МРНТИ 20.53.19 **https://doi.org/10.58805/kazutb.v.3.28-1083**

ВЗАИМОЗАМЕНЯЕМЫЕ И НЕВЗАИМОЗАМЕНЯЕМЫЕ ТОКЕНЫ СО СНИМКАМИ В JAVA

[1]Б. Рысбекқызы✉, [1]Ә.Ж. Изат, [2]А. Рыспек

[1]Карагандинский технический университет им. Абылкаса Сагинова, Караганда, Казахстан,
[2]Astana IT University, Астана, Казахстан

✉*Корреспондент-автор: bakhytgulz@mail.ru*

Многие блокчейн-приложения обмениваются токенами, такими как биткоин и эфир, или реализуют их с помощью смарт-контрактов. Тенденция в блокчейне заключается в неизменном применении стандартов взаимодействия токенов от платформы к платформе, облегчая задачи проектирования с помощью надежных и широко используемых спецификаций. Однако использование семантики целевого языка может привести к технологическим преимуществам и более эффективным контрактам. Целью статьи представлен реинжиниринг реализации стандартов ERC-20 и ERC-721 OpenZeppelin в Takamaka, Java-фреймворке для программирования смарт-контрактов. Актуальным является решение проблемы о типах, разрешенных для владельцев токенов, и новая реализация для создания моментальных снимков токенов на основе древовидных карт, которая возможна в виртуальной машине Java. Более того, это относится и к ERC-721, где ранее отсутствовал механизм моментального снимка. Тот же механизм моментального снимка может быть применен и за пределами смарт-контрактов для токенов.

Ключевые слова: смарт-контракт, модернизация программного обеспечения, Блокчейн, Токен, ERC-20, ERC-721.

FUNGIBLE AND NON-FUNGIBIE TOKENS WITH SNAPSHOTS IN JAVA

[1]B. Rysbekkyzy✉, [1]A.Zh. Izat, [2]A. Ryspek

[1]Abylkas Saginov Karaganda Technical University, Karaganda, Kazakhstan,
[2]Astana IT University, Astana, Kazakhstan,
e-mail: bakhytgulz@mail.ru

Many blockchain applications exchange tokens such as bitcoin and ether, or implement them using smart contracts. The trend in blockchain is to consistently apply token interaction standards from platform to platform, making design tasks easier with reliable and widely used specifications. However, using the semantics of the target language can lead to technological advantages and more efficient contracts. The purpose of the article is to reengineer the implementation of the ERC-20 and ERC-721 OpenZeppelin standards in Takamaka, a Java framework for programming smart contracts. It is relevant to solve the problem of the types allowed for token holders, and a new implementation for creating snapshots of tokens based on tree maps, which is possible in the Java virtual machine. Moreover, this also applies to ERC-721, which previously lacked a snapshot mechanism. The same snapshot mechanism can be applied outside of smart contracts for tokens.

Keywords: smart contract, software upgrade, Blockchain, Token, ERC-20, ERC-721

JAVA-ДАҒЫ СУРЕТТЕРІ БАР АУЫСПАЛЫ ЖӘНЕ АУЫСПАЛЫ ЕМЕС ТОКЕНДЕР

[1]Б. Рысбекқызы✉, [1]Ә.Ж. Изат, [2]А. Рыспек

[1]Әбілқас Сағынов атындағы Қарағанды техникалық университеті, Қарағанды, Қазақстан,
[2]Astana IT University, Астана, Қазақстан,
e-mail: bakhytgulz@mail.ru

Көптеген блокчейн қосымшалары Bitcoin және Ether сияқты токендермен алмасады немесе оларды ақылды келісімшарттар арқылы жүзеге асырады. Блокчейндегі Тенденция платформадан платформаға токендердің өзара әрекеттесу стандарттарын үздіксіз қолдану болып табылады, бұл сенім-

ді және кеңінен қолданылатын сипаттамалар арқылы дизайн тапсырмаларын жеңілдетеді. Алайда, мақсатты тілдің семантикасын қолдану технологиялық артықшылықтар мен тиімді келісімшарттарға әкелуі мүмкін. Мақаланың мақсаты-Takamaka-да ERC-20 және ERC-721 OpenZeppelin стандарттарын іске асыруды қайта құру, ақылды келісімшарттарды бағдарламалауға арналған Java шеңбері. Токен иелері үшін рұқсат етілген түрлер мәселесін шешу және Java виртуалды машинасында мүмкін болатын ағаш карталарына негізделген таңбалауыштардың суреттерін жасау үшін жаңа іске асыру өзекті болып табылады. Сонымен қатар, бұл бұрын суретке түсіру механизмі болмаған ERC-721-ге де қатысты. Дәл осындай суретке түсіру механизмін таңбалауыштар үшін ақылды келісімшарттардан тыс қолдануға болады.

