Своп: одноранговый протокол торговли токенами на платформе Эфириума

Сайт протокола Своп
ENCNJP
Майкл Овед, Дон Моситес
Опубликовано 21 июня 2017 г.

Реферат

Демонстрируется одноранговая методика торговли токенами ERC20 на платформе блокчейна Эфириум. Прежде всего приводится описание ограничений, связанных с использованием реестров заявок в блокчейне, и предлагается эффективный альтернативный метод одноранговой торговли токенами: согласование условий вне блокчейна и заключение сделки в блокчейне. Затем описывается протокол, посредством которого стороны могут сообщать другим о своем намерении торговать токенами. Установив связь, корреспонденты беспрепятственно сообщают друг другу цены и передают заявки. В рамках этого процесса стороны могут запрашивать цены у независимой третьей стороны, «оракула», с тем, чтобы подтверждать достоверность информации. В заключение мы демонстрируем «умный контракт» (смарт-контракт) на платформе Эфириума, позволяющий исполнять заявки с использованием блокчейна Эфириум.

Содержание

- Введение
  - Реестры заявок
  - Одноранговый протокол (P2P)
  - Ознакомление с протоколом Своп
- Протокол связи между одноранговыми узлами
- Протокол связи с Индексатором
- Протокол связи с Оракулом
- Смарт-контракт
- Заключение

Введение

На протяжении последних двенадцати месяцев число цифровых активов на платформе Эфириум (Ethereum) резко возрастало по мере того, как сценарии взаимодействия все чаще оформлялись как «умные контракты» (смарт-контракты). Наше исходное допущение заключается в том, что эта тенденция будет продолжаться в будущем; поэтому мы считаем, что по мере такого роста увеличится спрос на операции обмена (свопинга) между

пользователями, переходящими от одного сценария взаимодействия к другому или перебалансирующими свои портфели токенов. Обменам на основе реестров заявок в блокчейне свойственны некоторые неотъемлемые ограничения, но эти ограничения могут быть в определенной степени преодолены благодаря проектировочным решениям, общее описание которых приводится в нашей статье. Мы стремимся предложить альтернативу реестрам заявок в блокчейне, определив набор протоколов, позволяющий разблокировать ликвидность активов и обеспечить возможность беспрепятственного развития платформы Эфириума без таких ограничений.

Реестры заявок

Реестры заявок (order books) обеспечивают в высшей степени автоматизируемый способ приведения предложения в соответствие со спросом на тот или иной свободнообращающийся актив. Традиционно такие реестры централизованы и применяются в сочетании с протоколом исполнения заявок, позволяющим создавать, исполнять и отменять заявки, связываясь с центральным источником достоверной информации. Реестры заявок были перепроектированы в духе децентрализации с тем, чтобы ими можно было пользоваться в блокчейнах. Тем не менее, применение реестра заявок в блокчейне связано с некоторыми ограничениями.

Реестры заявок в блокчейне не масштабируются. Исполнение кода в блокчейне влечет за собой издержки, в связи с чем автоматизированный цикл «заявка — отмена заявки — заявка» быстро начинает требовать чрезмерных затрат и препятствует применению реестра заявок в качестве высокопроизводительной автоматизируемой системы согласования спроса и предложения. Фактически, если этот алгоритм согласования исполняется в блокчейне, сторона, создающая заявки, будет нести связанные с исполнением кода затраты, существенно возрастающие по мере увеличения объема реестра заявок.

Реестры заявок в блокчейне общедоступны. Так как транзакция создания заявки в блокчейне обрабатывается майнерами, эти майнеры узнают о существовании заявки прежде, чем заявка публикуется в реестре. Таким образом возникает возможность заключения сделок на основе еще не доступной другим сторонам информации, способная оказывать существенное влияние на первоначальную заявку. Кроме того, так как опубликованная заявка общедоступна, цена исполнения заявки одинакова для всех, что лишает поставщика возможности регулировать ликвидность в зависимости от спроса.

Реестры заявок в блокчейне не обеспечивают равноправие. Физически распределяемым системам неотъемлемо свойственно запаздывание связи между узлами сетей. Так как майнеры географически распределены, перед опытными и хорошо оснащенными сторонами открываются возможности совмещения и регистрации заявок с преодолением времени запаздывания, свойственного блокчейну — то есть, в результате, возможность действовать на основе информации о заявках прежде, чем это могут сделать другие стороны. В связи с такой асимметрией доступа к информации не столь опытные и не так хорошо

оснащенные стороны вполне могут терять всякий интерес к какому-либо участию в экосистеме.

Одноранговый протокол (P2P)

Альтернативный вариант одноранговой торговли позволяет индивидуальным сторонам торговать друг с другом напрямую. Большинство повседневно осуществляемых нами транзакций носят одноранговый характер: приобретение чашки кофе в кафе, продажа обуви на сайте аукциона eBay или покупка кошачьего корма на сайте Amazon. Так как все эти приватные транзакции осуществляются между людьми или предприятиями, каждая из сторон знает, с кем она заключает сделку, и в конечном счете может выбирать другую сторону.

Одноранговая торговля масштабируется. Заявки передаются от одной индивидуальной стороны к другой и выполняются поочередно, в отличие от заявок на общедоступной бирже, где полное исполнение заявок не гарантируется. В результате отмена заявок на бирже становится регулярным явлением, тогда как одноранговые заявки, как правило, выполняются, потому что транзакции предлагаются сторонам, уже выразившим интерес к заключению сделки. Кроме того, согласование предложения и спроса в одноранговом режиме может быть обеспечено посредством не требующего больших затрат процесса обнаружения участников того же ранга, в отличие от алгоритмического согласования интересов сторон, дорогостоящего как в блокчейне, так и вне блокчейна.

Одноранговая торговля приватна (конфиденциальна). После того, как две стороны находят друг друга и решают заключить сделку, для согласования условий их сделки не требуется участие каких-либо третьих сторон. Корреспонденция между этими сторонами продолжает оставаться приватной на всем протяжении переговоров, что не предоставляет другим сторонам возможность извлекать преимущества за счет процесса регистрации заявок. Заявка становится общеизвестной только тогда, когда она предъявляется к исполнению.

