Участники сети обмениваются информацией, генерируя новые блоки и добавляя их в конец цепи или получая информацию из хранящихся ранее сгенерированных блоков. Каждый участник имеет собственную асимметричную ключевую пару, публичный ключ которой служит идентификатором (адресом) участника в системе.

Обработка информации в Системе происходит в сети, представляющей собой совокупность узлов, взаимодействующих между собой.

На каждом узле сети хранится полная копия всех блоков цепи. Распределенное хранение блоков в совокупности с вышеприведенными свойствами цепи обеспечивает согласованность и

доступность данных, устойчивость к злонамеренному изменению данных (навязыванию ложных данных).

Доступ участников к сети происходит с использованием сторонних приложений-клиентов, взаимодействующих с внешним API узла сети (см. п. 3.9).

Дистрибутив платформы IZZZIO включает:

Компоненты разрабатываемые ООО "Изио"
Программное обеспечение узла распределенного реестра
Подключаемый модуль базовой криптографии SHA256 + ECDSA-SHA2
Подключаемый модуль организации шифрованных соединений по протоколу Диффи-Хеллмана

Дополнительные подключаемые модули не входящие в дистрибутив:

Компоненты разрабатываемые ООО "Изио"
Подключаемый модуль криптографии на основе bitcore-lib
Подключаемый модуль ГОСТ криптографии на основе WebCrypto GOST Library
Подключаемый модуль ГОСТ криптографии на основе КриптоПро CSP 4.0
Подключаемый модуль ГОСТ криптографии на основе VipNet CSP 4.2

При генерации блока участник помещает в блок данные – транзакцию (блок содержит строго одну транзакцию).

Блок данных содержит следующую информацию:

- хэш текущего блока;
- хэш предыдущего блока;
- порядковый номер блока;
- дата и время начала генерации блока (UTC);
- дата и время окончания генерации блока (UTC);
- данные блока (одна транзакция).

Формат представления и хранения данных блока - JSON. Для хранения данных блока на узле ПО используется хранилище типа ключ-значение LevelDB, с порядковым номером блока в качестве ключа.

Так как узлы сети одновременно работают над генерацией новых блоков для добавления их в цепь, при синхронизации данных между узлами возникают конфликты. Для определения валидности блока (проверки блока) и разрешения конфликтов используется набор математических правил - алгоритм консенсуса.

Подробное описание алгоритмов консенсуса приведено в пункте 3.6.

Платформа IZZZIO поддерживает создание Тьюринг-полных смарт-контрактов.

Смарт-контракт - программный код, помещаемый в блок цепи и предназначенный для обработки информации (формирования, контроля, предоставления) единым (синхронным) образом на всех узлах сети Подробное описание смарт-контрактов приведено в пункте 2.6.

Тело транзакции содержит строковый идентификатор типа транзакции, данные транзакции (см. ниже), электронную подпись полного тела транзакции, выработанную с использование приватного ключа участника, создавшего транзакцию.

Блокчейн-платформой IZZZIO распознаются следующие идентификаторы типы транзакций:

- EcmaContractDeployBlock - транзакция с кодом смарт-контракта (размещение смарт-контракта в распределенном реестре);
- EcmaContractCallBlock - транзакция вызова метода смарт-контракта (запуск на исполнение метода смарт-контракта с заданными аргументами);
- прочие идентификаторы - транзакция данных (запись участником в распределенный реестр произвольной пользовательской информации).

Структурная схема узла сети приведена на рисунке 1.