Түйін сөздер: смарт-келісімшарт, бағдарламалық жасақтаманы жаңғырту, Блокчейн, Токен, ERC-20, ERC-721.

Введение. Для мирового крипторынка этот год может стать годом «фундаментальных изменений». Среди них расширение возможностей спотовых ETF (exchange-traded fund), интеграция искусственного интеллекта и токенизация активов. Все это, поможет создать более устойчивую и прибыльную экосистему. Президент Республики Казахстан Касым-Жомарт Токаев отметил, что в глобальной финансовой сфере наметился разворот к широкому использованию криптовалют и других цифровых активов и в таких условиях стране необходимо иметь четкую стратегию действий, направленных на преодоление новых вызовов.

Блокчейны используют [1] избыточное параллельное выполнение одних и тех же транзакций в децентрализованной сети машин для обеспечения их выполнения в соответствии с набором предопределенных правил. А именно, блокчейны затрудняют для отдельной машины нарушение семантики транзакций или их упорядочения: неправильно работающая отдельная машина немедленно выводится из консенсуса и изолируется. Биткойн [2] стал первой историей успеха блокчейна. Биткойн-транзакции - это переводы криптовалюты между счетами с особым правилом, согласно которому одни и те же средства не могут быть потрачены дважды. Криптовалюта [3] Bitcoin стала первым примером блокчейн-токена.

Через несколько лет после Биткоина другой блокчейн, называемый Ethereum [4, 6], представил возможность программирования транзакций на реальном императивном языке программирования Solidity, код которого скомпилирован для виртуальной машины Ethereum (EVM). Транзакции Ethereum по-прежнему оплачиваются в терминах его собственного токена ether, но они выполняют гораздо больше, чем переводы собственных токенов. А именно, транзакции также могут запускать конструкторы объектов и методы кодовых единиц, называемых смарт-контрактами, так что блокчейн Ethereum становится своего рода мировым компьютером, который сохраняет одни и те же объекты в памяти всех машин в сети блокчейна. Транзакции, включенные в блоки блокчейна, должны быть одинаковыми на каждой машине сети и приводить к одному и тому же результату. Машины, которые не соблюдают это правило, будут выведены из консенсуса, и их будущие транзакции будут отклонены другими машинами.

Как правило, смарт-контракты пишутся с использованием доменных языков, таких как Solidity, которые имеют специфические функции и ограничения для блокчейна. Совсем недавно тенденция в разработке смарт-контрактов сместилась в сторону использования хорошо известных языков программирования общего назначения, предоставляющих полезные синтаксические функции, отсутствующие в доменных языках, наряду с готовыми к использованию доступными наборами инструментов разработчика. Это открыло возможность реинжиниринга и оптимизации реализации нескольких существующих стандартов для различных блокчейнов. Java относится к числу этих языков общего назначения, поскольку пользуется большой попу-

лярностью [5] и современным набором инструментов программирования.

Популярный класс смарт-контрактов Solidity реализует динамический реестр переводов монет между счетами. Эти монеты являются не нативными токенами, а скорее новыми, производными токенами, реализованными в программном обеспечении с помощью смарт-контракта. Естественные и производные токены могут быть классифицированы многими способами [4,6]. Самая популярная классификация – это разделение на взаимозаменяемые и невзаимозаменяемые токены. Взаимозаменяемые токены взаимозаменяемы друг с другом, поскольку они имеют одинаковую номинальную стоимость, которая не привязана к какому-либо конкретному экземпляру токена. Как нативные токены, так и традиционные (фиатные) валюты являются взаимозаменяемыми токенами. Их основное применение - в области коллективного финансирования проектов и в первоначальных предложениях монет для поддержки стартапов. Так как, несменяемые токены имеют значение, которое зависит от их конкретного экземпляра. Следовательно, в целом они не взаимозаменяемы. В настоящее время их основное применение находится на рынке произведений искусства, где они представляют собой письменное заявление о передаче авторских прав правообладателю, в играх и, в целом, при нотариальном заверении.