Одноранговой торговлей обеспечивается равноправие. Так как заявки создаются и передаются в процессе непосредственного взаимодействия двух сторон, никакие внешние участники не получают преимущество. Постольку, поскольку они сотрудничают с несколькими независимыми сторонами, участники могут согласовывать цены, сравнимые с теми, которые были бы им предложены на бирже, или более выгодные. Кроме того, стороны, назначающие цены за исполнение заявок, могут делать это агрессивно, не опасаясь того, что другие будут применять автоматизированные стратегии извлечения преимуществ за счет запаздывания связи.

В связи с ограничениями масштабируемости, приватности и обеспечения равноправия, свойственными реестрам заявок в блокчейне, возникла необходимость в альтернативной системе. Сегодняшней экосистеме Эфириума требуется открытое одноранговое решение задачи обмена активами.

Ознакомление с протоколом Своп

Своп (Swap) — протокол, способствующий внедрению надежной одноранговой экосистемы торговли токенами в блокчейне Эфириум. Ниже приводится описание расширяемой структуры, поддерживающей эффективные процессы обнаружения корреспондентов и проведения переговоров между ними. Такие протоколы предназначены стать основой экосистемы торговли активами и ускорить развитие экосистемы Эфириума. Публикуя эту статьи и предлагая ее на обсуждение, мы надеемся получить отзывы сторон, заинтересованных в развитии экосистемы с тем, чтобы разработать высококачественные протоколы, обеспечивающие возможность реализации широкого ассортимента приложений.

Протокол связи между одноранговыми узлами

Корреспонденты могут согласовывать условия сделки быстро, на равноправной основе и приватно (конфиденциально), обменявшись всего лишь несколькими сообщениями. В нашем документе «Заявителем» (Maker) называется сторона, предоставляющая заявку, а «Исполнителем» (Taker) — сторона, исполняющая заявку. Так как обе стороны пользуются узлами связи одного и того же ранга, каждая из сторон может в любое время брать на себя роль Заявителя или Исполнителя. В приведенном ниже описании допускается, что токены соответствуют стандарту ERC20, и что с использованием предлагаемого протокола можно торговать любыми токенами, удовлетворяющими этому стандарту.

Ниже приведена схема основной последовательности операций, выполняемых протоколом. Заявитель и Исполнитель согласовывают условия сделки вне блокчейна. «Контрактом» при этом называется «умный контракт» (смарт-контракт) на платформе Эфириума, который вызывается Исполнителем, когда он готов исполнить заявку в блокчейне.

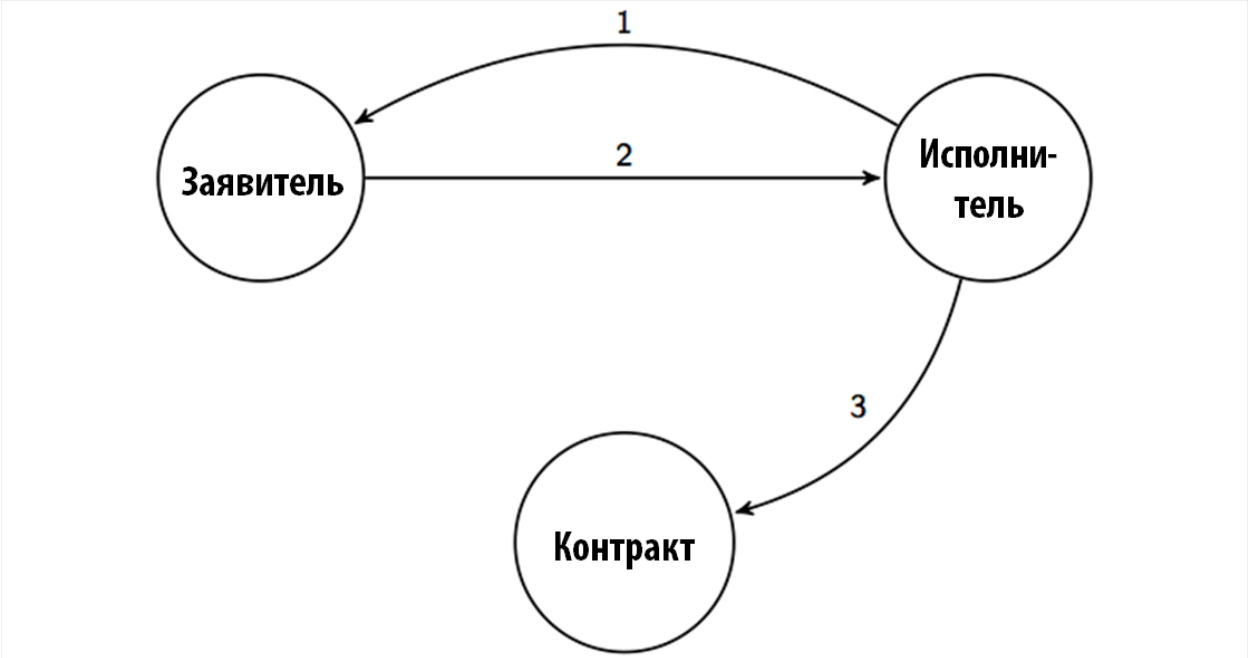

Рис. 1. Запрос, предоставление и исполнение заявки

1. Исполнитель вызывает Заявителя: *getOrder*.
2. Заявитель отвечает, предоставляя заявку.
3. Исполнитель вызывает Контракт: _fillOrder()_ .

Прикладные интерфейсы программирования заявок (Order API)

Следующие прикладные интерфейсы программирования (API) — удаленные вызовы процедур (RPC) без учета транспортного уровня, применяемые при установлении связи между одноранговыми узлами сети и сервисами. В примерах вместо адресов соответствующих стандарту ERC20 токенов используются тикеры (сокращенные обозначения) токенов. Подписи приведенных ниже вызовов приводятся с целью обсуждения, так как дальнейшие технические детали будут опубликованы в отдельном документе.