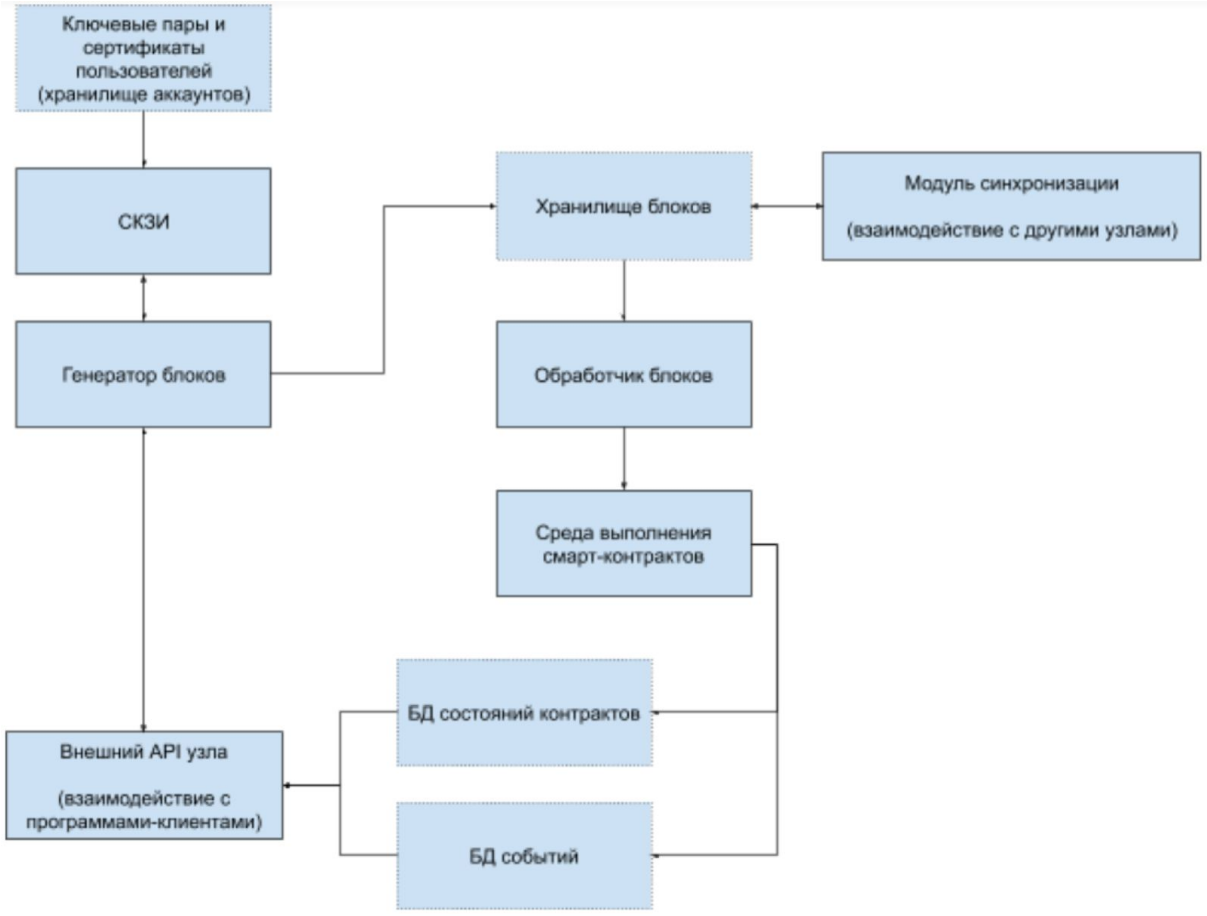

Рис. 1 Структурная схема узла сети

Исходные тексты ПО узла распределенного реестра блокчейн-платформы IZZZIO написаны с использованием языка программирования JavaScript.

3.2 Требования к программной и аппаратной платформам

ПО узла распределенного реестра блокчейн-платформы IZZZIO функционирует на (П)ЭВМ аппаратной платформы x86 или x86_64, работающей под управлением одной из следующих ОС:

- Windows 7/8/8.1/ 10 (x86, x64);
- Windows Server 2008 R2/2012/2012 R2 (x64);
- CentOS 6/7 (x86, x64);
- ТД ОС АИС ФССП России (GosLinux) (x86, x64);
- Red OS (x86, x64);
- Fedora 19/20 (x86, x64);
- Mandriva Enterprise Server 5, Business Server 1 (x86, x64);
- Oracle Linux 6/7 (x86, x64);
- OpenSUSE 13.2, Leap 42 (x86, x64);
- SUSE Linux Enterprise Server 11SP4/12, Desktop 12 (x86, x64);