Материалы и методы. Появилось несколько стандартов для взаимозаменяемых и нефункционирующих токенов, которые должны гарантировать корректность [7], доступность, интероперабельность, управление и безопасность смарт-контрактов [8,9], которые управляют токенами. Среди них запрос Ethereum на комментарий #20 (ERC-20 [11,12]) и #721 (ERC-721 [10]) являются наиболее популярными для взаимозаменяемых и невзаимозаменяемых токенов, соответственно, также за пределами Ethereum [4,6]. Они предоставляют разработчикам список правил, необходимых для корректной интеграции токенов с другими смарт-контрактами и приложениями, внешними по отношению к блокчейну, такими как кошельки, исследователи блоков, децентрализованные финансовые протоколы и игры.

Наиболее популярными реализациями стандарта ERC-20 являются Solidity, разработанная OpenZeppelin, командой программистов сообщества Ethereum, которые предоставляют полезные и безопасные смарт-контракты и библиотеки, и ConsenSys [8], позже устаревшая в пользу OpenZeppelin. Реализация ledger в OpenZeppelin использует сопоставление с целым числом, указывающим количество токенов, которыми владеет кто то. Эта карта является динамической, следовательно, она позволяет отслеживать процесс создания и передачи токенов. OpenZeppelin расширяет ERC-20 с помощью моментальных снимков, т.е. неизменяемые представления состояния токен-контракта, которые показывают его бухгалтерскую книгу в определенный момент времени. Они полезны для расследования последствий атаки, для создания форков токена и для реализации механизмов, основанных на балансах токенов, таких как взвешенное голосование. Моментальные снимки необходимы также для обеспечения неизменяемого представления бухгалтерской книги, которое может быть запрошено клиентом без риска того, что оно изменится во время запроса, что приведет к возникновению состояния гонки.

В случае ERC-721 стандартная реализация находится в Solidity, опять же от Open Zeppelin [8]. Эта реализация не предоставляет механизма моментального снимка, несмотря на полезность такой функции. Причина в том, что и без того очень сложная реализация в Solidity моментальных снимков для ERC-20 становится неразрешимой для более сложного стандарта ERC-721.

Спорный вопрос о токенах ERC-20 и ERC-721 заключается в том, кто может владеть токенами. Общепризнано, что учетные записи, принадлежащие внешним пользователям, могут содержать токены: это учетные записи, контролируемые людьми или внешними приложениями, следовательно, их поведение не является фиксированным. Как только они получат токен, человек или приложение могут решить сохранить его или продать в будущем. В этом случае нет риска

того, что токен останется заблокированным. Однако стандарты ERC-20 и ERC-721 также позволяют контрактам содержать токены. Это проблематично, поскольку контракты контролируются их кодом, который является неизменяемым. Если контракт получает токен, а его код не готов к работе с ним, в результате токен застревает навсегда: контракт никогда не будет его использовать и никогда не продаст его вперед. В Solidity нет решения этой проблемы, поскольку Solidity представляет как внешние учетные записи, так и контракты с одинаковым типом адреса, что делает их неразличимыми. Реализации ERC-721 пытались ограничить эту проблему ошибочным и хрупким способом, требуя, чтобы контракты, получающие токены ERC-721, должны были реализовывать интерфейс IERC721Receiver. По крайней мере, это подтверждает, что программист контракта был осведомлен о том, что последний может получать токены. К сожалению, в Solidity нет экземпляра operator для проверки реализации интерфейса, поскольку значения адресов не упакованы в Ethereum и не содержат информации о динамическом типе [9]. Как следствие, решение, основанное на стандарте ERC-165, является сложным, хрупким и, наконец, добровольным, так что его можно очень легко обойти, если контракты обманывают в отношении реализуемых ими интерфейсов.

Обсуждение и результаты. Актуальностью и практическим применением документа заключается в следующем:

- подробный анализ реализации ERC-20 и ERC-721 в OpenZeppelin Solidity;

- *are-разработанное решение, использующее функции Java, неприменимые в Solidity, для создания более чистых реализаций обоих стандартов;

- проверка того, что контракты, содержащие токены ERC-721, действительно реализуют приемник IERC721;

- реализация механизма для создания эффективных моментальных снимков в виртуальной машине Java (JVM), как для ERC-20 и ERC-721, которые фактически применялись и за пределами этих двух видов смарт-контрактов.

