УДК 004

СОЗДАНИЕ И ТЕСТИРОВАНИЕ СМАРТ-КОНТРАКТА

Андреева Ю. А., Сафарьян О. А.
Донской государственный технический университет, г. Ростов-на-Дону, Российская Федерация
julliaand@yandex.ru
safari_2006@mail.ru

Представлена реализация смарт-контракта аренды жилья и тестов на языке Solidity, работающих быстрее готовых тестов на 25%, что облегчает отладку смарт-контрактов для их загрузки в блокчейн-систему Ethereum.

Ключевые слова: блокчейн, Ethereum, язык смарт-контрактов Solidity, тестирование, Remix, JavaScript тесты, Mist Ethereum Wallet.

UDC 004

SMART-CONTRACT CREATION AND TESTING

Andreeva Yu. A., Safaryan O. A.
Don State Technical University, Rostov-on-Don, Russian Federation

julliaand@yandex.ru
safari_2006@mail.ru

The paper presents the implementation of residential lease smart-contract and tests in Solidity programming language, which works faster than common tests on 25%, that simplifies testing smart-contracts for their deploying in the blockchain-system Ethereum.

Keywords: blockchain, Ethereum, language of smart-contracts Solidity, testing, Remix, JavaScript tests, Mist Ethereum Wallet.

Введение. Информационные технологии стремительно развиваются по всему миру и внедряются во все сферы жизни человека. После выхода в 2009 году блокчейн-платформы Bitcoin появилась идея создания такой распределенной блокчейн-системы Ethereum, которая позволила бы совершать «честные» транзакции с помощью уникальных смарт-контрактов [1]. Сеть Ethereum пригодна для любой финансово-экономической сферы, и на сегодняшний день ведутся разработки различных сервисов и приложений крупными кампаниями. Разработчики данной блокчейн-системы полагают, что в ближайшие 10–20 лет произойдет переход от привычной «ручной» системы к системе Ethereum не только в кампаниях, но и на государственном уровне во многих странах ввиду открытости и «прозрачности» транзакций и более низкого уровня стоимости.

Разработка смарт-контракта требует тщательного анализа и проработки функций. Одним из самых важных этапов жизненного цикла смарт-контрактов является тестирование. Тестирование призвано устранить различные ошибки и проверить соответствие функций «умных» контрактов заявленным требованиям. Данная процедура необходима для уменьшения потенциальных информационных рисков, так как после загрузки контракта в сеть его невозможно изменить в случае обнаружения ошибок [2].

Медицина, государственные реестры и недвижимость являются главными сферами разработки и потенциального внедрения смарт-контрактов. В апреле 2018 года японская кампания Ruden Holdings совместно с Blockchain Global Limited создала сервис для оплаты стоимости недвижимости в биткоинах и огласила результаты тестирования. Однако он не был внедрен и в открытом доступе, насколько известно на сегодняшний день, подобных сервисов нет [3].

Целью данного исследования является создание и тестирование смарт-контракта в сфере недвижимости. К задачам относятся:

- создание смарт-контракта аренды жилья;
- написание тестов для данного смарт-контракта.

Основная часть. Для написания контрактов на платформе Ethereum предусмотрен специальный язык программирования Solidity, однако большинство существующих

высокоуровневых языков программирования также имеют библиотеки и надстройки для их написания [4].

Протестировать смарт-контракт можно в известных локальных тестовых сетях Ethereum, TestRPC, Truffle, Parity, используя готовые общие тесты, или написать собственные тесты на языках программирования Solidity, JavaScript, Python, C/C++ и тд. Написанные вручную тесты для конкретного смарт-контракта позволяют учесть все особенности функций и более точно протестировать код.

В ходе исследования на языке программирования Solidity написан смарт-контракт со следующим функционалом:

```
contract Rent {

    struct Ad{
        uint256 number;
        uint256 price;
        bool isRent;
        address owner;
    }

    struct Agreement{
        address tenant;
        address owner;
        uint price;
        uint256 number;
        bool isRent;
    }

    Ad[] public pullAd;
    Agreement[] pullAgreement;
```

Рис. 1. Структура смарт-контракта Rent

- Публикация объявления. На вход функции поступают ежемесячный платеж за аренду жилья, кадастровый номер квартиры и метка булевого типа аренды. На выходе функции номер объявления в системе.

```
function createAd(uint256 price, uint256 numberKadastr, bool isRent) public{
    pullAd.push(Ad(
                    numberKadastr,
                    price,
                    isRent,
                    msg.sender
                ))-1;
}
```

