Санкт-Петербургский государственный университет

Кафедра системного программирования

Фефелов Алексей Андреевич

Разработка прототипа блокчейн-системы с динамически задаваемыми ограничениями на смарт-контракты

Курсовая работа

Научный руководитель:
ст. преп. Кириленко Я. А.

Консультант:
к. ф.-м. н. Березун Д. А.

Санкт-Петербург
2019

Оглавление

Введение

Блокчейн (Blockchain) – это способ распределённого хранения данных, появившийся в 2008 году. Данные представляются в виде блоков транзакций, которые сформированы в цепь и, как правило, связаны между собой посредством хеширования — каждый блок хранит хеш предыдущего. В вычислении хеша блока участвуют все транзакции, записанные в нем. Это позволяет практически гарантировать, что изменение любой транзакции в блоке приведет к изменению хеш-кода блока, а значит, практически невозможно подделать данные, которые хранятся в нем. Данные блокчейна хранятся одновременно у всех участников сети (узлов).

Блокчейн-технология обрела популярность в связи с тем, что она обеспечивает неизменяемость блоков транзакций и возможность проверки подлинности текущего состояния системы любым узлом. В частности, это дает возможность избавления от централизованного агента, регулирующего все транзакции, а любой участник сети может в любой момент времени лично проверить всю историю, начиная с момента создания блокчейна, повышая тем самым безопасность и минимизируя необходимость доверия третьим лицам.

Блокчейн-технология очень быстро привлекла к себе внимание и встал вопрос о границах ее применимости: в данный момент не до конца понятно, где и как ее можно применить. Сейчас в большинстве случаев блокчейн используется для создания криптовалют и распределенных и неизменяемых баз данных.

С помощью данной технологии получилось создать ряд вполне успешных финансовых систем, таких как Bitcoin, Ethereum и др., но до сих пор нет такой системы, в которой можно было бы накладывать дополнительные ограничения на сделки и контракты. Это могло бы быть полезно при реализации финансовой системы, ограничения для которой задаются третьими лицами (контролирующими органами). Это может быть как реальная финансовая система в рамках государства, где ограничения — это законы, так и виртуальная — в компьютерных играх с

развитой экономикой.

Таким образом актуальной становится задача разработки прототипа блокчейн-системы, позволяющей динамически накладывать дополнительные ограничения на смарт-контракты и автоматизировать действия, требующие непременного исполнения.

1. Постановка задачи

Целью данной работы является разработка прототипа блокчейн-системы, ограничения на выполнение смарт-контрактов в которой могут динамически задаваться третьими лицами.

Для ее достижения были поставлены следующие задачи:

- сделать обзор популярных блокчейн-систем и фреймворков;
- оценить возможность реализации данной системы с использованием существующих фреймворков;
- разработать архитектуру системы;
- реализовать прототип системы;
- провести апробацию.

2. Обзор

К данному моменту уже появилось огромное количество блокчейн-систем. Были рассмотрены следующие системы: Bitcoin, Litecoin, Bitcoin Cash, Dash, Peercoin, Ethereum, R3 Corda, Cardano, Ripple, Stellar, Hyperledger Iroha, Hyperledger Fabric, IOTA, EOS, Cosmos, Tezos, Slimcoin. Написана и опубликована обзорная статья [9].

Существует два принципиально разных типа блокчейн-систем: *публичные* и *приватные*.

Система называется публичной, если любой желающий может присоединиться к ней, прослушивать сеть, принимать участие в валидации транзакций, вносить изменения в систему — создавать транзакции и блоки и публиковать их в сеть. Все остальные системы называются приватными.

Смарт-контракт — это программируемый объект, который работает поверх блокчейна. Смарт-контракт содержит набор правил и условий, на которых участники договариваются. Если все правила и условия соблюдены, то смарт-контракты автоматически выполняются после запроса или транзакции. Невозможно повлиять на их исполнение или изменить эти условия после того, как контракт опубликован в сети. Смарт контракты хранятся в блокчейне и могут хранить некоторые дополнительные данные, необходимые для логики исполнения.