Разработанные смарт-контракты доступны в библиотеке поддержки Takamaka в виде непатентованного кода с открытым исходным кодом [8]. Они работают в блокчейне Hot moka [4], следовательно, не в блокчейнах, подобных Ethereum. В частности, смарт-контракты Hotmoka написаны на подмножестве Java под названием Takamaka, следовательно, не в Solidity. Такамака использует инструментарий байт-кода и аннотации к коду (метки, начинающиеся с @, такие как @View или @FromContract) для реализации концепций, специфичных для смарт-контрактов. Токены ERC-20 и ERC-721 ранее не портировались на Takamaka, следовательно, это первая версия ERC-20 и ERC-721 для этой платформы. Более того, насколько нам известно, это первая реализация токенов [10] ERC-721, которые корректно проверяют владельцев для реализации IERC721Receiver и которые предоставляют моментальные снимки. Решение и проблемы, связанной с типом держателей, которым разрешено хранить токены, которую Solidity пытается решить, используя частичный и хрупкий подход ERC-стандарт 165.

Языки могут иметь различные семантики для на первый взгляд похожих конструкций или требовать различных стилей кодирования для оптимизации, что чаще всего бывает в случае компиляции в разные виртуальные машины. Например, Vyper и Solidity компилируются для одной и той же EVM, и перевод из Solidity в Vyper почти мгновенный. Takamaka компилируется для JVM, и перевод из Solidity в Takamaka более сложен. Во многих случаях различные языки программирования имеют специфические решения, которые нельзя буквально перевести: например, в Java есть оператор instanceof, поэтому бессмысленно переводить технику, основанную на ERC-165 в Solidity для разрешения контрактам иметь только ERC-721 токены, если они явно не объявляют о реализации определенного интерфейса. Просто использовать instanceof в Java. Тем не менее, наш анализ обоих языков выявляет некоторые шаблоны перевода из Solidity в Takamaka.

ERC-20 и его реализация на открытом дири-

жабле.

Стандарт ERC-20 [11] определяет интерфейс с девятью функциями и двумя событиями, т.е. неизменяемыми метками, сохраняемыми в блокчейне для подтверждения некоторых логических поворотных моментов. Владельцами токенов являются адреса. В Solidity это нетипизированные указатели на внешние учетные записи (своего рода банковские счета, контролируемые внешним приложением или человеком) или на контракты (объекты, управляемые их кодом). Хотя в принципе контракты могут содержать токены, это может быть проблематично, если их код не запрограммирован для работы с такими токенами. В таком случае токены могут остаться заблокированными навсегда, поскольку только контракт может передавать их, но код контракта не имеет отношения к передаче токенов. Поэтому обычно предполагается, что токенами владеют только внешние учетные записи, но реализации ERC-20 не проверяют это ограничение и не запрещают передавать токены в контракты, даже непреднамеренно.

Прямые переводы: функция totalSupply() возвращает целочисленное общее количество токенов, находящихся в обращении. balanceOf(владелец адреса) показывает количество токенов, которыми владеет владелец. передача (адрес to, значение uint) переносит токены value с баланса вызывающего абонента на баланс to (uint - целое число без знака, равное 256 битам). Эта функция должна генерировать событие передачи.

Делегированные переводы: функция approve (делегат адреса, uint cap) позволяет делегату передавать до токенов cap от имени вызывающего абонента. Он должен выдать событие утверждения. передача от (владелец адреса, адрес кому, значение uint) передает токены значения от владельца к кому, но только в том случае, если владелец одобрил это вызывающей стороне. Эта функция должна генерировать событие передачи, разрешение (владелец адреса, делегат адреса) дает количество токенов, которые делегат был одобрен для передачи от имени владельца.

Необязательная информация: name() возвращает название токенов. symbol() возвращает символ токенов. функция decimals() возвращает количество десятичных разрядов токенов.

Первая часть интерфейса это просто API, динамической книги балансов токенов. Код OpenZeppelin, показанный на рис.1, хранит баланс пользователя в field_balances2 сопоставления типа (адрес ¼ [uint), который привязывает каждый адрес к количеству токенов, которые он хранит, и с целым полем _totalSupply, назначенным во время создания контракта. Вторая часть интерфейса позволяет владельцам токенов делегировать другим участникам передачу ограниченного количества токенов. Open Zeppelin реализует это через поле _allowances типа mapping (address =[сопоставление (address =[uint)): сопоставление от каждого владельца токена к другому сопоставлению от каждого делегата к его разрешенной шапке. Третья часть - просто информация о токенах в манифесте.