Прикладной интерфейс программирования заявки (Order API) используется вне блокчейна; в нем указываются асинхронные вызовы, которыми корреспонденты обмениваются в процессе согласования условий сделки. По усмотрению исполнителя, он может выбрать цикл «запрос — предоставление заявки» в качестве синхронного «запроса — ответа». Так как Заявитель подписывает заявку, впоследствии Исполнитель может представить заявку с целью исполнения смарт-контракта.

getOrder
(запрос о предоставлении заявки)

getOrder(makerAmount, makerToken, takerToken, takerAddress)
getOrder(сумма Заявителя, токен Заявителя, токен Исполнителя, адрес Исполнителя)
Called by a Taker on a Maker, requesting an order to trade tokens.
Исполнитель вызывает Заявителя, запрашивая заявку на торговлю токенами.

```
Пример: “I want to buy 10 GNO using BAT.”
«Я желаю купить 10 токенов GNO, пользуясь токенами BAT».
getOrder(10, ‘GNO’, ‘BAT’, <takerAddress>)
getOrder(10, ‘GNO’, ‘BAT’, <адрес Исполнителя>)
```

provideOrder
(предоставление заявки)

provideOrder(makerAddress, makerAmount, makerToken, takerAddress, takerAmount, takerToken, expiration, nonce, signature)
provideOrder(адрес Заявителя, сумма Заявителя, токен Заявителя, адрес Исполнителя, сумма Исполнителя, токен Исполнителя, дата истечения срока действия заявки, отметка текущего времени, подпись)

Вызов Исполнителя Заявителем с предоставлением подписанной заявки, готовой к исполнению.

```
Пример: “I’ll sell you 10 GNO for 5 BAT.”
«Я продам 10 токенов GNO за 5 токенов BAT».
provideOrder(<makerAddress>, 10, ‘GNO’, <takerAddress>, 5, ‘BAT’, <expiration>,
<nonce>, <signature>)
provideOrder(<адрес Заявителя>, 10, ‘GNO’, <адрес Исполнителя, 5, ‘BAT’, <дата истечения срока действия заявки>, <отметка текущего времени>, <подпись>)
```

Прикладной интерфейс программирования котировок (Quote API)

Котировки позволяют сторонам предоставлять друг другу информацию о ценах и не подлежат исполнению. Впоследствии котировки могут быть преобразованы в заявки, если их условия удовлетворяются обоими корреспондентами.

getQuote
(запрос о предоставлении котировки)

getQuote(makerAmount, makerToken, takerTokens)
getQuote(сумма Заявителя, токен Заявителя, токены Исполнителя)

Исполнитель вызывает Заявителя, запрашивая котировку в конкретных токенах.

```
Пример: “How much would it cost to buy 10 GNO using BAT?”
«Сколько будет стоить покупка 10 токенов GNO в токенах BAT?»
getQuote(10, ‘GNO’, [‘BAT’])
```

provideQuote
(предоставление котировки)

provideQuote(makerAmount, makerToken, takerAmounts)
provideQuote(сумма Заявителя, токен Заявителя, суммы Исполнителя)

Исполнитель вызывает Заявителя, предоставляя котировки в токенах Исполнителя.

```
Пример: “It will cost you 5 BAT for 10 GNO.”
«Покупка 10 токенов GNO будет стоить 5 токенов BAT».
provideQuote(10, ‘GNO’, { ‘BAT’: 5 })
```

Протокол связи с Индексатором

Индексатор — функционирующий вне блокчейна сервис, группирующий и согласующий одноранговые узлы связи с учетом их намерения торговать, то есть того, желают ли возможные Заявители и Исполнители покупать или продавать токены. Индексаторы — функционирующие вне блокчейна сервисы, накапливающие информацию о таких «намерениях торговать» и способствующие согласованию одноранговых узлов связи с учетом их намерения покупать или продавать конкретные токены. Многие возможные Заявители могут сообщать о своем намерении торговать и, когда Исполнитель посылает Индексатору запрос о поиске подходящих корреспондентов, он может получить множество результатов поиска. После того, как Исполнитель выберет Заявителя, с которым он хотел бы торговать, они начинают переговоры с использованием протокола связи между одноранговыми узлами, описание которого приведено выше. После того, как между Заявителем и Исполнителем будет достигнуто соглашение, заявка исполняется посредством вызова смарт-контракта.

На приведенной ниже схеме иллюстрируются взаимодействия Заявителя, Исполнителя и Индексатора. Все стороны — Заявитель, Исполнитель и Индексатор — взаимодействуют вне блокчейна, пользуясь любым предпочитаемым ими средством передачи сообщений.