- Red Hat Enterprise Linux 6/7 (x86, x64);
- Ubuntu 14.04/14.10 (x86, x64);
- Linux Mint 15/16/17 (x86, x64);
- Debian 8 (x86, x64);
- Astra Linux Special Edition (x86-64)
- ALT Linux 7 (x86, x64);
- ROSA 2011, Enterprise Desktop X.1 (Marathon), Enterprise Linux Server (x86, x64);
- РОСА ХРОМ/КОБАЛЬТ/НИКЕЛЬ (x86, x64);
- FreeBSD 9, pfSense 2.x (x86, x64);
- Mac OS X (Darwin) от версии 10.6.

Для работы ПО узла распределенного реестра требуется интерпретатор JavaScript Node.js версии 8 или 10.

Для хранения данных используются библиотеки (и модули Node.js доступа к ним):
- LevelDB - хранилище типа ключ-значение;
- sqlite3 - встраиваемая СУБД реляционной модели.

Минимальные аппаратные требования к аппаратной части (П)ЭВМ для работы ПО узла распределенного реестра:
- Центральный процессор - тактовая частота 1.5 ГГц;
- ОЗУ - 512 Мб.
- Свободное пространство на жестком диске: 1 Гб для размещения программных компонентов, дополнительное пространство для размещения обрабатываемых данных (хранилища блоков цепи, локальных данных смарт-контрактов, журнала событий).

3.3 Описание конфигурации блокчейн-платформы IZZZIO версия 1.2.0

Платформа предоставляет пользователям большое количество настраиваемых возможностей. Конфигурация узлов выполняется созданием файла config.json в директории, из которой происходит запуск узла IZZZIO или с помощью явного указания пути конфигурационного файла с помощью параметра запуска --config. Файл конфигурации представляется в формате JSON.

Возможные параметры конфигурационного файла в Системе:

p2pPort: 6013,	Порт p2p
httpServer: 'localhost',	Адрес привязки RPC/API и интерфейса
rpcPassword: '',	Пароль для подключения к RPC/API и интерфейсу (HTTP Basic Auth)

initialPeers: []	Массив со списком стартовых узлов
allowMultipleConnectionsFromIp: true,	Разрешить множественные p2p подключения с одного адреса. Рекомендуемые значения: False - если в сети много зацикливаний, True - если используется прокси для коннекта
maxPeers: 80,	Максимальное количество p2p подключений

upnp: {
enabled: true,
token: 'iz3node'
}
	Конфигурация автоматического поиска узлов с помощью DNS запроса
	Включить автоматическое обнаружение узлов в сети
	Токен по которому узел будет искать другие узлы

networkPassword: '',	"пароль" доступа к сети. Используется для создания приватных сетей
blockAcceptCount: 20,	Количеств блоков подтверждения транзакции (LC)
hearbeatInterval: 10000,	Внутренний таймер узла. Влияет на скорость повторных попыток подключений, сохранения состояний на диск и т.д.
peerExchangeInterval: 5000,	Частота обмена списками пиров
maxBlockSend: 600,	Максимальное количество отправляемых блоков. Должно быть больше blockQualityCheck (LC)
blockQualityCheck: 100,	Количество блоков "сверх", которые запрашиваются для проверки валидности цепочки (LC)
limitedConfidenceBlockZone: 288,	Зона "доверия" (LC). Цепочку ранее этой зоны менять нельзя. Должно быть больше blockQualityCheck
generateEmptyBlockDelay: 300 * 1000,	Частота создания пустых блоков при простое сети. false - для отключения