Рис. 2. Публикация объявления смарт-контракта Rent

- Уничтожение объявления. Функция может вызываться либо владельцем, либо арендатором при заключении договора.

```
function destroyAd(uint256 addressAd) public{ delete pullAd[addressAd];}
/*destroy an advertisement*/
function destroyAdByKadastr(uint256 numberKadastr) public{
        for(uint256 i = 0; i < pullAd.length; i++)
            if(pullAd[i].number == numberKadastr)
                destroyAd(i);
}
```

Рис. 3. Уничтожение объявления смарт-контракта Rent

- Уничтожение договора. На вход функции поступают номер договора.

```
function destroyAgreementByKadastr(uint256 numberKadastr) public{
        for(uint256 i = 0; i < pullAgreement.length; i++)
            if(pullAgreement[i].number == numberKadastr)
                destroyAgreement(i);
}
```

Рис. 4. Уничтожение договора смарт-контракта Rent

• Оплата проживания на момент заключения договора. На вход функции номер объявления. На выходе — номер подписанного договора аренды.

```
/*sign a lease*/
function createAgreement(uint256 numberKadastr) public returns(bool){
    Ad memory currentAd = pullAd[getAdByKadastr(numberKadastr)];
    address tenantAddress = msg.sender;
    //if (balanceOf[tenantAddress] < currentAd.price) return false;
    transfer( tenantAddress, currentAd.price);
    pullAgreement.push(Agreement(
        tenantAddress,
        currentAd.owner,
        currentAd.price,
        currentAd.number,
        currentAd.isRent
    ));
    destroyAdByKadastr(numberKadastr);
    //getAdByKadastr(numberKadastr);
    return true;

}
```

Рис. 5. Подписание договора на аренду смарт-контракта Rent

Кроме вышеперечисленных функций, контракт включает создание токенов, необходимых для проведения транзакций.

Для написания собственных тестов был выбран язык Solidity. Особенность тестирования смарт-контракта таким способом заключается в вызове функций контракта Rent из другого контракта — contract RentTest.

```
contract RentTest {

    Rent rentToTest;


        uint256 tempPrice = uint256(10);
    uint256 tempKudastr = uint256(100121);
    uint256 numberAd;


function beforeAll () public {
    rentToTest = new Rent(10);
    tempPrice = uint256(10);
    tempKudastr = uint256(100121);
    -numberAd;
}
```

Рис. 6. Структура тестового смарт-контракта TestRent

Для тестирования написаны функции проверки существования объявления и договора.

```
function checkCreateAd() public {
    rentToTest.createAd(tempPrice,tempKudastr,false);

   Assert.equal(rentToTest.countAd(), uint256(1), "it should 1 item");
   Assert.equal(rentToTest.isRent(0), bool(false), "it should be not rent");
    //Assert.equal(rentToTest.arAd[1].Price, uint256(10), "it should be price 10");
}

function checkCreateContract () public {


    rentToTest.createAgreement(tempKudastr);


    Assert.equal(rentToTest.countAd(), uint256(1), "it should be empty");
    Assert.equal(rentToTest.countAgreement(), uint256(1), "it should 1 item");
    //Assert.equal(startClintBalance-nowClintBalance, uint256(10), "it should be change 10");
}
```

Рис. 7. Функции тестового смарт-контракта TestRent

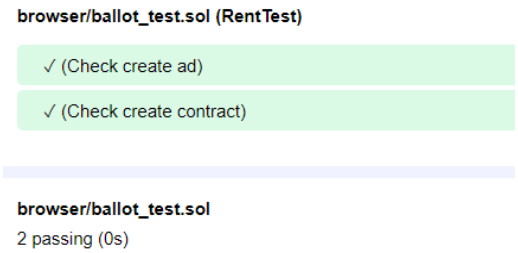

Рис. 8. Результат тестирования смарт-контракта Rent

Использование возможностей языка Solidity для написания тестов имеет преимущество в сравнении с тестами, написанных на языке программирования JavaScript на платформе Truffle, заключающееся в том, что нет необходимости в скачивании и подключении сторонних библиотек тестирования и миграции контракта. Среда Remix Solidity позволяет компилировать контракт и тестировать его в браузере до загрузки в приватную сеть Ethereum'а [5]. Главное преимущество, по сравнению с js-тестами, заключается в написании не отдельных связанных между собой *.js файлов, а родительского контракта, который взаимодействует с контрактом аренды напрямую с использованием встроенных инструментов языка Solidity, что повышает как скорость загрузки тестов, так и скорость тестирования функций контракта в целом. Общие тесты предусматривают набор всевозможных аргументов, в то время как в конкретных тестах определенные аргументы для проверки входных данных. Наличие меньшего количества аргументов повышают скорость тестирования каждой функции контракта.