Как transfer, так и transfer From используют внутреннюю функцию _transfer, которая перемещает токены от владельца к получателю to, вызывая обработчик _beforeTokenTransfer. По умолчанию это ничего не делает, но подклассы могут переопределить его, чтобы добавить дополнительные функциональные возможности в контракт. Функция _transfer выполняет защитную проверку на наличие пропущенных значений (адрес(0)), которые могут возникнуть в результате неправильного использования контракта. Функция transfer From дополнительно проверяет, действительно ли владелец токенов делегировал msg.sender (вызывающему функцию) передать хотя бы токены value от его имени. Эта проверка выполняется после вызова _transfer, что нормально, поскольку функции Solidity не фиксируют свои побочные эффекты в случае сбоя. Код перевода их заканчивается вызовом _approve (не показан), который уменьшает допустимое значение.

OpenZeppelin добавляет функцию _mint, которая инициализирует общий запас токена: она является внутренней, поскольку предназначена для вызова из конструкторов подклассов, кото-

рые развертывают фактические экземпляры контракта. Эта функция использует адрес(0) для представления того факта, что отчеканенные токены берутся из ниоткуда.

```
contract ERC20 is IERC20 {
  mapping (address => uint) private _balances;
  mapping (address => mapping (address => uint)) private _allowances;
  uint private _totalSupply;
  string private _name;
  string private _symbol;

  constructor(string name_, string symbol_) { _name = name_;  _symbol = symbol_; }

  function totalSupply() public view virtual override returns (uint) {
    return _totalSupply;
  }

  function balanceOf(address owner) public view virtual override returns (uint) {
    return _balances[owner];
  }

  function transfer(address to, uint value) public virtual override {
    _transfer(msg.sender, to, value);
  }

  function allowance(address owner, address delegate) public view virtual override returns (uint) {
    return _allowances[owner][delegate];
  }

  function transferFrom(address owner, address to, uint value) public virtual override {
    _transfer(owner, to, value);
    uint currentAllowance = _allowances[owner][msg.sender];
    require(currentAllowance >= value, "transfer excess");
    _approve(owner, msg.sender, currentAllowance - value);
  }

  function _transfer(address owner, address to, uint value) internal virtual {
    require(owner != address(0), "transfer zero address");
    require(to != address(0), "transfer to zero address");
    _beforeTokenTransfer(owner, to, value);
    uint senderBalance = _balances[owner];
    require(senderBalance >= value, "transfer excess");
    _balances[owner] = senderBalance - value;
    _balances[to] += value;
    emit Transfer(owner, to, value);
  }

  function _mint(address account, uint amount) internal virtual {
    require(account != address(0), "mint to zero address");
    _beforeTokenTransfer(address(0), account, amount);
    _totalSupply += amount;
    _balances[account] += amount;
    emit Transfer(address(0), account, amount);
  }

  function _beforeTokenTransfer(address from, address to, uint amount) internal virtual {
  }
}
```

Рис.1 - Фрагмент реализации ERC-20 OpenZeppelin в Solidity

ERC-721 и его реализация на открытом дирижабле.

Стандарт ERC-721 [11] определяет интерфейс с десятью функциями и тремя событиями. Что касается стандарта ERC-20, владельцами токенов могут быть как внешние учетные записи, так и контракты, но контрактов след

ует избегать, если только они не были явно запрограммированы для работы с токенами ERC-721.

Функции стандарта ERC-721 предназначены для:

1. Прямые переводы: balanceOf (владелец адреса) показывает количество токенов, которыми владеет владелец. ownerOf (uint tokenId) возвращает владельца данного токена, если таковой имеется. transferFrom(адрес from, адрес to, uint tokenId) передает данный токен из from в to. В общем, вызывающий эту функцию должен совпадать с from или, по крайней мере, быть авторизован для передачи данного токена от имени from (смотрите позже). Эта функция даже не пытается проверить, является ли to внешней учетной записью или контрактом, который сможет работать с токеном. Если это не так, токен будет передан в to и закреплен навсегда. Из-за этого эта функция считается небезопасной. Эта функция должна генерировать событие передачи. safeTransferFrom(адрес from, адрес to, uint tokenId) ведет себя как transferFrom, но дополнительно пытается гарантировать, что to является внешней учетной записью или контрактом, способным работать с токеном. В этом смысле он считается безопасным.

