ИНФОРМАЦИОННЫЕ И ПРОГРАММНЫЕ АСПЕКТЫ РАЗРАБОТКИ И ПРИМЕНЕНИЯ СМАРТ-КОНТРАКТОВ

Федосеев С.В.*

***Ключевые слова**: байткод смарт-контракта, программные инструменты для разработки смарт-контрактов, виртуальная машина Ethereum, интегрированные среды разработки, язык программирования Solidity, оракул в блокчейн-сети, набор команд виртуальной машины Solidity, стековая организация вычислительной машины.*

Аннотация.

Цель работы: обоснование методических подходов к решению задач разработки и применения смарт-контрактов.

Методы: логическое моделирование правовых отношений и информационных связей, связанных с применением смарт-контрактов; системный анализ взаимосвязи предметной области правовой сферы, объектов сферы информационных сетевых технологий, программных средств, математических алгоритмов, основных объектов и методов теории принятия решений и теории множеств.

Результаты: выполнен анализ характеристик и методов применения программных инструментов, предназначенных для разработки, тестирования, развертывания и применения смарт-контрактов; определены характеристики, особенности использования и алгоритмические возможности языка программирования высокого уровня, предназначенного для разработки смарт-контрактов; выполнен анализ особенностей функционирования и применяемого набора операций виртуальной машины блокчейн-сети; представлена модель информационного взаимодействия постоянного хранилища, основной памяти и стека виртуальной машины блокчейн-сети; исследованы средства обеспечения информационного взаимодействия смарт-контракта с внешней средой.

DOI: 10.21681/1994-1404-2021-3-25-33

Введение. Общая характеристика смарт-контрактов

Концепция смарт-контрактов не является новой. Прототипы современных смарт-контрактов появились еще в конце 1990-х годов. Возобновление интереса к концепции смарт-контрактов связано с появлением технологии *блокчейн* и развитием средств обработки информации в сети [2, 7, 8, 12, 15].

Применяемые в сетях блокчейн программные и интерфейсные средства и языки программирования имеют высокую алгоритмическую сложность и обладают специфическими особенностями [1, 16, 17]. Анализ этих алгоритмов и особенностей представляет несомненный интерес.

Смарт-контракт — это договор между двумя и более сторонами об установлении, изменении или прекращении юридических прав и обязанностей, в котором часть или все условия записываются, исполняются и/или обеспечиваются компьютерным алгоритмом автоматически в специализированной программной среде [1, 5, 13].

Смарт-контракт может быть также определен как документ (компьютерная программа), соответствующий содержанию соглашения, которое автоматически выполняется при возникновении соответствующих условий.

Основными чертами смарт-контракта являются:

- реализация соглашения между сторонами согласно бизнес-логике;
- семантическая однозначность;
- автоматическое выполнение при достижении определенных условий;
- обеспечение правоприменения.

Вопрос правоприменения смарт-контракта является одним из важнейших [3, 10, 11, 17]. Смарт-контракт не может являться основанием для разрешения судебных споров, что связано с трудностью подтверждения соответствия «код — это закон». Однако обеспечивается следующее положение: если не возникают события, не предусмотренные условиями смарт-контракта, то отсутствует и необходимость в арбитре или третьей стороне для контроля либо влияния на его выполнение.

Если же такие события происходят, то смарт-контракт, не имеющий правил принятия решений в таких условиях, просто останавливается, прекращает свою работу.

* ***Федосеев Сергей Витальевич,*** *кандидат технических наук, доцент, профессор кафедры информационного права, информатики и математики Российского государственного университета правосудия, г. Москва, Российская Федерация.*
E-mail: fedsergvit@mail.ru

1. Рикардианские контракты

Идея представления текста заключаемого договора с использованием программного кода с последующим автоматическим его исполнением впервые была реализована в рикардианских контрактах. Эти контракты применялись в платежной системе *Ricardo* (отсюда их название) и предназначались для торговли облигациями. Первоначально рикардианские контракты реализовывались без использования технологии блокчейн.

Рикардианский контракт может быть определен как юридически обязывающий цифровой договор, определяющий условия взаимодействия двух или нескольких сторон, который криптографически [4, 9] подписан и подтвержден и читается как человеком, так и машиной[1].

Договор излагается на юридическом языке с внесением в текст меток для обеспечения машинного чтения. Обязательная *юридическая сила* основывается на предусмотренной процедуре предварительного согласования договора. Это обстоятельство, а также то, что контракт содержит условия, определенные в юридических терминах, позволяет использовать рикардианские контракты для судебного разрешения споров [18].

Следует заметить, что рикардианский контракт излагает намерения сторон, фиксирует юридическое соглашение, а также основанные на этом соглашении действия, которые предполагаются происходящими в будущем.

Наиболее часто рикардианские контракты применяются в финансовой сфере, которая характеризуется относительно простой бизнес-логикой, что делает возможным использовать простые конструкции программного кода.

Рикардианские контракты могут стать важной частью соглашений о блокчейн и заменить смарт-контракты, так как они могут одновременно действовать как смарт-контракты.