blockHashFilter: {
blockEndls: []
}
	Фильтр корректных блоков для LCPoA

	4 символа в конце блока. Сюда должен попасть совпадающий с Genesis Block хеш
genesisTiemstamp: 1492004951 * 1000,	Таймстамп 0 блока сети в миллисекундах
lcpoaVariantTime: 1,	Количество миллисекунд, требуемое на генерацию одного хеша блока (LCPoA)
validators: []	Плагины дополнительных консенсусов. ВАЖНО: Передача блоков плагинам консенсусов идёт в том-же порядке, в котором они указаны в этом параметре.
emptyBlockInterval: 10000,	Интервал проверки необходимости выпуска пустого блока
maxTransactionAtempts: 5,	Сколько попыток добавить блок предпринимается до решения отмены добавления блока в сеть
checkExternalConnectionData: false,	Проверять внешние данные на соответствие (конфигурации других узлов)
enableMessaging: false,	Разрешить использование шины сообщений (необходима для некоторых консенсусов)
recieverAddress: '123',	Внутрисетевой адрес узла сети. Используется в шине сообщений, а также протоколе StarWave
messagingMaxTTL: 3,	Максимальный предел скачков сообщения
maximumInputSize: 2 * 1024 * 1024,	Максимальный объем обрабатываемых данных от внешних узлов. Рекомендуется не больше 15 мб
allowMultipleSocketsOnBus: false,	Разрешить на подключение сокетов с разными адресами на один адрес узла
walletFile: './wallet.json',	Адрес файла кошелька
workDir: '.',	Рабочая директория
blocksDB: 'blocks',	Как хранить БД блоков: false - для хранения в ОЗУ, mem://blocks.json для хранения в ОЗУ и

	записи на ПЗУ при выгрузке. Строка, для хранения на диске
blocksSavingInterval: 300000,	Интервал сохранения блоков. false = для отключения автосохранения, или количество миллисекунд
appEntry: false,	Точка входа в "приложение" DApp. False - если не требуется
startMessage: false,	Сообщение, которое выводится при запуске узла

ecmaContract: {
enabled: true,
allowDebugMessages: false,
contractInstanceCacheLifetime: 10000,
}
	Конфигурация смарт контрактов EcmaContracts
	Система обработки контрактов включена
	Разрешает вывод сообщений смарт контрактам (используется для отладки)
	Время жизни экземпляра виртуальной машины контракта

masterContract: 5	Главный контракт в системе. Должен поддерживать методы управления ресурсами
hashFunction: 'SHA256',	Функция, используемая для вычисления хэша. Регистрируется плагинами
signFunction: '',	Функция вычисления цифровой подписи и генерации паролей(пусто-по умолчанию), варианты: 'GOST' 'GOST256' 'NEWRSA'
keyLength: 2048,	Длина ключа (передается в некоторые из плагинов)
plugins: []	Массив со списком загружаемых плагинов (кроме плагинов консенсусов)

3.4 Взаимодействие узлов сети друг с другом

Механизм обнаружения узлом сети других узлов (peers) определяется в конфигурационных параметрах блока. В Системе поддерживаются 2 механизма:

- Явно заданный список сетевых адресов для подключения;
- Обнаружение других узлов с использованием стандартного механизма DNS Service Discovery. При его использовании узлы регистрируют себя (в качестве сервиса с неким фиксированным идентификатором на DNS-сервере), список других узлов может быть запрошен у DNS-сервера как список зарегистрированных сервисов с заданным идентификатором).

После подключения взаимодействие узлов друг с другом происходит с использованием протокола WebSocket (на нижнем транспортном уровне используется протокол TCP/IP).

На первом этапе производится передача имени узла, проверка совместимости конфигурационных параметров узлов (активные модули, условия проверки блоков, алгоритм консенсуса), проверка совпадения пароля сети (пароль также задается в блоке конфигурационных параметров узла, в открытом виде не передается).

Далее, в процессе работы, узлы взаимодействуют с целью синхронизации цепи блоков.