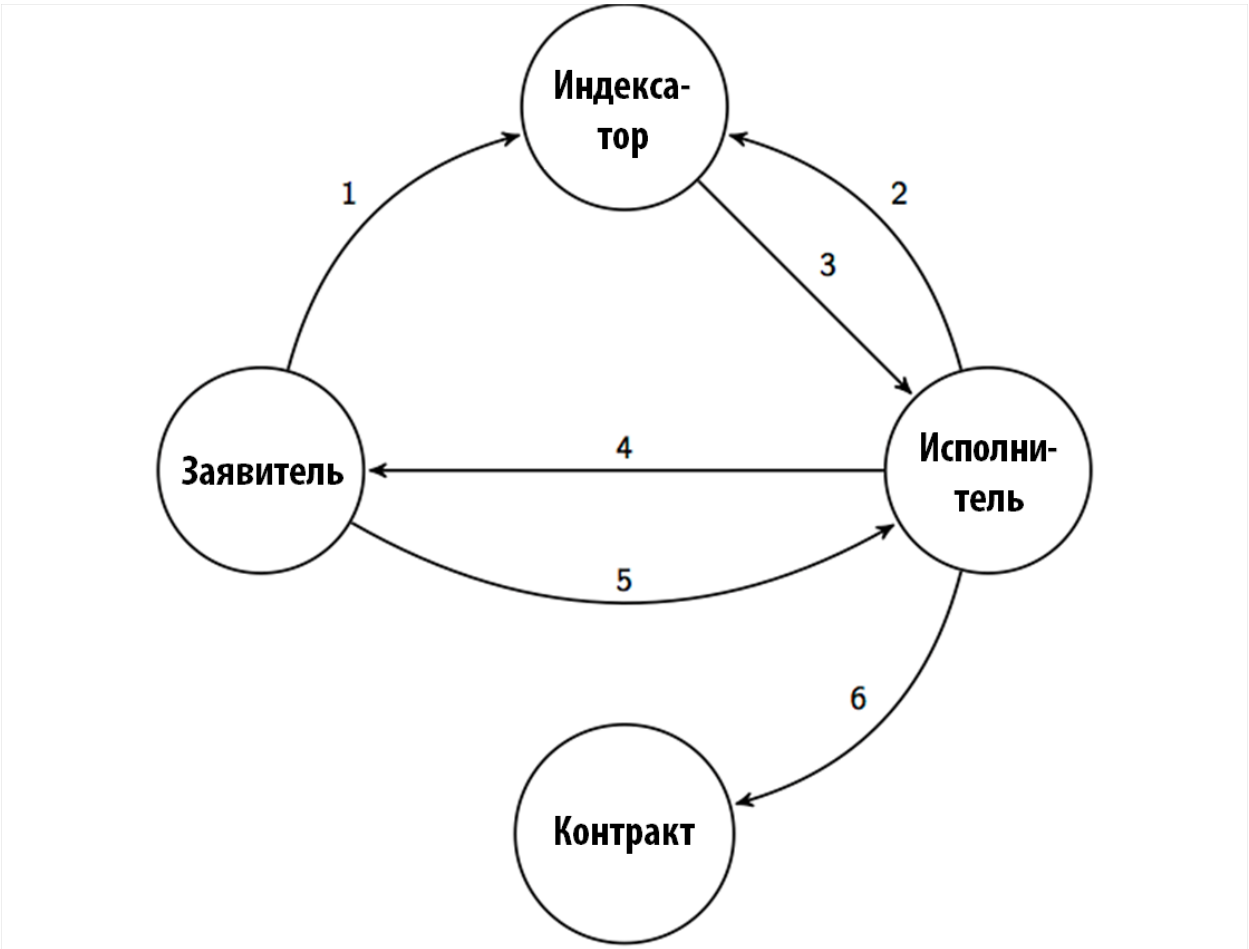

Рис. 2. Поиск корреспондента и заключение сделки

1. Заявитель вызывает Индексатора: *addIntent*.
2. Исполнитель вызывает Индексатора: *findIntent*.
3. Индексатор вызывает Исполнителя: *foundIntent(maker)*.
4. Исполнитель вызывает Заявителя: *getOrder*.
5. Заявитель отвечает, предоставляя заявку.
6. Исполнитель вызывает Контракт: *fillOrder(order)*.

На следующей схеме иллюстрируются взаимодействия нескольких Заявителя, Исполнителя и Индексатора. Каждый Заявитель независимо оповещает о своем намерении. Исполнитель запрашивает о поиске Заявителей, выражающих конкретное намерение, и Индексатор отвечает, предоставляя список соответствующих адресов в системе Эфириума и подробностей.