2. Инструменты для разработки смарт-контрактов

Набор программных средств, применяемых для разработки смарт-контрактов, достаточно широк. На рис.1 представлены лишь наиболее часто применяемые инструменты.

Для разработки программного кода смарт-контракта в *Ethereum* могут быть использованы два языка программирования:

- *Solidity* — наиболее популярный язык, являющийся, по сути, стандартом для рассматриваемой предметной области;
- *Vyper* — более простой язык, реже используемый, имеющий сходство с *Python*.

Другие, разработанные ранее языки программирования (такие, как *Mutan*, *LLL*, *Serpent*) в настоящее время практически не используются.

Компиляторы. Программный код, разработанный на языке высокого уровня (*Solidity* и *Vyper* являются такими языками), должен быть подвергнут компиляции для получения байткода (машинного кода), представляемого в двоичном формате.

В большинстве случаев эта процедура выполняется в *Ethereum* с использованием компилятора *Solc*. Полученный в результате компиляции байткод реализуется на виртуальной машине *Ethereum* (*EVM — Ethereum Virtual Machine*).

Основными функциями компилятора *Solc* являются:

- вывод программного кода контракта в двоичном формате;
- генерация *ABI*-интерфейса (*Application Binary Interface* — двоичный интерфейс приложений), который обеспечивает взаимодействие между байткодом (уровень виртуальной машины *EVM*) и программным кодом на языке высокого уровня;
- компиляция.

IDE (Integrated Development Environment) — интегрированные среды разработки).

Интегрированная среда *Remix* наиболее часто используется для разработки, отладки, тестирования и развертывания смарт-контрактов, написанных на языке *Solidity*.

Используется также и другая среда разработки — *Ethereum Studio*, обеспечивающая совершенный доступ к сети *Ethereum*.

API и инструменты.

API (*Application Programming Interface*) — программный интерфейс приложения. Именно посредством *API* отдельные программные компоненты взаимодействуют между собой. Компоненты могут быть низко- и высокоуровневыми, образовывать при этом сложную иерархию и использовать для взаимодействия *API* друг друга.

MetaMask — это инструмент, обеспечивающий возможность взаимодействия браузеров *Firefox* и *Chrome* с блокчейном *Ethereum* и позволяющий производить проверку транзакций, управлять учетными записями.

Личный блокчейн для тестирования.

Проверка программного кода разработанного смарт-контракта может быть выполнена в тестовой сети или в специально настроенной частной сети. Однако наиболее удобным для такой проверки программным средством является эмулятор *TestRPC*, значительно ускоряющий процедуру тестирования.

Ganache — инструмент, выполняющий те же функции, что и эмулятор *TestRPC*, но имеющий более совершенный пользовательский интерфейс и позволяющий получать подробную информацию о блоках и транзакциях.

Фреймворки.

Это программное обеспечение (платформа), облегчающее разработку и объединение разных компонентов большого программного проекта.

Truffle — среда разработки, которая с успехом применяется в *Ethereum* для тестирования и развертывания смарт-контрактов в любой сети этого блокчейна: тестовой, частной, публичной.

[1] *См.*: URL: https://iang.org/papers/fc7.html .