На рис. 9 показан график зависимости скорости тестирования (в миллисекундах) контракта аренды жилья от количества итераций с использованием собственных тестов contract TestRent в Remix Solidity.

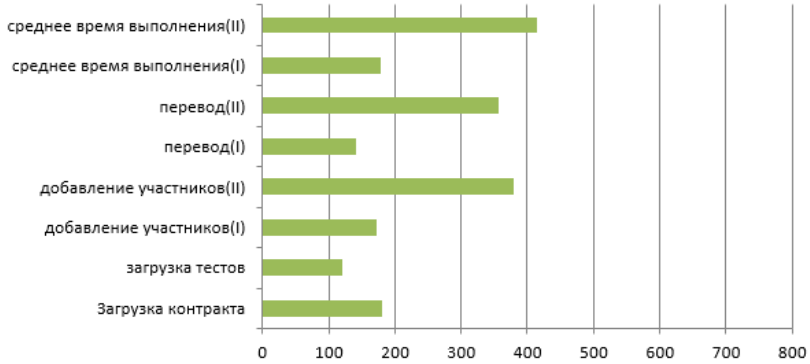

Рис. 9. График скорости тестирования смарт-контракта Rent с использованием TestRent

Рис. 10 отражает зависимость скорости тестирования (в миллисекундах) контракта от количества итераций в тестовой среде Mist Ethereum Wallet с использованием встроенных тестов.

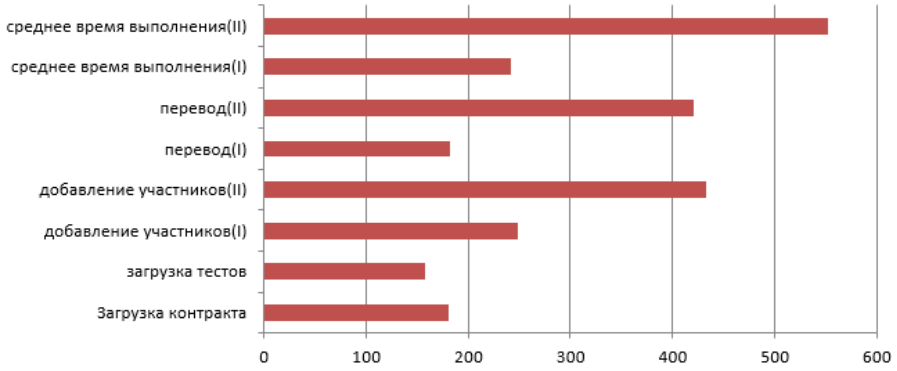

Рис. 10. График скорости тестирования смарт-контракта Rent с использованием Mist Ethereum Wallet

Среднее время выполнения контракта на малых итерациях с использованием своих тестов уменьшилось на 26,5%, на больших итерациях скорость тестирования повышена на 24,8% в сравнении с тестами в среде Ethereum Mist.

Заключение. В качестве достижений проведенного исследования можно отметить создание протестированного смарт-контракта сдачи и съема жилья в аренду в блокчейн-системе Ethereum, а также написание собственных тестов, работающих в среднем на 25% быстрее на больших и малых итерациях в сравнении с тестовой сетью Mist Ethereum Wallet. Умный контракт может быть использован для автоматизированного заключения договоров аренды при отсутствии посредников или взят за основу дальнейших разработок полноценных сервисов недвижимости с добавлением возможности оформления кредитов на недвижимость на основе социальных рейтингов и финансовой истории. При этом необходимо взаимодействие с другими смарт-контрактами, обеспечивающими функционирование банков и других систем.

Библиографический список

1. Ethereum blockchaim app platform/ Build unstoppable application. — Режим доступа: https://www.ethereum.org (дата обращения: 15.02.2019).

2. The importance of unit testing for smart-contracts. — Режим доступа: https://www.verypossible.com/blog/the-importance-of-unit-testing-for-smart-contracts (дата обращения: 18.02.2019).

3. Japanese company trials BTC and smart-contracts in real estate transactions. — Режим доступа: https://news.bitcoin.com/japanese-company-btc-smart-contracts-real-estate/ (дата обращения: 19.02.2019).

4. The 6 most common blockchain programming languages. — Режим доступа: https://www.verypossible.com/blog/the-6-most-common-blockchain-programming-languages (дата обращения: 16.02.2019).

5. Debajani Mohanty. Ethereum for Architects and Developers: With Case Studies and Code Samples in Solidity / Debajani Mohanty. – Apress, 2018. – 267 p. – p. 56-76.