3.5. Создание новых блоков в сети

Создание нового блока выполняется модулями LCPoA и DLCPoA (см. пункт 3.6):

1. Модулю передаются данные блока, которые необходимо добавить в цепочку блоков
2. К данным блока добавляется startTimestamp - время начала генерации блока

Далее все действия выполняются итеративно до момента нахождение правильного хеша:

1. В timestamp блока устанавливается информация о текущем времени на устройстве в миллисекундах (GMT+0 с поправкой на отставание часов)
2. Выполняется генерация хеш-суммы из всех данных блока
3. Выполняется проверка на соответствие хеш суммы заданным фильтрам. При удачной проверке, генерация блока завершается, блок добавляется в цепочку.

3.6 Алгоритмы консенсуса

Платформа IZZZIO поддерживает два основных алгоритма консенсуса (фильтра): Limited Confidence Proof of Activity (LCPoA) и Dynamic LCPoA (DLCPoA).

3.6.1 LCPoA

LCPoA (Limited Confidence Proof-of-Activity, рус. Доказательство активности с ограниченным доверием) - гибридный алгоритм консенсуса сети блокчейн, состоящий из двух технических элементов:

- Proof-of-Activity (рус. “Доказательство активности”) - принцип, основанный на решении задачи, схожей с задачей принципа Proof-of-Work, но с пониженной сложностью (обычно решение задачи генерации блока занимает от долей секунды до нескольких минут). Повышение вычислительных затрат на генерацию блока уменьшает вероятность возникновения конфликтов и увеличивает сложность генерации цепочки блоков нарушителем для проведения атаки навязывания ложных данных.

- Limited Confidence (рус. "Ограничение доверия") - система автоматического создания “контрольных точек” в сети блокчейн, достигается за счет введения жесткого порога возможности перезаписи блоков. ПО узла при синхронизации может перезаписать не более limtedConfidenceBlockZone (параметр конфигурации узла) последних сохраненных в хранилище блоков. В случае расхождения большего числа блоков автоматическая синхронизация не производится. Узлы остаются в рассинхронизированном состоянии, разбор конфликта и проверка валидности данных производится администраторами сети/узлов.

Limited Confidence
Limited Confidence - это механизм создания контрольных точек в сети, ограничивающий возможность перезаписи блоков сети блокчейн дальше заданного лимита от последнего блока. Например:

Ситуация 1

В настройках сети примера число LC задано как 10
Клиент1 - содержит цепочку блоков от 0 до блока 30
Клиент2 - содержит цепочку блоков от 0 до 34, при этом у текущего клиента, блок 29 отличается от схожего блока Клиента1

В такой ситуации Клиент1 скачает кусок цепи Клиента2 от блока 29 до блока 34, т.е. блок 29 будет заменен, и цепочки обоих клиентов станут одинаковыми.

Ситуация 2

В настройках сети примера число LC задано как 10
Клиент - содержит цепочку блоков от 0 до блока 30
Атакующий - пересоздал цепочку блоков от 1 до 31, заменив все транзакции в сети

В такой ситуации Атакующий предложит
синхронизировать цепочку от 1 до 31 блока, однако минимальный блок, с которого можно синхронизировать: 31 - 10 = 21
Синхронизация с цепью атакующего в этом случае выполнена не будет.

Меньшее число LC снижает шанс перезаписи цепочки, бОльшее число снижает шанс появления не синхронизируемых (тупиковых) цепочек.

Proof of Activity

В качестве алгоритма генерации блоков используется модификация Proof of Work с пониженной сложностью. Особенность заключается в использовании в качестве nonce - Unix timestamp, а также в дополнительных проверках и ограничениях:

- В качестве общего времени используется пояс GMT+0
- В блок необходимо записывать информацию о времени старта генерации блока
- В timestamp блока записывается nonce блока

Проверяются следующие параметры:

- Совпадение хеша блока с фильтром разрешенных хешей
- Время старта генерации блока не больше текущего времени сети, и не больше timestamp блока
- timestamp не больше текущего времени сети, и не меньше начала времени генерации блока
- timestamp и время старта генерации блока не менее timestamp предыдущего блока

При этом разрешается делать допуск по каждой проверке не более 60 секунд.
Также возможно использование классического PoW или других алгоритмов совместно с механизмом LC.