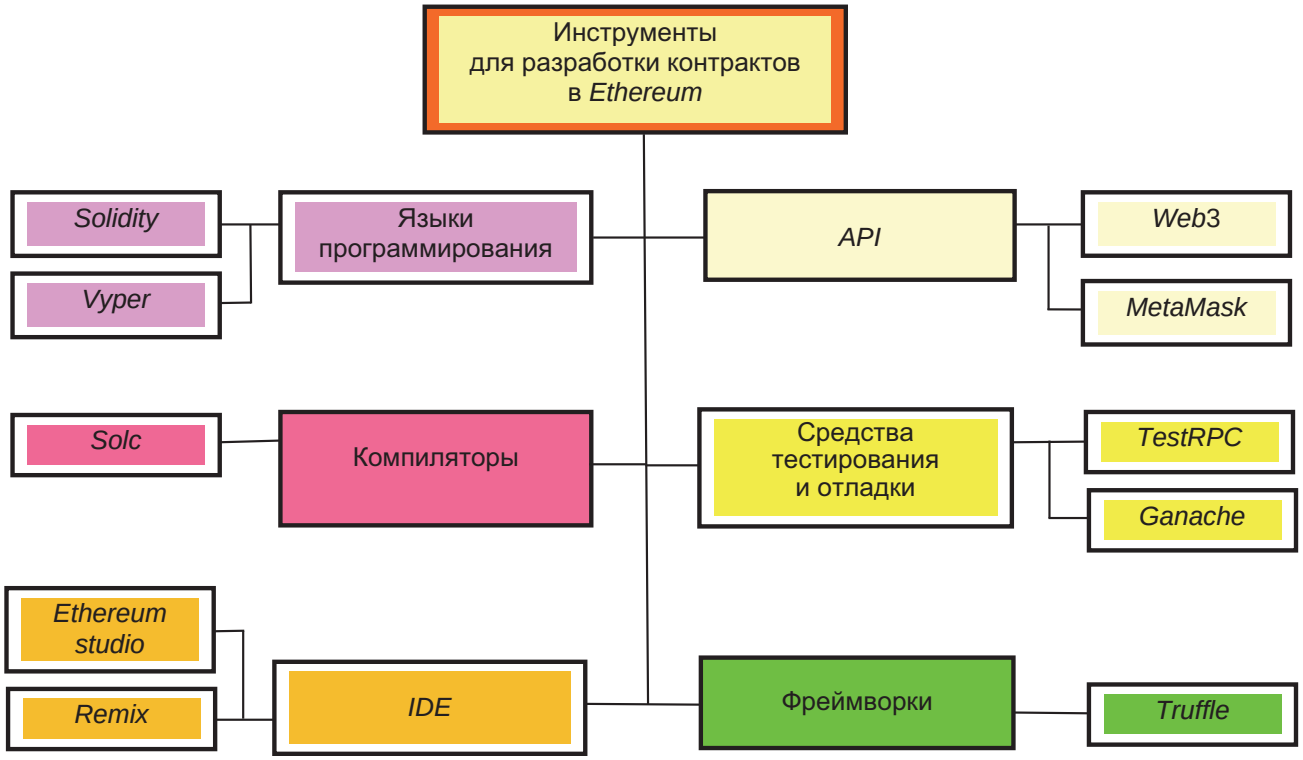

Рис. 1. Программные средства разработки контрактов в Ethereum

3. Язык программирования *Solidity*

Solidity определяется как объектно-ориентированный, предметно-ориентированный язык программирования, применяемый при разработке самовыполняющихся контрактов (смарт-контрактов) для блокчейн-платформы *Ethereum*.

Синтаксис языка *Solidity* подобен синтаксису языков *Java* и *C*. Этот язык успешно применяется для написания программных кодов смарт-контрактов. Однако по своим возможностям, по соответствию стандартам объектно-ориентированного программирования он уступает таким развитым языкам, как *Java*, *C++*, *C#*.

Особенностями языка *Solidity* являются динамические типы возвращаемых значений и статическая типизация переменных. Последнее означает, что любая переменная (локальная или глобальная) обязательно должна иметь тип, который указывается явно. Это позволяет на этапе компиляции выявлять ошибки, связанные с интерпретацией типов данных.

Реализация базовых принципов объектно-ориентированного программирования (наследование, инкапсуляция, полиморфизм) также имеет ряд особенностей.

Так, функциональностью класса (базовой сущности традиционного объектно-ориентированного языка) в *Solidity* наделяется контракт. И именно между контрактами осуществляется *наследование*, в результате которого переменные и функции наследуемого контракта сохранятся в точном соответствии в наследующем контракте и будут в нем доступны для использования. В связи с этим *Solidity* иногда определяют как контрактно-ориентированный язык.

В *Solidity* используются две категории типов данных: примитивные (простые) и ссылочные. Различие между ними состоит в том, что при выполнении операции присваивания или при передаче в функцию в качестве параметра использование данных первой категории приводит к передаче самого значения, а при использовании данных ссылочного типа — ссылки на то место памяти, где располагается используемое значение.

Основные типы данных.

Логический (булевый тип) — *bool*. Данные этого типа могут принимать два возможных значения: *true* и *false*.

Целые числа — этот тип данных может представлять целые знаковые *(int)* или целые беззнаковые числа (*uint*).

Адресный тип (*address*) — данные этого типа имеют значность 160 *бит* и состоят из нескольких полей, каждое из которых имеет свое предназначение при направлении запросов к контрактам и организации взаимодействия с ними.

Если на месте названия типа указано ключевое слово *var*, то выполняется автоматическое определение типа, при этом компилятор устанавливает переменной тип по значению, которое этой переменной присваивается.

Локальные переменные (типа *memory*) используются только при выполнении функции, с которой они связаны, и стираются по завершении этой функции.

Переменные состояния контракта (типа *storage*) связаны уже не с функцией, а с самим контрактом. Они определяют свойства контракта аналогично свойствам класса в объектно-ориентированном программировании.

Литералы в языке *Solidity* могут быть следующих видов: целочисленные литералы (последовательности десятичных цифр); строковые литералы (последовательность символов); шестнадцатеричные литералы.

Массивы в *Solidity*, так же как и в других языках программирования, определяются как набор смежных однотипных элементов, расположенных по определенному адресу в памяти. Отдельный элемент массива адресуется по значению его индекса.

С массивами в языке *Solidity* связаны два метода: *push* — добавляет новый элемент в массив; *length* — позволяет определить количество элементов в массиве.

Структуры в *Solidity* представляют собой конструкции, позволяющие объединять в единую логическую группу разнотипные данные.

Ключевое слово *struct* используется для объявления переменной структурного типа. Доступ к элементам структуры осуществляется с использованием оператора доступа — точки между объявленным именем структуры и именем адресуемого элемента.

Управляющие конструкции представлены в *Solidity* следующим набором средств разработки программного кода: *if…else; do; while; for; break; continue; return*. Эти управляющие конструкции используются так же, как и в других языках программирования (*JavaScript* или *C*).

Функции в *Solidity* связаны с контрактом. Они, как обычно, представляют собой модуль, который состоит из набора команд и многократно используется в процессе вычислений.