Первые смарт-контракты появились в блокчейне Bitcoin [8]. Там их можно было написать на примитивном, не тьюринг-полным скриптовом языке Bitcoin-Script. Полноценные контракты на нем написать практически невозможно. Первые тьюринг-полные смарт-контракты появились в Ethereum [10]. Обычно смарт контракты в Ethereum реализуются в Solidity или другом языке программирования высокого уровня, который компилируется в виртуальную машину Ethereum — (*Ethereum Virtual Machine*).

Ethereum Virtual Machine — EVM, является виртуальной машиной, изолированной от внешнего мира, в пределах которой выполняются все транзакции.

Написание смарт-контрактов сопряжено с множеством ошибок, которые могут стоить огромных денег [1, 2]. Поэтому продолжают появляться фреймворки для верификации смарт-контрактов [6, 5].

Смарт-контракты сейчас поддержаны во многих системах [3], но ни в одной из них нельзя накладывать дополнительные ограничения на выполнение смарт-контрактов. В частности, поэтому мы решили создать прототип системы, которая позволять это сделать. Это нужно для того, чтобы была возможность регулировать их выполнение. Например, при построении финансовой системы ограничениями будут являться законы и нормативные акты и каждая транзакция будет проверяться на соответствие им. Основная идея состоит в том, чтобы пользователи могли написать любой контракт на каком-либо языке (в данном случае — C#), но изменять состояние блокчейна они могли только используя заданный набор механизмов (действий), которые могут быть изменены в любой момент времени. Именно это мы считаем динамически задаваемыми ограничениями, упоминание которых было во введении.

3. Архитектура системы

В данной главе будет описана архитектура системы, рассмотрены ее компоненты.

3.1. Описание компонент системы

Для решения поставленной задачи было решено реализовать две блокчейн-системы, которые взаимодействуют друг с другом. Изначально планировалось хранение в первой системе набора предикатов, которые, принимая на вход произвольную транзакцию (контракт), говорили бы, является ли данная транзакция или контракт корректными, но позже стало понятно, что лучше формализовывать сделки, то есть задать набор действий, которые являются корректными в каком-то смысле. У каждого действия есть 0 или более зависимостей, которые должны выполниться в какой-то момент времени. Например, при переводе денег нужно заплатить налог: действие — перевод, зависимость — оплата налога, которая исполняется в момент перевода. Пользователи же, используя этот набор действий, могут создать свои контракты, которые являются либо примитивным действием, либо их комбинацией. Список аккаунтов пользователей, вся информация про них, а также активные контракты хранятся во втором блокчейне. В дальнейшем, первую систему будем называть RuleChain, а вторую UserChain. В системе есть 3 типа узлов: NotaryNode, Government и OrderingService.

Узлы первого типа (NotaryNode) хранят в себе обе блокчейна, а так же их актуальное состояние. Их задача – валидировать входящие блоки (не транзакции) и поддерживать работоспособность обоих систем.

Второй тип узлов (Government) имеет права на изменения состояния RuleChain. Он может добавлять и удалять новые элементарные действия и зависимости к ним. Предполагается, что доступ к этим серверам имеет только ограниченный круг лиц, поскольку имея доступ к нему очевидным образом можно легко нарушить инварианты системы.

Узел последнего типа (OrderingService) должен быть один. Он сделан для того, чтобы принимать в себя все входящие транзакции пользо-