Для алгоритма LCPoA в конфигурационных параметрах узла задается ограничение на значение хеш-функции блока (строковый фильтр, применяемый к шестнадцатеричному представлению значения хэш-функции).

3.6.2 DLCPoA

DLCPoA (Dynamic Limited Confidence Proof-of-Activity, рус. Динамическое доказательство активности с ограниченным доверием) - это алгоритм консенсуса, позволяющий регулировать скорость сети.

Алгоритм DLCPoA представляет собой разновидность алгоритма LCPoA (модифицированная версия) с динамическим управлением сложностью генерации блоков. В случае использования DLCPoA в конфигурационных параметрах узлов вместо фиксированного фильтра задается целевая скорость генерации блоков сетью, а параметры фильтра вычисляются автоматически.

Для генерации хеша нового блока в сети необходим подбор подходящего хеша, валидность которого определяется определенными фильтрами. В алгоритме LCPoA фильтры задаются в конфигурации, тогда как в DLCPoA фильтры рассчитываются автоматически, по формуле, исходя из скорости работы сети. Если скорость близка к целевой, то фильтр становится сложнее, если далека от целевой, то фильтр легче, вплоть до нулевой сложности, когда подходит любой хеш.

Формула, определяющая сложность сети:

abs(log10(1000 - расстояние мсек между блоками) / (1000/целевая скорость сети)

Формула, определяющая выбор хеша:

hashSigma <= round(допустимое количество вариантов хеша - (допустимое количество вариантов хеша * сложность)*

**hashSigma: последние 6 или более символов хеша в 16-ричном представлении, т.е. 6 байт*

Динамическая конфигурация сложности алгоритма позволяет предотвратить атаки на скорость работы сети, и поддерживать сеть работоспособной под любыми нагрузками.

3.6.3 Proof-of-Authority

Алгоритм PoA использует массив заранее созданных публичных ключей в качестве подтверждения корректности блока. Список ключей выпускается в сеть в течении первых 5 блоков. Все последующие транзакции, для успешного прохождения через этот алгоритм консенсуса, должны быть подписаны одним из выпущенных ключей.

3.7 Смарт-контракты

В основе блокчейн-сети IZZZIO лежит использование системы смарт-контрактов - алгоритмов, предназначенных для работы с данными в системе.

Основные особенности:

- JavaScript ES6. Введена поддержка работы с псевдослучайными числами (Math.random) и датой и временем (Date)
- События
- Взаимодействие с другими контрактами в сети
- Поддержка объектов больших чисел (BigNumber) для безопасных вычислений с плавающей запятой
- Встроенные итерируемые и не итерируемые структуры данных: KeyValue, BlockchainArraySafe, BlockchainArray, TokenRegister, BlockchainMap

Смарт-контракт - это программный код, помещаемый в блок цепи и предназначенный для обработки информации (формирования, контроля, предоставления) единым (синхронным) образом на всех узлах сети.

Методы смарт-контрактов - операции над объектами системы.

Объект системы - некоторая сущность в цифровом пространстве, обладающая определённым состоянием и поведением, имеющая определенные свойства (атрибуты) и операции над ними (методы).

EcmaContracts реализует среду управления и выполнения смарт-контрактов.

Для исполнения смарт-контрактов используется специальная изолированная среда исполнения EcmaContracts, использующая JavaScript-движок с открытым исходным кодом V8 для создания виртуализированного окружения (Isolated VM, см. пункт 3.7.3) исполнения кода контрактов. Это позволяет полностью изолировать произвольный код контрактов как от системы, так и от других контрактов, а также контролировать доступность ресурсов (ОЗУ, процессорное время, лимиты вызовов) для каждой сущности виртуальной машины.