Для объявления функции используется ключевое слово *function*. Далее указываются: уникальное имя функции, набор параметров (может быть пустым), спецификаторы видимости (модификаторы доступа), ключевое слово *constant* (может отсутствовать), тип возвращаемого значения.

Ключевое слово *constant* определяет невозможность изменения функцией значений переменных в контракте. Она может использовать эти значения только для выполнения вычислений.

Если вызов функции выполняется из пределов рассматриваемого контракта, то такой вызов называется *внутренним*. Вызов называется *внешним*, если происходит обращение к функции, расположенной в другом контракте.

В *Solidity* при объявлении функций используются следующие спецификаторы видимости (модификаторы доступа):

external — функции этого типа могут быть вызваны из других контрактов и транзакций; стандартные процедуры не позволяют вызвать *external*-функцию внутри контракта, однако использование ключевого слова *this* позволяет это сделать;

internal — функции этого типа используются для обеспечения наследования в языке *Solidity*; такая функция, являясь внутренней функцией контракта-родителя, становится доступной дочерним контрактам;

public — функции типа *public* доступны как извне (из других контрактов и транзакций), так и изнутри контракта, они могут быть вызваны;

private — функции типа *private* отличаются от *internal* тем, что они недоступны для вызова из наследуемых дочерних контрактов и могут быть вызваны только из того контракта, где были объявлены.

Язык *Solidity* является представителем предметно-ориентированных языков (*DSL — Domain Specific Languages*). Это языки программирования, которые отражают специфику определенной предметной области, используют особые понятия и правила. По сравнению с универсальными языками программирования (*GPL — General-purpose Programming Languages*) они имеют меньший набор функций, который приспособлен для решения задач этой предметной области.

Приведем в качестве примера список доменных языков, используемых для разработки смарт-контрактов в финансовой сфере:

AxLang — DSL для написания проверяемых смарт-контрактов *Ethereum*;

Hyperledger Composer — среда разработки с открытым исходным кодом и *DSL* для написания смарт-контрактов в блокчейне *Hyperledger*;

Imandra — набор языков и инструментов для моделирования и проверки свойств финансовых бирж и смарт-контрактов *Ethereum*;

KolibriFX — язык для определения торговых стратегий иностранной валюты (*FX*) для исполнения на их облачной торговой платформе;

Marlowe — язык, ориентированный на финансовые контракты на блокчейнах;

Pyramid — DSL для написания смарт-контрактов для виртуальной машины *Ethereum*;

Rholang — язык смарт-контрактов, разработанный для технологии блокчейн *Synereo*, *RChain*.

4. Виртуальная машина *Ethereum* (*EVM*)

Виртуальная машина *Ethereum* представляет собой стековую среду, которая играет определяющую роль при реализации смарт-контрактов.

На этой виртуальной машине выполняется байткод контракта, полученный компилированием (переводом на машинный язык) исходного кода контракта, представленного на языке высокого уровня.

Машина *EVM* реализована в виде программы, экземпляр которой находится на любом узле сети *Ethereum*, имеющем полную функциональность. Она является тьюринг-полным автоматом, т. е. обладает полным набором инструкций (команд) для выполнения сложных алгоритмов обработки данных. В настоящее время *EVM* располагает набором из приблизительно 140 команд.