2. Делегирование: функция approve (делегат адреса, uint tokenId) позволяет делегату передавать данный токен от имени вызывающей функции, которая должна быть владельцем токена или сама являться авторизованным оператором для токена. Предыдущий делегат (если таковой имеется) теряет свое делегирование после вызо-

ва этой функции. Эта функция генерирует событие утверждения. Функция устанавливает утверждение для всех (address operator, bool approved) позволяет оператору передавать все токены, принадлежащие вызывающей функции (если значение approved равно true), или удаляет это право (если значение approved равно false). Можно разрешить большее количество операторов для каждого владельца токена. Эта функция выдает подтверждение для всего события. get Approved (идентификатор токена uint) возвращает делегат для данного токена, если таковой имеется. Одобрено для всех (владелец адреса, оператор адреса) определяет, был ли оператор уполномочен передавать все токены, принадлежащие владельцу.

3. Необязательная информация: name () возвращает название токенов. symbol () возвращает символ токенов.

Реализация стандарта ERC-721 в OpenZeppelin относительно длительная, поэтому приводим только часть кода на рис.2. Большая часть информации хранится в четырех картах: _owners указывает, кто является владельцем каждого данного токена; _balances сообщает, сколько токенов принадлежит каждому данному владельцу; _tokenApprovals указывает, какой делегат был авторизован для каждого данного токена (если таковой имеется); и _operatorApprovals выдает набор утвержденных операторов для каждого владельца токена. Обратите внимание, что отображение (address =[book) на самом деле является набором утвержденных операторов: Solidity не имеет типа set, следовательно, наборы кодируются как их характерная карта.

TransferFrom вызывает вспомогательную функцию _transfer, которая уменьшает баланс отправителя, увеличивает баланс получателя и присваивает токен получателю (to). Нет никакой проверки того факта, что to на самом деле является внешней учетной записью или контрактом, способным работать с токеном, который он получает. Идея заключается в том, что контракты, готовые к получению токенов ERC-721, должны быть явно помечены их программистом как реализующие интерфейс IERC721Receiver, чей единственный метод onReceive вызывается, когда контракты получают токен ERC-721. В общем, было бы достаточно проверить это в instanceof IERC721Receiver, чтобы убедиться, что программист действительно ожидал, что контракт получит токены ERC-721, и вызвать onReceive в этом случае. Но это невозможно в Solidity, поскольку в этом языке отсутствует оператор instanceof и, как правило, в нем отсутствует какой-либо способ проверки динамического типа значений. Это не просто упущенная функция: на самом деле реализовать такую проверку в Solidity невозможно, поскольку Ethereum реализует данные в виде распакованных значений, так что их динамический тип недоступен и никакой оператор instanceof никогда не может быть реализован. Из-за этого ограничения программисты Solidity используют очень громоздкую технику, основанную на стандарте ERC-165 [12,13], заключающуюся в добавлении функции, которая выдает хэш сигнатур методов, реализуемых контрактом. Вызывая эту функцию, во время выполнения можно угадать интерфейсы, реализуемые контрактом. Этот метод очень прост, поскольку контракты могут свободно обманывать и притворяться, что реализуют интерфейс, который они на самом деле не реализуют. Однако это лучшее, что может сделать программист в Solidity. Существует еще более слабый подход к решению этой проблемы. А именно, стандарт токенов ERC-223 [12,13] требует, чтобы получатель токена передавал интерфейс IERC223Recipient, а затем вызывал его метод tokenReceived. Если получатель не реализует такой метод, транзакция завершается неудачей. Это даже слабее, чем ERC-165, поскольку он не пытается гарантировать, что получатель действительно был объявлен для реализации IERC223Recipient: приведения не проверяются в Solidity, они являются чистыми декорациями, чтобы заставить компилятор принять код, но они не проверяются во время выполнения.

Практическая значимость таких стандартов возрастает с развитием применения технологии блокчейн за пределами исходного контекста криптовалюты. Сегодня технология блок-

чейн используется в различных областях, таких как отслеживание энергопотребления, транспортные системы, управление цепочками поставок, IoV, IoT, облачные вычисления, поддержка децентрализации Интернета, а также включает технологию для децентрализованных идентификаторов. Различные контексты применения также подразумевают разные языки программирования, системы и платформы, которые должны реализовывать и поддерживать эти стандарты. Например, Hyperledger Fabric предлагает образцы реализации на Java, Go, Javascript. В отличие от этого, для Cosmos существуют реализации, написанные на Rust.