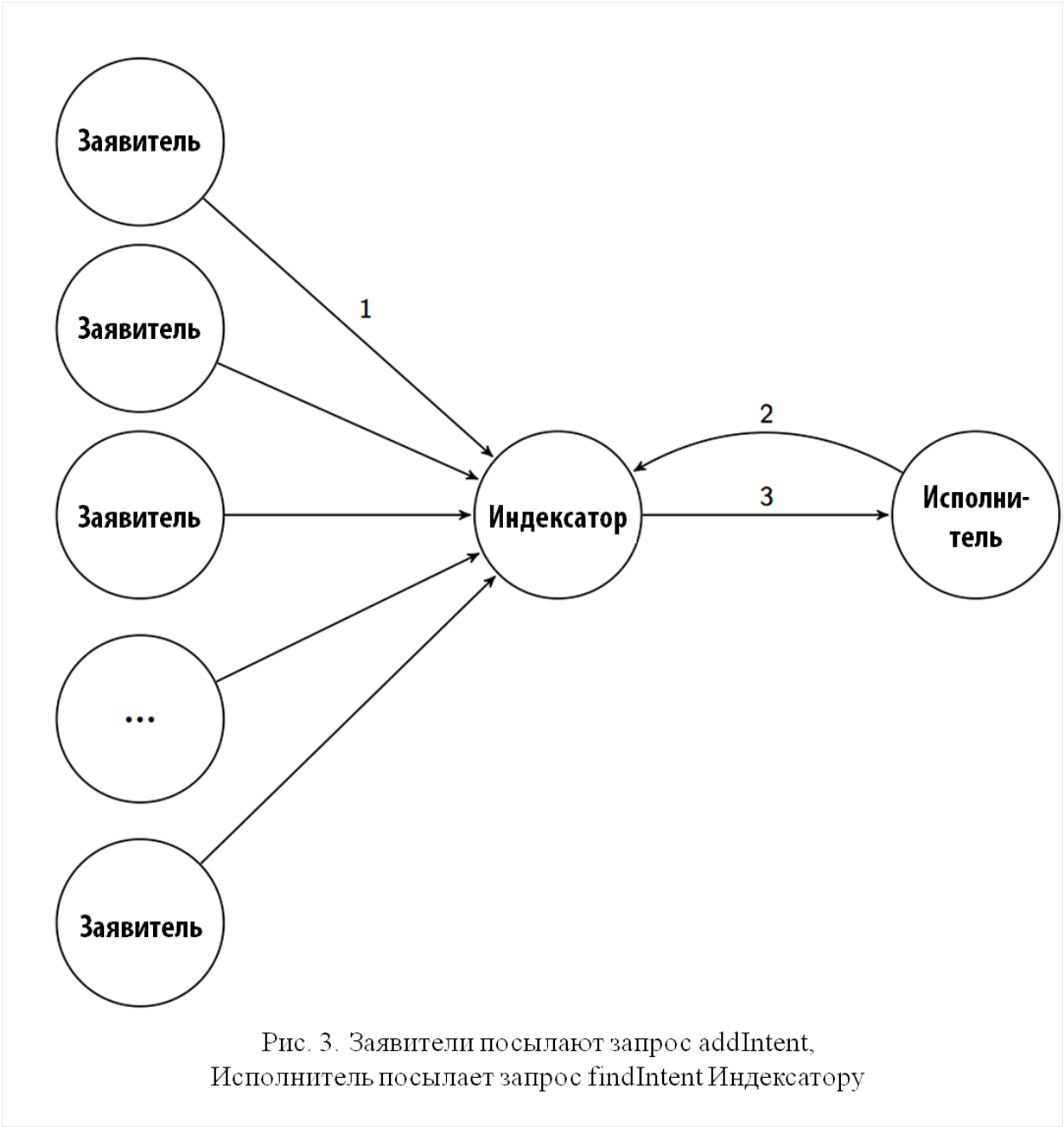

Рис. 3. Заявители посылают запрос addIntent,
Исполнитель посылает запрос findIntent Индексатору

1. Несколько заявителе вызывают Индексатора, посылая запрос *addIntent*.
2. Исполнитель вызывает Индексатора, посылая запрос *findIntent*.
3. Индексатор вызывает Исполнителя, предоставляя ему список Заявителей, сообщивших о соответствующих намерениях: *foundIntent(maker)*.

После того, как Исполнитель находит подходящих Заявителей, он может воспользоваться интерфейсом программирования заявок (Order API), запрашивая заявки у каждого из Заявителей с тем, чтобы сравнить поступившие предложения. Если Исполнитель решает ис-

полнить ту или иную заявку, он исполняет заявку (fillOrder) посредством вызова смарт-контракта.

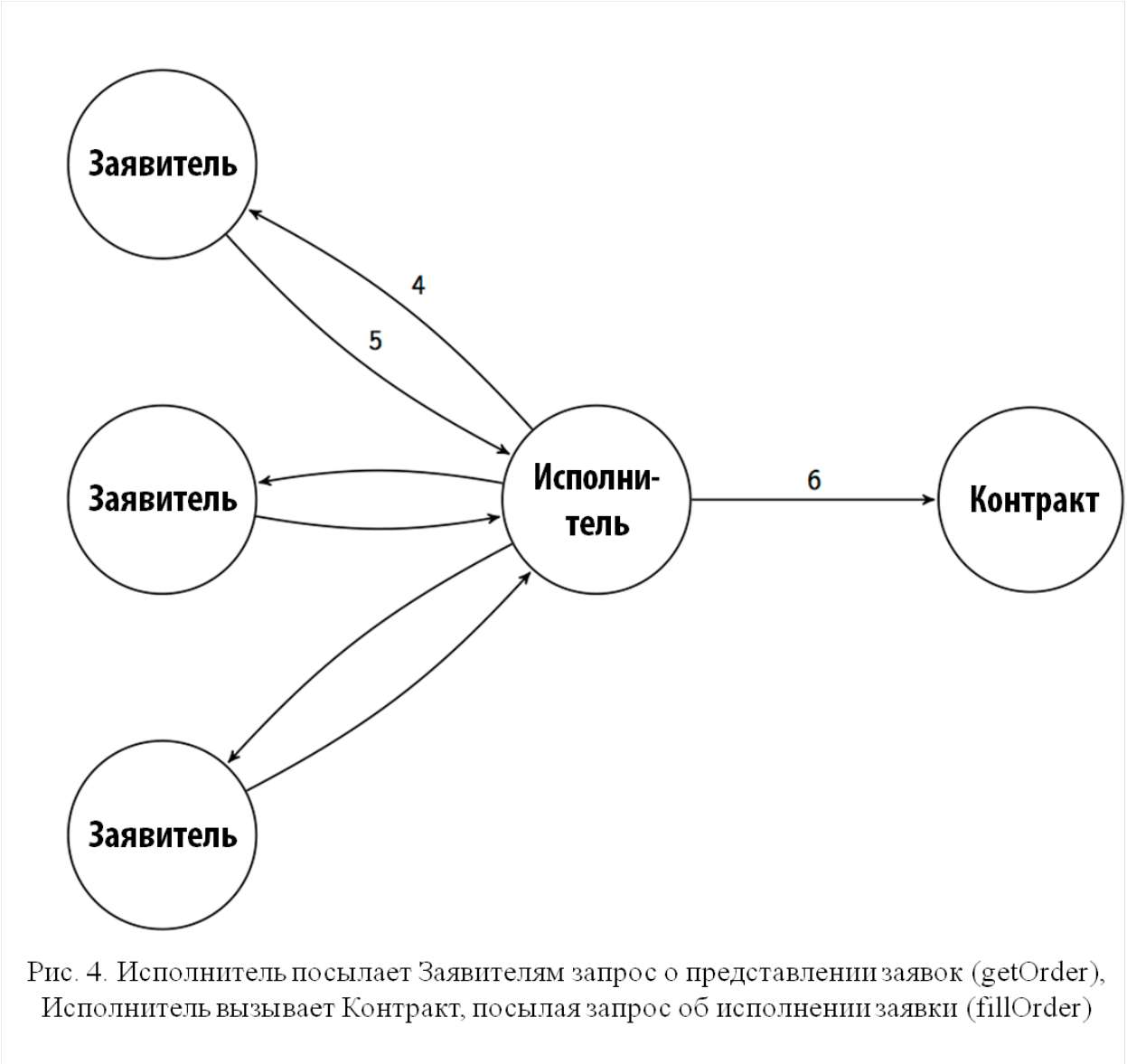

Рис. 4. Исполнитель посылает Заявителям запрос о представлении заявок (getOrder), Исполнитель вызывает Контракт, посылая запрос об исполнении заявки (fillOrder)

1. Исполнитель рассылает нескольким Заявителям запрос о представлении заявок: *getOrder*.
2. Заявители отвечают, представляя заявки.
3. Исполнитель выбирает заявку и вызывает Контракт, чтобы исполнить заявку: *fillOrder(order)*.