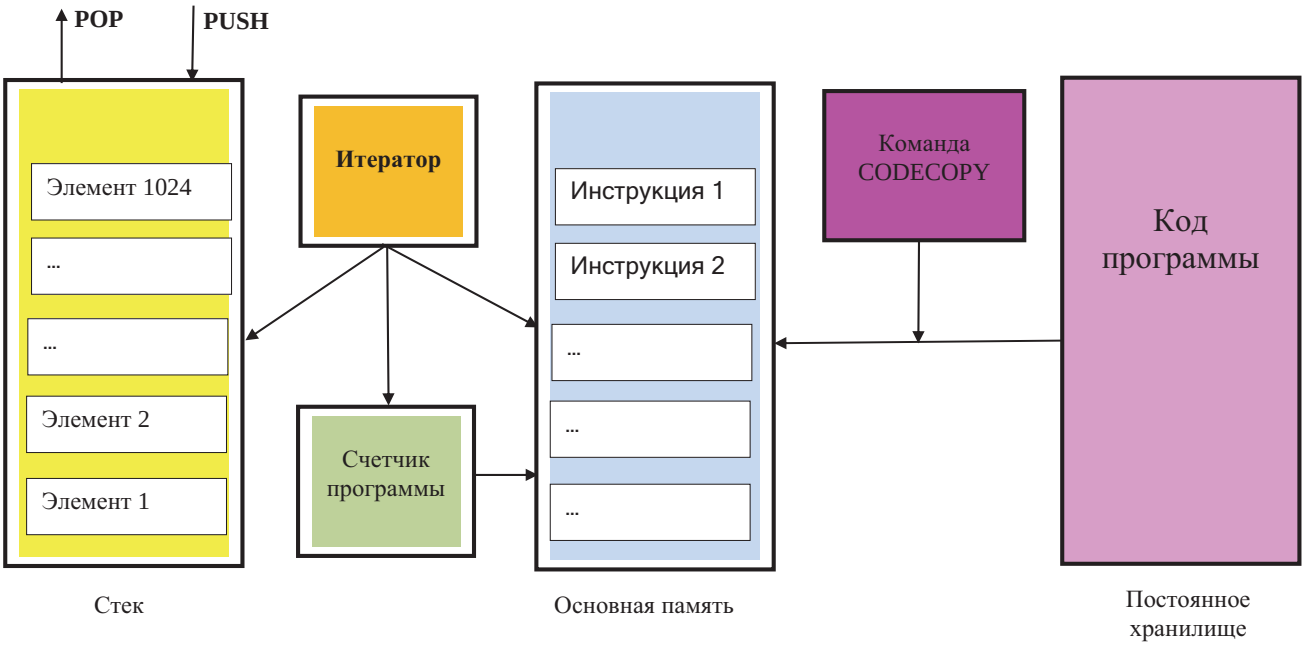

Рис. 2. Информационное взаимодействие постоянного хранилища, основной памяти и стека

Инструкции из этого набора обеспечивают выполнение процедур, разных по вычислительной сложности, которая измеряется специальной единицей вычислительной работы — *газ* (*gas*).

Выделяют два варианта платы за *газ*:

– фиксированный расход газа (*газ*, затраченный при выполнении команд, например, арифметических);
– динамически регулируемый расход газа (например, за размещение операндов и команд во всех местах хранения, кроме стека).

Суммирование вычислительных сложностей всех инструкций, составляющих байткод конкретного контракта, позволяет определить объем вычислительной работы, необходимой для его выполнения.

За размещение в сети *Ethereum* контракта или транзакции с отправителя взимается минимальная фиксированная плата — 21 000 *газ*. Если при реализации байткода контракта выявляется нехватка газа, то виртуальная машина останавливает работу и сообщает об ошибке.

Байткод, выполняемый на *EVM*, не может иметь доступ к внешним ресурсам (файлы и сеть). Такая изолированная среда способствует безопасному выполнению сложных и непроверенных программных кодов.

Важнейшим обстоятельством является то, что виртуальная машина *EVM* реализована на основе стека, ограниченного 1024 элементами. В этом случае любая выполняемая инструкция использует стек, а доступ к размещенным в стеке операндам осуществляется в порядке очереди (режим *LIFO — last in, first out*).

В отличие от виртуальных машин с регистровой организацией, при использовании стека пропадает необходимость в адресации регистров, что значительно упрощает и укорачивает тело команды, снижает сложность вычислительных процедур.

Кроме стека, данные в *EVM* хранятся еще в двух местах.

Первое — это основная память (аналог оперативной памяти в традиционной вычислительной машине). Байткод контракта реализуется только после его размещения в основной памяти, которая очищается, когда байткод контракта завершает выполнение.

Второе место — это постоянное хранилище с адресацией вида «ключ — значение» (аналог дискового накопителя), где размещается код программы контракта. Для пересылки кода программы из постоянного хранилища в основную память используется инструкция *CODECOPY*.

В *EVM* ширина машинного слова составляет 256 *бит* (32 *байт*), что принципиально отличается от обычных значений этого параметра в традиционных процессорах (от 8 до 64 *бит*).

В этом случае при очевидном возрастании алгоритмической сложности выполнения операций происходит существенное сокращение количества обращений к памяти, что значительно снижает время выполнения программы контракта.

Последнее объясняется следующими причинами:

– большая ширина машинного кода для инструкций приводит к сокращению количества используемых команд, упрощает их адресацию, делает команды более «информативными» (в теле команды размещается большее количество управляющих воздействий);
– использование «длинных» операндов приводит к сокращению их количества. Однако если значительная часть обрабатываемых операндов имеет меньшую длину, то это приводит к неэффективному использованию памяти;

– машинный код шириной 256 *бит* более удобен для выполнения криптографических операций (например, вычисления хеш-функции [6]).

Модель информационного взаимодействия постоянного хранилища, основной памяти и стека представлена на рис. 2.

Программный код, содержащийся в постоянном хранилище, пересылается в основную память и подвергается компиляции. Затем происходит последовательное выполнение его инструкций. Инкрементация счетчика программы происходит после чтения из основной памяти очередной инструкции, а обновление содержимого стека — после ее выполнения.

При выполнении очередной инструкции байткода виртуальная машина находится в соответствующем этой инструкции состоянии, которое характеризуется такими параметрами, как объем доступного газа, значение счетчика программы, содержимое стека.

Итератор (см. рис. 2), реализуя свои функции, обеспечивает установление соответствующего состояния *EVM*.

Функциями итератора являются:

– считывание очередной команды байткода из основной памяти;
– выполнение команд PUSH/POP (добавление/удаление элементов стека);
– изменение значения счетчика программы.

Набор команд (операций) виртуальной машины *EVM* составляют команды (операции) разных категорий:

– арифметические, логические, криптографические операции;
– операции, управляющие информацией об окружении, о блоке;
– операции со стеком, памятью, хранилищем, потоком управления;
– операции сохранения, дублирования, замены;
– журнальные операции;
– системные операции.