Выводы. В этой статье представлены несколько шаблонов для миграции кода из Solidity в Takamaka, применяемых к конкретным примерам контрактов ERC-20 и ERC-721. Он показывает улучшение буквального перевода кода контрактов ERC-20 с снимками с использованием карт с неизменяемыми клонами, недоступными в Solidity, но реализованными в других языках программирования. Показано, что та же техника применяется к контрактам ERC-721, где ранее отсутствовали снимки. Результаты подтверждены тестовым случаем, который демонстрирует реинжиниринг реализации стандартов ERC-20 и ERC-721 OpenZeppelin в Takamaka, Java-фреймворке для программирования смарт-контрактов. Возможность создания неизменяемых клонов карт полезна для упрощения кода других контрактов, где Solidity должен использовать сложный код или прибегать к событиям для отметки исторического развития данных и их восстановления.

Литература

1. Alazab, M., Alhyari, S., Awajan, S.W., Abdallah, A.B.: Blockchain technology in supply chain management: an empirical study of the factors affecting user adoption/acceptance//Clust. Comput.-2021.- Vol.24(1).- P.83–101. DOI10.1007/s10586-020-03200-4.

2. Alfandi, O., Khanji, S.I.R., Ahmad, L., Khattak, A.M.: A survey on boosting IoT security and privacy through Blockchain//Clust. Comput.- 2021. -Vol.24(1).- P.37-55. DOI 10.1007/s10586-020-03137-8.

3. Тусупов Д.А., Керімқұл С.Е., Изат Ә.Ж., Рысбекқызы Б., Криптовалютаны айырбастау - майнингті реттеу проблемасы//Труды Университета, Караганда, КарТУ им. А.Сагинова.- 2023.- 3(92) - С.504-512. DOI 10.52209/1609-1825_2023_3_504.

4. Antonopoulos, A.M.: Mastering Bitcoin: Programming the Open Blockchain, 2nd edn. O’Reilly, Sebastopol (2017). URL https://vk.com.- Date of access: 19.05.2025

5. Antonopoulos, A.M., Wood, G.: Mastering Ethereum: Building Smart Contracts and DApps. O’Reilly, Sebastopol (2018).URL https://books.google.co.in.- Date of access: 19.05.2025

6. Benini, A., Gambini, M., Migliorini, S., Spoto, F.: Power and pitfalls of generic smart contracts. In: Third International Conference on Blockchain Computing and Applications (BCCA’21), Tartu, Estonia, November 2021.- P.179–186. DOI 10.1109/BCCA53669.2021.9657048.

7. Buterin, V.: Ethereum Whitepaper (2013).URL https://ethereum.org/ en/whitepaper. Accessed October 2022.- Date of access: 19.05.2025

8. Carbonnelle, P.: PYPL. URL https://pypl.github.io/PYPL.html. Accessed July 2022.- Date of access: -21.05.2025

9. ConsenSys: Consensys Tokens. ConsenSys. URL https://github.com/ ConsenSys/Tokens. Accessed July 2022.- Date of access: 21.05.2025

10. Crafa, S., Di Pirro, M., Zucca, E.: Is Solidity solid enough?// Lecture Notes in Computer Science, St. Kitts and Nevis.- 2019.- P. 138-153.

11. Crosara, M., Olivieri, L., Spoto, F., Tagliaferro, F.: Re-engineering ERC-20 smart contracts with efficient snapshots for the Java virtual machine. In: Third International Conference on Blockchain Computing and Applications (BCCA'21), Tartu, Estonia, November 2021. – P.187–194. IEEE (2021) DOI 10.1109/BCCA53669.2021.9657047

12. Entrinken, W., Shirley, D., Evans, J., Sachs, N.: EIP-721: ERC721 Token Standard, Ethereum Improvement Proposals, No.721 (2018). URL https://eips.ethereum.org/EIPS/eip-721. - Date of access: -21.05.2025

13. Das, D., Banerjee, S., Chatterjee, P., Biswas, M., Biswas, U., Alnumay, W.S.: Design and development of an intelligent transportation management system using Blockchain and smart contracts//Clust. Comput. - 2022. -Vol.25(9). - P.1-15. DOI10.1007/s10586-022-03536-z

References

1. Alazab, M., Alhyari, S., Awajan, S.W., Abdallah, A.B.: Blockchain technology in supply chain management: an empirical study of the factors affecting user adoption/acceptance//Clust. Comput.-2021.-Vol.24(1).- P.83–101. DOI10.1007/s10586-020-03200-4.