Прикладной интерфейс программирования Индексатора (Indexer API)

Прикладной интерфейс программирования Индексатора (Indexer API) позволяет управлять информацией о намерениях торговать, которой обмениваются одноранговые узлы. Одноранговые узлы и Индексатор обмениваются следующими вызовами.

addIntent
(сообщение о намерении торговать)

addIntent(makerToken, takerTokens)
addIntent(токен Заявителя, токены Исполнителя)

Сообщение о намерении купить или продать некоторое количество токенов.

```
Пример: “I want to trade GNO for BAT.”
«Я желаю продать токены GNO за токены BAT».
addIntent(‘GNO’, [‘BAT’])
```

removeIntent
(удаление сообщения о намерении торговать)

removeIntent(makerToken, takerTokens)
removeIntent(токен Заявителя, токены Исполнителя)

Удаление сообщения о намерении торговать токенами.

```
Пример: “I am no longer interested in trading GNO for BAT.”
«Меня больше не интересует покупка токенов GNO за токены BAT».
removeIntent(‘GNO’, [‘BAT’])
```

getIntent
(получение сообщения о намерении торговать)

getIntent(makerAddress)
getIntent(адрес Заявителя)

Список действительных сообщений о намерении торговать, относящихся к указанному адресу.

```
Пример: “List the tokens that [makerAddress] wants to trade.”
«Список токенов, которыми [адрес Заявителя] желает торговать».
getIntent(<makerAddress>)
getIntent(<адрес Заявителя>)
```

findIntent
(поиск сообщений о намерении торговать)

findIntent(makerToken, takerToken)
findIntent(токен Заявителя, токен Исполнителя)

Поиск корреспондента, желающего торговать конкретными токенами.

```
Пример: “Find someone trading GNO for BAT.”
«Найти кого-либо, кто желает купить токены GNO за токены BAT».
findIntent(‘GNO’, ‘BAT’)
```

foundIntent
(результаты поиска корреспондента, желающего торговать токенами.

foundIntent(makerAddress, intentList)
foundIntent(адрес Заявителя, список желающих торговать)

Индексатор нашел корреспондента, желающего торговать.

```
Пример: “Found someone selling 10 GNO for BAT.”
«Найден корреспондент, продающий 10 токенов GNO за токены BAT».
foundIntent(<makerAddress>, [{ makerAmount: 10, makerToken: 'GNO', takerTokens:
['BAT'] })
foundIntent(<адрес Заявителя>, [{ сумма Заявителя: 10, токен Заявителя: 'GNO', токены
Исполнителя: ['BAT'] })
```

Протокол связи с Оракулом

Оракул (Oracle) — функционирующий вне блокчейна сервисе, предоставляющий Заявителям и Исполнителям информацию о ценах. Оценивая заявку перед тем, как представить ее Исполнителю, Заявитель может запросить у Оракула рекомендуемую справедливую цену. Сходным образом, получив заявку, Исполнитель может послать Оракулу запрос о подтверждении справедливости указанной в заявке цены. Оракул предоставляет эту информацию о ценах с тем, чтобы помочь как Заявителю, так и Исполнителю принимать более информированные решения и способствовать скорейшему завершению переговоров о торговле токенами.