В *таблице* приведены характеристики арифметических операций: мнемоническое название; значение кода операции; количество элементов, добавляемых в стек (PUSH) и удаляемых из стека (POP); вычислительная сложность выполнения в единицах газа.

В качестве занимательной подробности отметим, что самой сложной является системная операция создания новой учетной записи с заданным кодом. Ее выполнение «стоит» 32 000 единиц газа.

Таблица

Основные характеристики арифметических операций виртуальной машины *EVM*

Название	Значение кода операции	*POP*	*PUSH*	Газ	Описание
STOP	0x00	0	0	0	Останавливает выполнение
ADD	0x01	2	1	3	Сложение двух значений
MUL	0x02	2	1	5	Умножает два значения
SUB	0x03	2	1	3	Операция вычитания
DIV	0x04	2	1	5	Целочисленное деление
SDIV	0x05	2	1	5	Целочисленное деление со знаком
MOD	0x06	2	1	5	Деление с остатком
SMOD	0x07	2	1	5	Деление с остатком и знаком
ADDMOD	0x08	3	1	8	Сложение по модулю
MULMOD	0x09	3	1	8	Умножение по модулю

5. Средства обеспечения информационного взаимодействия смарт-контрактов с внешней средой

Сеть блокчейн, как и любая другая децентрализованная распределенная сеть, действует изолированно от внешнего мира, но при этом во многих случаях должна учитывать изменения условий внешней среды. Самостоятельно получать сведения о таких изменениях сеть не может, поэтому для обеспечения доступа к информационным *off-chain*-ресурсам привлекается третья сторона [14].

Решение проблемы коммутации с реальным миром особенно важно для смарт-контрактов, которые должны контролировать бизнес-логику, учитывать множество факторов и в идеальном случае выполняться автоматически.

Оракулы входят в экосистему смарт-контрактов и используются для предоставления им достоверных и полных сведений об *off-chain*-ресурсах.

Существенно, что оракул является только поставщиком, посредником между источником информации и смарт-контрактом (*рис*. 3.).

В зависимости от реализуемого алгоритма информационного взаимодействия оракулы могут относиться к разным типам:

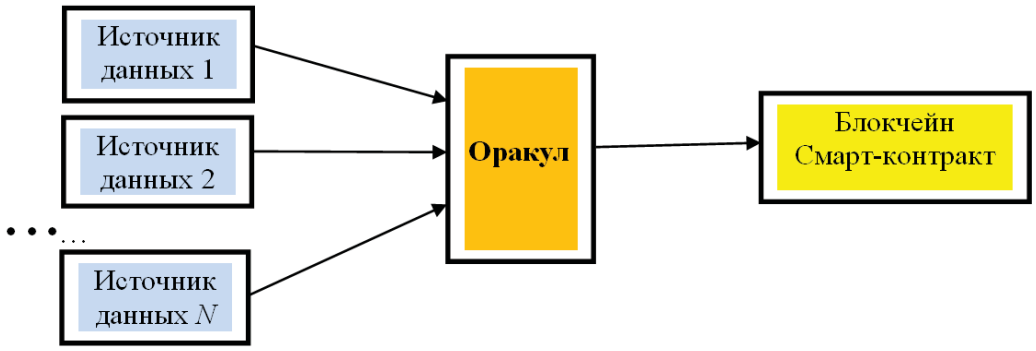

Рис. 3. Модель информационного взаимодействия оракула и смарт-контракта

– программный оракул — существует в виде программного продукта, обрабатывает сетевую информацию, получаемую с сайтов компаний;
– аппаратный оракул — фиксирует совершение событий (сигналов с разнообразных датчиков) и физическое выполнение условий;
– входящий оракул — направляет информацию из внешнего мира внутрь смарт-контракта;
– исходящий оракул — направляет информацию во внешний мир;
– оракул, основанный на консенсусе — предполагает использование не одного, а нескольких оракулов для повышения достоверности и полноты информации.

Услуги оракула могут быть оказаны при обращении к онлайн-сервисам, таким, например, как *www.oraclize.it* и *www.realitykeys.com*.

Сервис *TLSNotary/pagesigner* используется для доказательства подлинности информации, предоставляемой оракулом смарт-контракту. При этом с использованием криптографических методов подтверждается информационное взаимодействие между источником данных и оракулом.

Разработку программных продуктов, реализующих функции блокчейн-оракулов, ведут также такие компании, как *Smart Contract* (проект *ChainLink*), *BNC (BraveNewCoin)*.

Заключение

Таким образом, рассмотрены методы применения программных инструментов, предназначенных для разработки, тестирования, развертывания и применения смарт-контрактов, определены характеристики и особенности использования и алгоритмические возможности языка программирования высокого уровня, предназначенного для разработки смарт-контрактов; выполнен анализ особенностей функционирования и применяемого набора операций виртуальной машины блокчейн-сети; представлена модель информационного взаимодействия постоянного хранилища, основной памяти и стека виртуальной машины блокчейн-сети; исследованы средства обеспечения информационного взаимодействия смарт-контракта с внешней средой.