2. Alfandi, O., Khanji, S.I.R., Ahmad, L., Khattak, A.M.: A survey on boosting IoT security and privacy through Blockchain//Clust. Comput.- 2021. -Vol.24(1).- P.37-55. DOI 10.1007/s10586-020-03137-8

3. Tusupov D.A., Kerimқұl S.E., Izat Ә.Zh., Rysbekқyzy B., Kriptovaljutany ajyrbastau - majningti retteu problemasy//Trudy Universiteta, Karaganda, KarTU im. A.Saginova.- 2023.- 3(92) - S. 504-512. DOI 10.52209/1609-1825_2023_3_504. [in Kazakh]

4. Antonopoulos, A.M.: Mastering Bitcoin: Programming the Open Blockchain, 2nd edn. O'Reilly, Sebastopol (2017). URL https://vk.com .- Date of access: 19.05.2025

5. Antonopoulos, A.M., Wood, G.: Mastering Ethereum: Building Smart Contracts and DApps. O'Reilly, Sebastopol (2018).URL https://books.google.co.in.- Date of access: 19.05.2025

6. Benini, A., Gambini, M., Migliorini, S., Spoto, F.: Power and pitfalls of generic smart contracts. In: Third International Conference on Blockchain Computing and Applications (BCCA'21), Tartu, Estonia, November 2021.- P.179–186. DOI 10.1109/BCCA53669.2021.9657048.

7. Buterin, V.: Ethereum Whitepaper (2013).URL https://ethereum.org/ en/whitepaper. Accessed October 2022.- Date of access: 19.05.2025

8. Carbonnelle, P.: PYPL. URL https://pypl.github.io/PYPL.html. Accessed July 2022.- Date of access: -21.05.2025

9. ConsenSys: Consensys Tokens. ConsenSys. URL https://github.com/ ConsenSys/Tokens. Accessed July 2022.- Date of access: 21.05.2025

10. Crafa, S., Di Pirro, M., Zucca, E.: Is Solidity solid enough?// Lecture Notes in Computer Science, St. Kitts and Nevis.- 2019.- P. 138-153.

11. Crosara, M., Olivieri, L., Spoto, F., Tagliaferro, F.: Re-engineering ERC-20 smart contracts with efficient snapshots for the Java virtual machine. In: Third International Conference on Blockchain Computing and Applications (BCCA'21), Tartu, Estonia, November 2021. – P.187–194. IEEE (2021) DOI 10.1109/BCCA53669.2021.9657047

12. Entrinken, W., Shirley, D., Evans, J., Sachs, N.: EIP-721: ERC721 Token Standard, Ethereum Improvement Proposals, No.721 (2018). URL https://eips.ethereum.org/EIPS/eip-721. - Date of access: -21.05.2025

13. Das, D., Banerjee, S., Chatterjee, P., Biswas, M., Biswas, U., Alnumay, W.S.: Design and development of an intelligent transportation management system using Blockchain and smart contracts//Clust. Comput. - 2022. -Vol.25(9). - P.1-15. DOI10.1007/s10586-022-03536-z

Сведения об авторах

Рысбекқызы Б. - PhD, и.о.доцента Карагандинского технического университета им. Абылкаса Сагинова , Караганда, Казахстан, e-mail: bakhytgulz@mail.ru;

Изат Ә.Ж. -инженер Казахстанского многопрофильного института Реконструкции и Развития (КазМИРР), Караганда, Казахстан, e-mail: celovek.dobra@gmail.com;

Рыспек А. Н. - Astana IT University, Астана, Қазақстан, e-mail:bakhytgulz@mail.ru

Rysbekkyzy B. - PhD, Acting Associate Professor, Abylkas Saginov Karaganda Technical University, Karaganda, Kazakhstan, e-mail:Bakhytgulz@mail.ru;

Izat A.- Engineer of the Kazakhstan Multidisciplinary Institute of Reconstruction and Development (KazMIRR), Karaganda, Kazakhstan, e-mail: celovek.dobra@gmail.com;

Ryspek A. - Astana IT University, Астана, Kazakhstan„e-mail: bakhytgulz@mail.ru