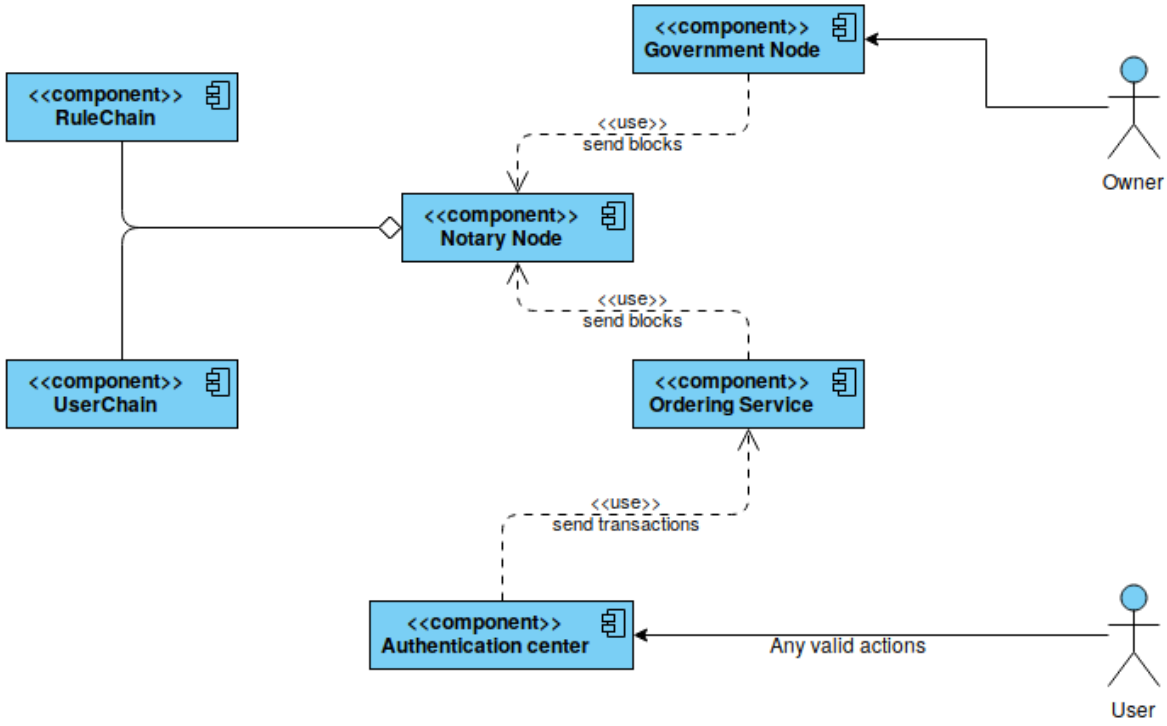

Рис. 1: Диаграмма компонент

вателей, предварительно их валидировать, формировать блок, подписывать и рассылать его всем узлам первого типа для принятия данного блока в цепь. Запросы в него поступают не напрямую от пользователей, а от центра идентификации, который получает запросы пользователей и формирует валидные транзакции. На первый взгляд кажется, что добавлением этого сервиса мы делаем систему централизованной, но на самом деле он отвечает только за формирование блока. Решение о принятии блока в цепь производится децентрализованно узлами первого типа. Кроме того, можно обобщить эту модель и сделать несколько таких сервисов, но тогда придется ещё дополнительно синхронизировать их друг с другом для того, чтобы во-первых, у них были одинаковые транзакции в пуле транзакций, во-вторых, чтобы в каждый момент времени не более, чем один OrderingService формировал блок и рассылал его остальным, в противном случае будут противоречия. Диаграмма

компонент представлена на Рис. 1.

Взаимодействие узлов друг с другом осуществляется посредством http протокола.

3.2. Архитектура RuleChain

Данный блокчейн, как уже было упомянуто выше, хранит в себе список допустимых действий (Actions), из которых впоследствии пользователи формируют контракты, а так же список зависимостей для каждого действия, которые должны выполниться в момент исполнения действия. Действия являются элементарными с точки зрения системы и должны быть исполнены за один шаг. В этом блокчейне обрабатываются следующие типы транзакций:

- добавление действия;
- удаление действия;
- добавление зависимости;
- удаление зависимости.

В состоянии блокчейна хранится список допустимых действий и список зависимостей к каждому из них.

3.3. Архитектура UserChain

Вся основная информация хранится именно в этом блокчейне. Он хранит список зарегистрированных в системе аккаунтов, всю информацию о них, список активных контрактов и их текущее состояние.