Литература

1. Блокчейн на пике хайпа. Правовые риски и возможности : монография / А.Ю. Иванов, М.Л. Башкатов, Е.В. Галкова и др. М. : Изд. дом Высшей школы экономики, 2020. 240 с.
2. Бегларян М.Е., Добровольская М.Ю. Блокчейн технология для цифровизации экономики: угрозы и перспективы // Правовая информатика. 2020. № 4. С. 46—54. DOI: 10.21681/1994-1404-2020-4-46-54 .
3. Ващекин А.Н., Дзедзинский А.В. Проблемы правового регулирования отношений в цифровом пространстве // Правосудие. 2020. Т. 2. № 2. С. 126—147. DOI: 10.37399/issn2686-9241.2020.2-126-147 .
4. Королев В.Т., Ловцов Д.А. Качество стандартизованной системы алгоритмов шифрования данных в ГАС РФ «Правосудие» // Правовая информатика. 2018. № 1. С. 49—59. DOI: 10.21681/1994-1404-2018-1-49-59 .
5. Ловцов Д.А. Имплементация «цифровых» прав в экономике: информационно-правовые аспекты // Российское правосудие. 2020. № 10. С. 42—53. DOI: 10.37399/issn2072-909X.2020.10.42-53 .
6. Ловцов Д.А. Теория защищенности информации в эргасистемах : монография. М. : Рос. гос. ун-т правосудия, 2021. 276 с. ISBN 978-5-93916-896-0.
7. Ловцов Д.А. Информационная безопасность автоматизированных блокчейн-систем: угрозы и способы повышения // Тр. II Междунар. науч.-прак. конф. «Трансформация национальной социально-экономической системы России» (22 ноября 2019 г.) / РГУП. М. : РГУП, 2020. С. 464—473. ISBN 978-5-93916-823-6.
8. Ловцов Д.А. Информационно-правовые основы правоприменения в цифровой сфере // Мониторинг правоприменения. 2020. № 2(35). С. 44—52. DOI: 10.21681/2226-0692-2020-2-44-52 .

9. Ловцов Д.А. Проблема гарантированного обеспечения информационной безопасности крупномасштабных автоматизированных систем // Правовая информатика. 2017. № 3. С. 66—74.
10. Ловцов Д.А., Терентьева Л.В. Правовое регулирование международных коммерческих электронных контрактов. Технологические и правовые аспекты электронной подписи // Lex russica. 2020. Т. 73. № 7. С. 115—126. DOI: 10.17803/1729-5920.2020.164.7.115-126 .
11. Максуров А.А. Блокчейн, криптовалюта, майнинг: понятие и правовое регулирование : монография. М. : Дашков и К. 2020. 198 с.
12. Проблемы создания цифровой экосистемы: правовые и экономические аспекты : монография / Е.Н. Абрамова, А.П. Алексеенко, С.Н. Белова и др. Под общ. ред. В.А. Вайпана, М.А. Егоровой. М. : Юстицинформ, 2021. 276 с.
13. Сажина М.А., Костин С.В. Блокчейн в системе управления знанием : монография. М. : ИНФРА-М, 2021. 90 с.
14. Тебернакулов А., Койфманн Я. Блокчейн на практике. М. : Альпина Паблишер, 2019. 260 с.
15. Управление бизнесом в цифровой экономике: вызовы и решения : монография / Под ред. И.А. Аренкова, Т.А. Лезиной, М.К. Ценжарик, Е.Г. Черновой. СПб. : СПбГУ, 2019. 360 с.
16. Цихилов А.М. Блокчейн: принципы и основы. М. : Интеллектуальная Литература, 2019. 188 с.
17. Черных А.М. Технологии распределенного реестра и защита документооборота судебной системы // Правовая информатика. 2020. № 4. С. 20—28. DOI: 10.21681/1994-1404-2020-4-20-28 .
18. Чурилов А.Ю. Правовое регулирование применения технологии блокчейн : монография. М. : Юстицинформ, 2021. 152 с.

*Рецензент: **Сухов Андрей Владимирович**, доктор технических наук, профессор, старший научный сотрудник научно-производственного объединения «Специальная техника и связь», Российская Федерация, г. Москва.*
E-mail: avs57@mail.ru

INFORMATION AND SOFTWARE ASPECTS OF THE DEVELOPMENT AND APPLICATION OF SMART CONTRACTS

Sergei Fedoseev, *Ph.D. (Technology), Associate Professor, Professor at the Department of Information Technology Law, Informatics and Mathematics of the Russian State University of Justice, Moscow, Russian Federation.*
E-mail: fedsergvit@mail.ru

Keywords: *smart contract bytecode, software tools for developing smart contracts, Ethereum virtual machine, integrated development environments, Solidity programming language, blockchain network oracle, Solidity virtual machine instruction set, stack computer architecture.*

Abstract.
Purpose of the paper: justifying methodological approaches to solving tasks of developing and applying smart contracts.
Methods used: logical modelling of legal and information relations related to using smart contracts, system analysis of the relationships in the subject area of the legal sphere, objects in the information network technologies sphere, software tools, mathematical algorithms, and the main objects and methods of decision theory and set theory.
Results obtained: an analysis of the characteristics and methods of using software tools for development, testing, deployment and application of smart contracts was carried out. The characteristics, features of use and algorithmic capabilities of a high-level programming language designed for the development of smart contracts were determined. An analysis of the features of functioning and applied set of operations of a blockchain network virtual machine was carried out. A model of information interaction of permanent storage, main memory and the stack of a blockchain network virtual machine was presented. Means for ensuring the information interaction of a smart contract with the external environment were studied.