EcmaContracts реализует принцип управления состоянием (state) сущностей контракта, при котором в цепочку блокчейн сохраняется только информация о вызовах методов смарт-контрактов и их аргументах. Результаты работы смарт контрактов сохраняются локально на узле пользователя, и загружаются при следующем вызове контракта. Все узлы в сети одновременно повторяют каждую транзакцию цепочки с самого первого блока, до последнего существующего, что приводит к полной синхронизации состояний контрактов на каждом из узлов.

Таким образом, EcmaContracts контролирует состояние, запущенные сущности виртуальных машин, потребление ресурсов, и обеспечивает среду выполнения контрактов необходимыми для работы методами и объектами. Взаимодействие с такими объектами позволяет реализовывать мощные и функциональные децентрализованные системы

В стандартную библиотеку JavaScript внесены следующие модификации:

- Функции получения текущих даты и времени возвращают время окончания генерации блока, содержащего вызов.
- Генератор случайных чисел инициализируется с использованием детерминированных данных, определяемых блоком, содержащим вызов.
- Добавлена возможность запуска других смарт-контрактов
- Добавлена поддержка работы с объектами больших чисел (тип BigNumber).
- Добавлены итерируемые и не итерируемые объекты, производящие запись и чтение данных в хранилище локальных данных смарт-контрактов.
- Поддержка записи данных в журнал событий.

В качестве хранилища локальных данных смарт-контрактов используется хранилище LevelDB (отличное от хранилища данных блоков цепи).

В качестве хранилища журнала событий используется база данных sqlite3. Получение данных из журнала событий возможно с использованием внешнего API узла сети (см. пункт 3.9).

Модуль работы со смарт-контрактами обрабатывает 2 типа транзакций:

- **EcmaContractDeployBlock** транзакция с кодом смарт-контракта (размещение смарт-контракта в распределенном реестре), содержит:

- ключ (адрес) отправителя;
- значение для инициализации генератора случайных чисел, подмешивается к стандартному значению;
- исходный код контракта;
- значение хэш-функции для кода контракта;
- дополнительная информация – произвольные данные, передаваемые пользователем и доступные коду смарт-контракта при исполнении через API среды исполнения.

- **EcmaContractCallBlock** блок вызова метода смарт-контракта (запуск на исполнение метода смарт-контракта с заданными аргументами), содержит:

- адрес контракта - порядковый номер блока типа EcmaContractDeployBlock, содержащего код контракта;
- значение для инициализации генератора случайных чисел, подмешивается к стандартному значению;
- имя метода;
- массив с значениями аргументов вызова;
- дополнительная информация – произвольные данные, передаваемые пользователем и доступные коду смарт-контракта при исполнении через API среды исполнения.

Смарт-контракт может иметь 2 служебных метода:

- init() - автоматически выполняется каждый раз при инициализации среды исполнения для исполнения любого другого вызываемого метода данного смарт-контракта, в том числе метода deploy();
- deploy() - автоматически выполняется при обработке EcmaContractDeployBlock (размещение смарт-контракта в распределенном реестре).

Системные ресурсы: объем ОЗУ, процессорное время, количество вызовов смарт-контракта в минуту, выделяемые для исполнения смарт-контракту определяются Мастер-контрактом.

Мастер-контракт – смарт-контракт, имеющий определенный набор интерфейсных методов, служащих для управления поведением Системы.

Номер блока, содержащего Мастер-контракт задается в конфигурационных параметрах узла сети и должен совпадать для всех узлов сети.

3.7.1 Алгоритм выполнения кода смарт-контракта

Выполнение кода смарт-контракта осуществляется по следующему алгоритму:

- Средой выполнения проверяется ЭП транзакций, содержащих вызов и код смарт-контракта;

- Запуск виртуальной среды, загрузка в нее предыдущего состояния контракта, инициализация генератора случайных чисел;
- Выполнение метода init();
- Выполнение вызываемого метода.

В случае, если выполнение контракта прошло без ошибок, обновляются локальное хранилище данных смарт-контракта, хранилище данных журнала событий.