Валидацией занимаются доверенные узлы (NotaryNode). Пользователи взаимодействуют с блокчейном только через центры идентификации, которые принимают запросы от пользователей, устанавливают их личность, формируют транзакции и отправляют их на OrderingService, который, в свою очередь, упорядочивает транзакции в порядке поступления и отправляет их по порядку в пул транзакций. Раз в n секунд

(в нашем прототипе n = 10) OrderingService получает все транзакции из пула, проверяет их корректность и формирует блок. После этого он отправляет блок доверенным узлам, который убеждаются в том, что сформированный блок корректен и встраивают его в цепь.

Контракты отправляются в виде исходного кода на C#, компиляция и исполнение происходят на сервере. Подробнее см. раздел смарт-контракты (4.1).

В этом блокчейне обрабатываются следующие типы транзакций:

- публикация контракта в сеть (deploy);
- вызов контракта.

В состоянии блокчейна хранится список зарегистрированных аккаунтов и список активных контрактов, представленных в виде динамических объектов.

4. Детали реализации

Было решено реализовывать прототип с нуля самостоятельно, так как переиспользовать существующие системы не представляется возможным: на момент начала разработки не существовало приватной системы с поддержкой смарт-контрактов, в которой можно было бы организовать взаимодействие блокчейнов друг с другом. В марте 2019 года появился Cosmos [4] и, вероятно, данную систему можно было бы реализовать, переиспользуя их наработки.

Данный прототип может быть реализован практически на любом языке программирования — требуется возможность разработки веб-сервера и примитивные типы данных. Для реализации прототипа был выбран фреймворк ASP.NET Core, так как он обладает всей необходимой функциональностью для реализации такой системы.

4.1. Смарт-контракты

В настоящий момент смарт-контракты пишутся пользователями на языке C#. Для написания и отладки контрактов пользователям предоставляется сборка со всеми необходимыми интерфейсами и перечислимыми типами данных, а именно: вся информация о типах аккаунтов и существующих элементарных действиях (Actions). Как только в RuleChain список активных действий обновляется, пользователям предоставляется новая сборка с изменениями. Каждый контракт принимает на вход Executer, который принимает на вход тип действия (ActionType), запрашивает его актуальное состояние у RuleChain и исполняет. Предполагается, что действия добавляются и удаляются достаточно редко, но изменяться они могут достаточно часто.

Контракты отправляются в виде исходного кода. Для того, чтобы была возможность исполнять их, были рассмотрены следующие варианты:

- подключить виртуальную машину NEO-VM [7];
- написать свой язык и интерпретатор к нему;

- динамически компилировать исходный код в dll библиотеку, после этого исполнять.

Первая задача оказалась неоправданно сложной в реализации. Создание языка, а также компилятора (интерпретатора) к нему является одним из возможных направлений дальнейшей работы. В данный момент было решено взять готовую технологию для динамической компиляции исходного кода и использовать ее.

Первой попыткой было использование стандартного генератора кода, поддерживаемого компанией Microsoft:
Microsoft.CSharp.CSharpCodeGenerator, однако после реализации с ее использованием оказалось, что она не поддерживается на операционной системе линукс (хотя в документации утверждается обратное).

Тогда было решено использовать более новую технологию для компиляции исходного кода — Roslyn. С помощью нее код компилируется в in-memory dll библиотеку. После этого с помощью механизмов рефлексии получаем из нее класс Contract, проверяем, что он содержит метод Execute и, если до этого момента не произошло никаких ошибок, сохраняем данный контракт в состоянии блокчейна.

Для того, чтобы система работала корректно, на контракты накладываются дополнительные ограничения: класс контракта должен называться Contract и содержать метод Execute с динамичкским числом параметров (это нужно, чтобы мы легко могли получить его из сборки механизмами рефлексии), при написании можно использовать только предоставленную сборку с моделями, а так же несколько наперед заданных системных библиотек. Сейчас это System и System.Math, но этот список в любой момент можно изменить. Кроме того запрещено использовать unsafe-код. Это сделано в целях безопасности: мы лишаем пользователей огромного количества возможностей для нарушения работоспособности нашей системы. Сейчас актуальным является вопрос: что ещё необходимо предпринять для того, чтобы можно было считать эту систему безопасной?