References

1. Blokchein na pike khaipa. Pravovye riski i vozmozhnosti : monografiia. A.Iu. Ivanov, M.L. Bashkatov, E.V. Galkova i dr. M. : Izd. dom Vysshei shkoly ekonomiki, 2020. 240 pp.
2. Beglarian M.E., Dobrovol'skaia M.Iu. Blokchein tekhnologiia dlia tsifrovizatsii ekonomiki: ugrozy i perspektivy. Pravovaia informatika, 2020, No. 4, pp. 46-54. DOI: 10.21681/1994-1404-2020-4-46-54 .
3. Vashchekin A.N., Dzedzinskii A.V. Problemy pravovogo regulirovaniia otnoshenii v tsifrovom prostranstve. Pravosudie, 2020. T. 2, No. 2, pp. 126-147. DOI: 10.37399/issn2686-9241.2020.2-126-147 .

4. Korolev V.T., Lovtsov D.A. Kachestvo standartizovannoi sistemy algoritmov shifrovaniia dannykh v GAS RF "Pravosudie". Pravovaia informatika, 2018, No. 1, pp. 49-59. DOI: 10.21681/1994-1404-2018-1-49-59 .
5. Lovtsov D.A. Implementatsiia "tsifrovykh" prav v ekonomike: informatsionno-pravovye aspekty. Rossiiskoe pravosudie, 2020, No. 10, pp. 42-53. DOI: 10.37399/issn2072-909X.2020.10.42-53 .
6. Lovtsov D.A. Teoriia zashchishchennosti informatsii v ergasistemakh : monografiia. M. : Ros. gos. un-t pravosudiia, 2021. 276 pp. ISBN 978-5-93916-896-0.
7. Lovtsov D.A. Informatsionnaia bezopasnost' avtomatizirovannykh blokchein-sistem: ugrozy i sposoby povysheniia. Tr. II Mezhdunar. nauch.-prak. konf. "Transformatsiia natsional'noi sotsial'no-ekonomicheskoi sistemy Rossii" (22 noiabria 2019 g.). RGUP. M. : RGUP, 2020, pp. 464-473. ISBN 978-5-93916-823-6.
8. Lovtsov D.A. Informatsionno-pravovye osnovy pravoprimeneniia v tsifrovoi sfere. Monitoring pravoprimeneniia, 2020, No. 2(35), pp. 44-52. DOI: 10.21681/2226-0692-2020-2-44-52 .
9. Lovtsov D.A. Problema garantirovannogo obespecheniia informatsionnoi bezopasnosti krupnomasshtabnykh avtomatizirovannykh sistem. Pravovaia informatika, 2017, No. 3, pp. 66-74.
10. Lovtsov D.A., Terent'eva L.V. Pravovoe regulirovanie mezhdunarodnykh kommercheskikh elektronnykh kontraktov. Tekhnologicheskie i pravovye aspekty elektronnoi podpisi. Lex russica, 2020. T. 73, No. 7, pp. 115-126. DOI: 10.17803/1729-5920.2020.164.7.115-126 .
11. Maksurov A.A. Blokchein, kriptovaliuta, maining: poniatie i pravovoe regulirovanie : monografiia. M. : Dashkov i K, 2020. 198 pp.
12. Problemy sozdaniia tsifrovoi ekosistemy: pravovye i ekonomicheskie aspekty : monografiia. E.N. Abramova, A.P. Alekseenko, S.N. Belova i dr. Pod obshch. red. V.A. Vaipana, M.A. Egorovoi. M. : Iustitsinform, 2021. 276 pp.
13. Sazhina M.A., Kostin S.V. Blokchein v sisteme upravleniia znaniem : monografiia. M. : INFRA-M, 2021. 90 pp.
14. Tebernakulov A., Koifmann Ia. Blokchein na praktike. M. : Al'pina Pablisher, 2019. 260 pp.
15. Upravlenie biznesom v tsifrovoi ekonomike: vyzovy i resheniia : monografiia. Pod red. I.A. Arenkova, T.A. Lezinoi, M.K. Tsenzharik, E.G. Chernovoi. SPb. : SPbGU, 2019. 360 pp.
16. Tsikhilov A.M. Blokchein: printsipy i osnovy. M. : Intellektual'naia Literatura, 2019. 188 pp.
17. Chernykh A.M. Tekhnologii raspredelennogo reestra i zashchita dokumentooborota sudebnoi sistemy. Pravovaia informatika, 2020, No. 4, pp. 20-28. DOI: 10.21681/1994-1404-2020-4-20-28 .
18. Churilov A.Iu. Pravovoe regulirovanie primeneniia tekhnologii blokchein : monografiia. M. : Iustitsinform, 2021. 152 pp.