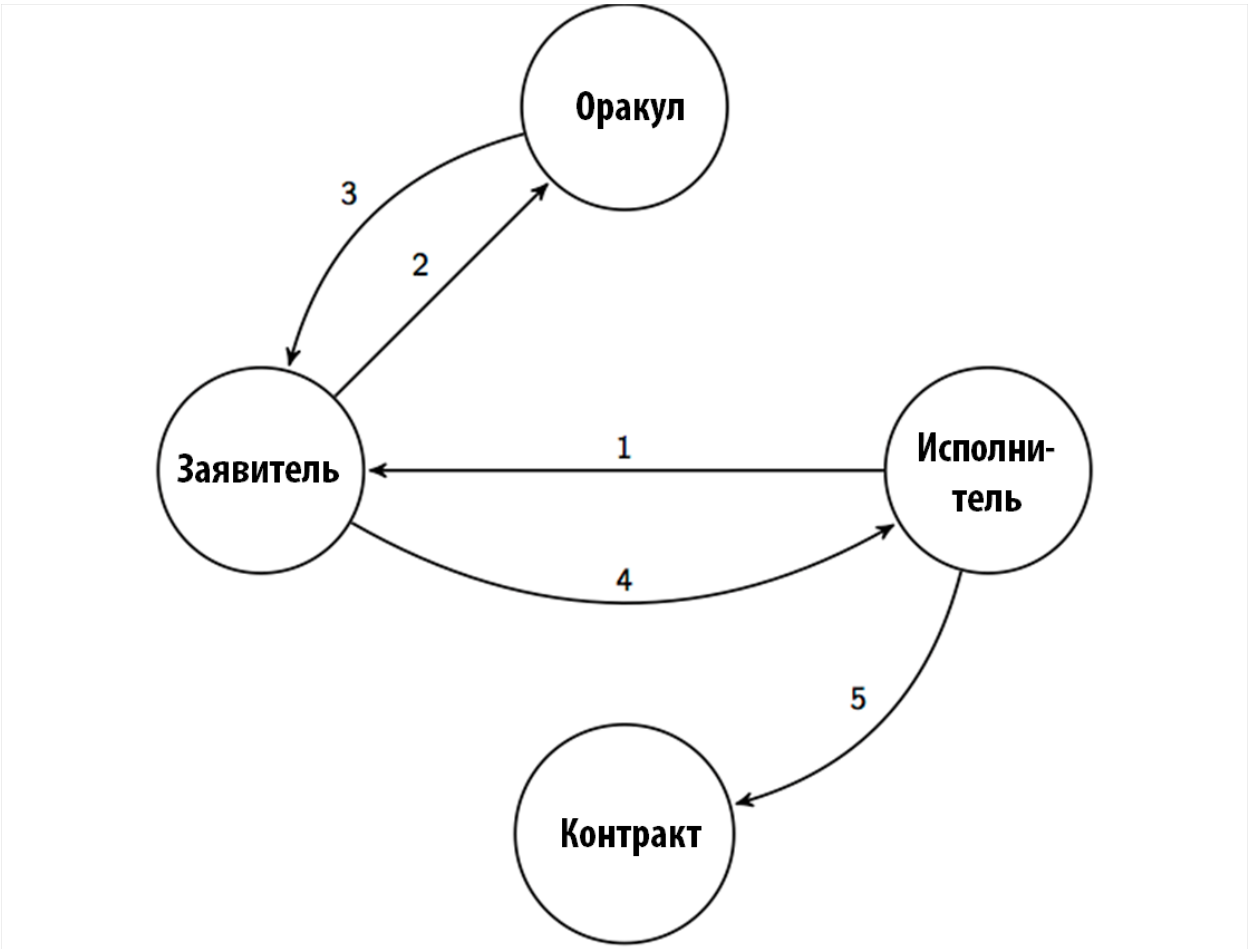

Рис. 5. Заявитель посылает запрос Оракулу перед предоставлением заявки

1. Исполнитель вызывает Заявителя, запрашивая заявку: *getOrder*.
2. Заявитель вызывает Оракула, запрашивая цену: *getPrice*.
3. Оракул сообщает Заявителю цену.
4. Заявитель, проанализировавший информацию о ценах, отвечает, предоставляя заявку.
5. Исполнитель выбирает заявку и вызывает Контракт, чтобы исполнить заявку: *fillOrder(order)*.

Сходное взаимодействие имеет место между Исполнителем и Оракулом после того, как Исполнитель получает заявку.

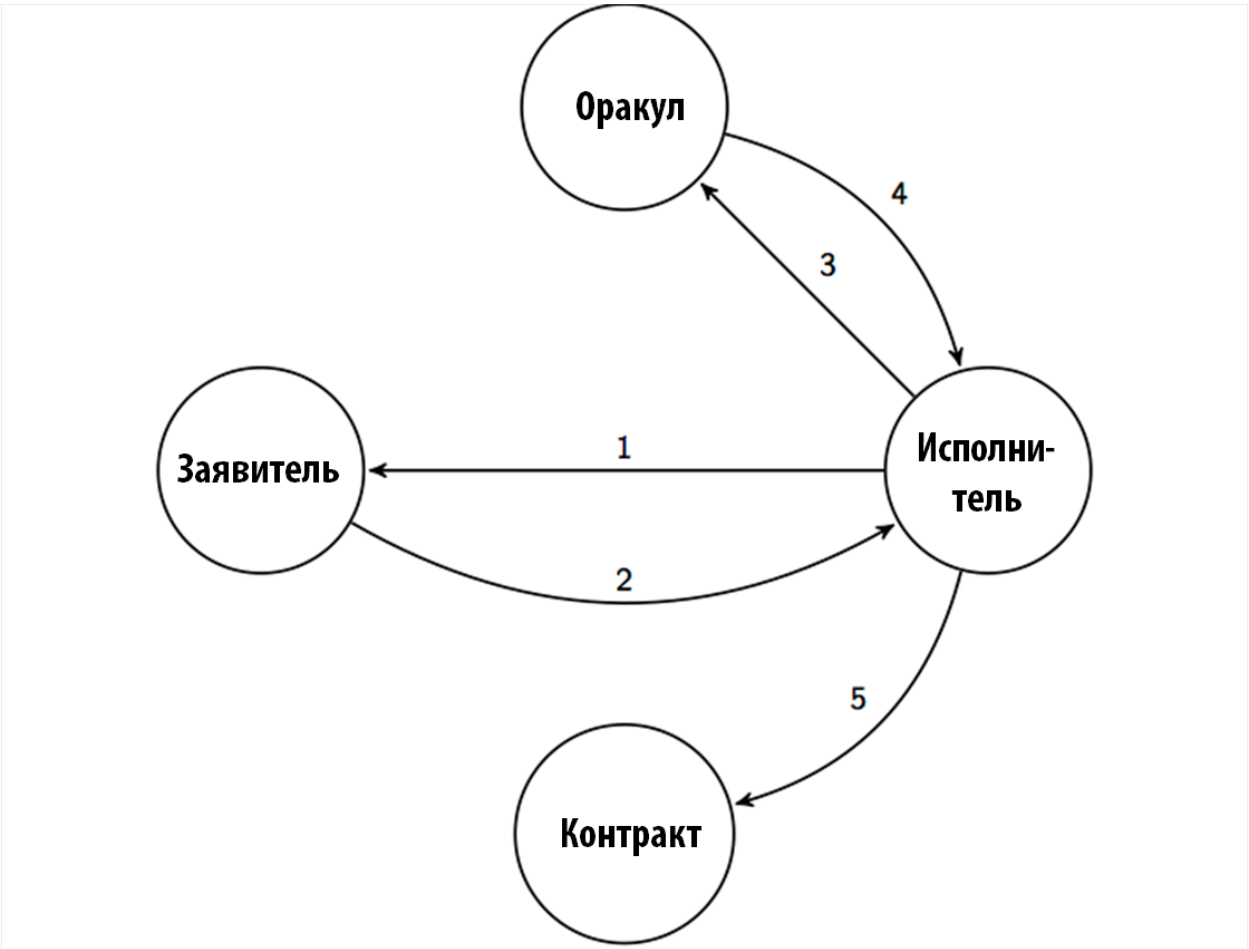

Рис. 6. Исполнитель посылает запрос Оракулу перед исполнением заявки

1. Исполнитель вызывает Заявителя, запрашивая заявку: *getOrder*.
2. Заявитель отвечает, предоставляя заявку.
3. Исполнитель вызывает Оракула, запрашивая цену: *getPrice*.
4. Оракул сообщает Исполнителю цену.
5. Исполнитель, проанализировавший информацию о ценах, вызывает Контракт, чтобы исполнить заявку: *fillOrder(order)*.