После компиляции используя механизмы рефлексии получаем из сборки класс Contract. Далее, создается его экземпляр и сохраняется

в список активных контрактов. Теперь у пользователей есть возможность вызывать этот контракт. Для вызова контракта необходимо инициировать соответствующую транзакцию. У каждого контракта есть предикат, который показывает состояние контракта (активен или нет). Он нужен для того, чтобы не хранить в состоянии системы контракты, которые больше никогда не будут вызывать. Как только этот предикат становится ложным, контракт удаляется из состояния системы. Пример контракта представлен на Рис. 2

```
public class Contract
{
    public bool IsActive { get; private set; } = true;
    public AccountAddress Owner { get; }

    public Contract()
    {
        var bytes = new byte[16] {1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1};
        var guid = new Guid(bytes);
        Owner = new AccountAddress(guid);
    }

    public void Execute(IActionExecutor executor, AccountAddress sender, params object[] paramsList)
    {
        if (!sender.Equals(Owner))
        {
            return;
        }
        var bytes2 = new byte[16] {2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2};
        var guid2 = new Guid(bytes2);
        var receiver = new AccountAddress(guid2);
        int amount = 100;
        object[] @params = {receiver, amount};
        executor.Execute(ActionType.TransferMoney, sender, @params);
        IsActive = false;
    }
}
```

Рис. 2: Смарт-контракт (перевод)

В нем есть один метод (Execute), который может быть вызван только один раз. После его вызова произойдет перевод денег с помощью вызова метода executor.Execute. Налог спишется автоматически. После того, как этот контракт исполнится, он будет удален из состояния.

4.2. Допущения и ограничения

Поскольку целью работы было создание прототипа, было сделано несколько допущений.

Во-первых, не уделяется никакого внимания безопасности: данные передаются в незашированном виде по сети, вся информация хранится в открытом виде.

Во-вторых, не реализовыван центр идентификации, поскольку пользователей у этой системы в данный момент нет, а для тестирования это не нужно.

Кроме того, как такового алгоритма консенсуса в данный момент нет, но он может быть легко добавлен в систему. Сейчас до конца непонятно, какой алгоритм здесь будет наиболее подходящим.

Наконец, никак не контролируется подлинность отправляемых сообщений. Предполагается, что центр идентификации является достаточно надежным, чтобы ему можно было доверить формирование транзакций пользователей.

Все эти ограничения носят чисто технический характер и могут быть убраны в случае необходимости.

5. Апробация

В данный момент в системе поддержаны только денежные отношения, поэтому сильно сложные контракты в ней просто не придумать.

При апробации были смоделированны сценарии, в которых должна работать данная система. Для тестирования системы было написано 2 контракта: перевод денег между людьми, а также кредит, выданный какой-либо организацией.

Цель апробации — проверить возможность исполнения и корректность работы смарт-контрактов, добавление и изменение действий (Actions) и ограничений.

В первом случае все просто: один пользователь переводит деньги другому и платит налог фиксированного размера, который действителен в момент заключания (а не исполнения) контракта. Для этого создается элементарное действие “перевод”, зависимость ”уплата налога” и сам контракт. Контракт исполняется сразу после публикации. Во время исполнения он производит перевод денег на счет, указанный в контракте. Налог взимается автоматически.

Было создано и исполнено 10 таких контрактов, причем процентная ставка постоянно менялась и каждый раз с пользователей взимался налог актуального размера.

Во втором случае все чуть чуть сложнее. Организация, например, банк, создает контракт ”кредит” и прописывает в него все условия (кому, сколько, под какой процент, ...). Для написания такого контракта можно использовать либо действие ”кредит” и тогда налог оплатится автоматически после полного погашения кредита, либо использовать простые переводы со счета организации на счет клиента и со счета клиента на счет организации. Налог в этом случае придется заплатить самостоятельно (тоже после полного погашения кредита). Контракт делается таким образом, чтобы, подписав его, клиент получил деньги на свой счет сразу, а после этого ежемесячно (ежеминутно для тестирования), банк вызывал данный контракт и списывал со счета клиента ежемесячный (ежеминутный) платеж.