В случае, если выполнение смарт-контракта завершилось с ошибкой, изменение локальных хранилищ не производится, в файлы журналов ПО узла распределенного реестра вносятся записи о возникших ошибках. При этом, если причина возникновения ошибок внешняя по отношению к коду и данным смарт-контракта, то состояние локальных хранилищ узла будет перейдет в рассинхронизированное состояние.

3.7.2 Работа смарт-контрактов в сети

При синхронизации каждый узел последовательно повторяет все действия с контрактами из цепочки блоков, и сохраняет измененные состояния локально. В результате локальные состояния на каждом из узлом полностью синхронизируются, что позволяет выполнять код децентрализованно и достоверно, и обеспечивает детерминированность результата работы.

3.7.3 Виртуальная среда (Isolated VM)

Для запуска кода контрактов используется специальная изолированная среда Isolated VM на базе JavaScript движка V8.
Каждой изолированной среде доступно ограниченное количество ресурсов (процессорное время, ОЗУ, количество вызовов) а также ограниченный функционал:

1) Контракты не могут использовать системные методы, а также методы среды выполнения напрямую
2) Сохранение данных между запусками может выполнятся только с помощью встроенного типа KeyValue и его производных BlockchainArraySafe, BlockchainArray, TokenRegister, BlockchainMap
3) Контрактам недоступны методы асинхронного выполнения кода, Promise, setTimeout, setInterval, nextTick
4) Код контракта работает полностью синхронно
5) Превышение разрешенных ресурсов моментально завершает выполнение контракта с ошибкой
6) Методы работы с датами заменены. В качестве текущей даты используется время текущего блока
7) Метод работы со случайными числами заменён. Генерация случайных чисел зависит от текущего состояния контракта
8) В коде контракта недоступно использование конструктора. Вместо конструктора используется метод init и deploy
9) Все контракты должны наследоваться от класса Contract или его производных

Также контрактам доступны дополнительные функции:

1) Встроенный тип BigNumber для безопасной работы с большими числами
2) Класс TokensRegister реализующий функционал токена (монеты)
3) Контракт TokenContract реализующий функционал токена (монеты)
4) ContractConnector класс для взаимодействия с другими контрактами в системе
5) TokenContractConnector класс для взаимодействия с другими контрактами в системе, которые наследуются от ContractConnector
6) SellerContractConnector класс для взаимодействия с контрактами-продавцами
7) Require - альтернативный способ взаимодействия с другими контрактами в системе
8) BlockchainArray - тип данных, схожий с массивом (Array), использующий объект KeyValue для хранения данных между запусками
9) BlockchainMap - тип данных, реализующий представление Map, использующий объект KeyValue для хранения данных между запусками
10) TypedKeyValue - аналог KeyValue, сохраняющий исходные типы значений
11) Event - класс для генерации событий
12) assert - встроенный класс, который позволяет выполнять проверки критически важных элементов

3.8 Токены

Токен – цифровой актив, обращающийся в распределенной сети. В блокчейн-платформе IZZZIO токены реализуются на основе смарт-контрактов. В стандартную библиотеку среды исполнения смарт-контрактов включены объекты, реализующие работу с токенами. Возможно как непосредственное использование этих токенов, так и создание с их использованием смарт-контрактов с расширенным функционалом, реализующих функционал токенов.

3.9 Внешний API узла распределенного реестра

Программное обеспечение узла распределенного реестра предоставляет внешний интерфейс программирования приложений (API). Внешний API может быть использован для подключения сторонних приложений-клиентов к узлу сети.

Внешний API имеет архитектуру REST, в качестве транспортного протокола используется HTTP, представление данных - JSON.

Внешний API узла содержит вызовы, имеющие следующую функциональность:

- получение информации об узле распределенного реестра: перечень активных функций узла, конфигурационные параметры, версия программного обеспечения;
- получение информации о других узлах сети;
- получение информации из журнала событий;
- получение информации о блоках распределенного реестра;
- получение информации о смарт-контрактах и результатах их выполнения;

- создание новых транзакций/блоков.