Прикладной интерфейс программирования Оракула (Oracle API)

Прикладной интерфейс программирования Оракула (Oracle API) используется Заявителями и Исполнителями с целью определения цен. Оракул сообщает рекомендуемые цены, не подлежащие обязательному исполнению.

getPrice
(запрос о рекомендации цены)

getPrice(makerToken, takerToken)
getPrice(токен Заявителя, токен Исполнителя)

Исполнитель или Заявитель вызывает Оракула, чтобы получит информацию о цене.

```
Пример: “What is the current price of GNO for BAT?”
«Сколько токенов BAT дают в настоящее время за токен GNO?»
getPrice(‘GNO’, ‘BAT’)
```

providePrice
(предоставление цены)

providePrice(makerToken, takerToken, price)
providePrice(токен Заявителя, токен Исполнителя, цена)

Оракул вызывает Заявителя или Исполнителя, чтобы сообщить цену.

```
Пример: “The current price of GNO for BAT is 0.5.”
«Текущее соотношение цены токена GNO к цене токена BAT составляет 0,5».
providePrice(‘GNO’, ‘BAT’, 0.5)
```

Смарт-контракт

Смарт-контракты («умные контракты») в системе Эфириум используются в целях исполнения или отмены заявок.

fillOrder(makerAddress, makerAmount, makerToken, takerAddress, takerAmount, takerToken, expiration, nonce, signature)
fillOrder(адрес Заявителя, сумма Заявителя, токен Заявителя, адрес Исполнителя, сумма Исполнителя, токен Исполнителя, дата истечения срока действия заявки, отметка текущего времени, подпись)

Единичный своп токенов (обмен токенами), вызываемый Исполнителем. Контракт обеспечивает соответствие отправителя сообщения Исполнителю, а также то, что указанный срок действия заявки еще не истек. Для того, чтобы заявки могли быть исполнены, одноранговые узлы должны предварительно утвердить перечисление на их счета токенов в таких количествах, которые позволяют изъять как минимум указанные в контракте суммы. Для того, чтобы стороны обменялись токенами, рассылается вызов с запросом о передаче соответствующих токенов по контракту: transferFrom. После успешного завершения выполнения этой функции в блокчейн загружается сообщение о событии исполнения заявки (Filled).

```
Пример: “I want to fill this order of 5 GNO for 10 BAT.”
«Я желаю исполнить эту заявку о покупке 5 токенов GNO за 10 токенов BAT».
fillOrder([maker], 5, ‘GNO’, [taker], 10, ‘BAT’, [expiration], [signature])
fillOrder([Заявитель], 5, ‘GNO’, [Исполнитель], 10, ‘BAT’, [дата истечения срока действия заявки], [подпись])
```

cancelOrder(makerAddress, makerAmount, makerToken, takerAddress, takerAmount, takerToken, expiration, nonce, signature)
cancelOrder(адрес Заявителя, сумма Заявителя, токен Заявителя, адрес Исполнителя, сумма Исполнителя, токен Исполнителя, дата истечения сроке действия заявки, отметка текущего времени, подпись)

Отмена заявки, уже полученной Исполнителем, но еще не исполненной, вызывается запросом соответствующего Заявителя. Отмена заявки приводит к тому, что заявка регистрируется, как уже исполненная в соответствии с контрактом, в связи с чем последующие попытки исполнить эту заявку приводят к отказу. После успешного завершения выполнения этой функции в блокчейн загружается сообщение о событии отмены заявки (Canceled).

```
Пример: “I want to cancel this order of 5 GNO for 10 BAT.”
«Я желаю отменить заявку на покупку 5 токенов GNO за 10 токенов BAT».
cancelOrder([maker], 5, ‘GNO’, [taker], 10, ‘BAT’, [expiration], [signature])
cancelOrder([Заявитель], 5, ‘GNO’, [Исполнитель], 10, ‘BAT’, [дата истечения срока действия заявки], [подпись])
```

Заявки на торговлю токенами Ether

Смарт-контракты («умные контракты») поддерживают торговлю токенами ether («эфирами», ETH). Если в заявке указан нулевой адрес токена Исполнителя (takerToken = 0x0), смарт-контракт проверяет стоимость в эфирах, которая была указана в полученном функциональном вызове, и перечисляет соответствующую сумму Заявителю от имени Исполнителя.

Заключение

Протокол Своп (Swap) позволяет удовлетворять растущий спрос на децентрализованный обмен активами в рамках сети Эфириум (Ethereum). Несмотря на новизну реестров заявок на базе блокчейна и их несомненное соответствие этическим нормам нашей экосистемы, им свойственны ограничения, которые, как мы считаем, в конечном счете будут препятствовать их конкуренции с доступными в настоящее время централизованными решениями тех же задач. Протокол Своп (Swap) обеспечивает возможность применения децентрализованного метода, на который не влияют такие ограничения.

Внедрив этот протокол, участники получают доступ к масштабируемому, приватному (конфиденциальному) и обеспечивающему равноправие методу регулирования ликвидности, не поступаясь доступом к выгодным ценам. Наш протокол и его интерфейсы прикладного программирования (API) расширяемы, и мы рекомендуем сообществу создавать приложения на их основе. Мы приветствуем отзывы и надеемся развивать сообщество Эфириума вместе с вами.

С вопросами, замечаниями и отзывами, пожалуйста, обращайтесь к нам по адресу team@swap.tech.