Для добавления более содержательных контрактов, необходимо добавить в систему аналог цифрового имущества. Это является одним из дальнейших направлений работы. После этого можно будет создавать более интересные и содержательные сделки, например встречные сделки купли-продажи с кредитованием, аукцион и т.д.

Таким образом, можно сделать вывод, что система работает и соответствует всем требованиям: пользователи могут создавать и исполнять в ней свои контракты, а ограничения на них задаются и меняются динамически, никак не мешая работе системы.

6. Направления дальшейшей работы

Во-первых, интерес представляет разработка собственного безопасного языка для написания смарт-контрактов и виртуальной машины. Это представляет огромный интерес, поскольку написание смарт-контрактов сопряжено с ошибками, допускаемыми людьми при их программировании. Поэтому если бы мы умели формально верифицировать все написанные контракты, мы могли бы намного сильнее им доверять. В этом направлении уже была проделана часть работы, но на защиту она не выносится, так как никаких существенных результатов получено не было.

Во-вторых, планируется разработка визуального языка программирования, на котором можно будет написать контракт любой сложности, скомпилировать его и опубликовать в сети. Благодаря этому языку любой человек, далекий от программирования, сможет самостоятельно создать контракт, а если ещё и получится верифицировать целевой язык, то человек может быть уверен в том, что написанный им контракт корректен.

Кроме того, планируется добавить в систему шифрование, безопасность и цифровое имущество для того, чтобы эта система была более реалистичным прототипом.

Наконец, хочется узнать, насколько такая система может быть полезна в игровой индустрии и в случае, если она окажется действительно полезной, то, возможно, начать двигаться в эту сторону.

Заключение

В ходе работы были достигнуты следующие результаты:

- сделан и опубликован обзор популярных блокчейн-систем и фреймворков;
- разработана архитектура системы;
- реализован прототип системы;
- проведена апробация.

Таким образом, поставленные задачи были полностью выполнены.

В результате создан прототип, который будет использован при проведении различных исследований.

Список литературы

[1] 300m in cryptocurrency accidentally lost forever due to bug. — URL: https://www.theguardian.com/technology/2017/nov/08/cryptocurrency-300m-dollars-stolen-bug-ether (online; accessed: 2018-12-14).

[2] A 50 million hack just showed that the dao was all too human. — URL: https://www.wired.com/2016/06/50-million-hack-just-showed-dao-human/ (online; accessed: 2018-12-14).

[3] Bartoletti Massimo, Pompianu Livio. An Empirical Analysis of Smart Contracts: Platforms, Applications, and Design Patterns // Financial Cryptography Workshops. — Vol. 10323 of Lecture Notes in Computer Science. — Springer, 2017. — P. 494–509.

[4] Cosmos white papeer. — URL: https://github.com/cosmos/cosmos/blob/master/WHITEPAPER.md.

[5] Formal Verification of Smart Contracts: Short Paper / Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet et al. // PLAS@CCS. — ACM, 2016. — P. 91–96.

[6] K framework evm-semantics. — URL: https://github.com/kframework/evm-semantics.

[7] NEO-VM. — URL: https://github.com/neo-project/neo-vm (online; accessed: 2018-12-14).

[8] Nakamoto Satoshi. Bitcoin: A peer-to-peer electronic cash system. — 2008. — URL: http://www.bitcoin.org/bitcoin.pdf.

[9] Survey on blockchain technology, consensus algorithms, and alternative distributed technologies / Aleksey Fefelov, Nikita Mishin, Vyacheslav Bushev et al. — 2019.

[10] Wood Gavin. Ethereum: A secure decentralised generalised transaction ledger EIP-150 REVISION (759dccd - 2017-08-07). — 2017. — Accessed: 2018-12-14. URL: https://ethereum.github.io/yellowpaper/paper.pdf.