Своп (Swap): одноранговий протокол торгівлі токенами на платформі Етереуму (Ethereum)

Веб-сайт протоколу Своп
Англ. Кит. Яп.
Майкл Овед, Дон Мосайтс
Опубліковано 21 червня 2017 р.

Короткий огляд

Ми пропонуємо вашій увазі однорангову методика торгівлі токенами стандарту ERC20 на платформі блокчейну Етереум. Перш за все, окреслимо обмеження, пов'язані з використанням реєстрів замовлень у блокчейні, та запропонуємо вагому альтернативу однорангової торгівлі токенами: торги поза блокчейном та укладення угоди в блокчейні. Далі надамо опис протоколу, за допомогою якого сторони можуть інформувати інших про намір торгувати токенами. Після підімкнення до мережі контрагенти можуть вільно надавати повідомлення про ціни та передавати один одному замовлення. В ході цього процесу сторони можуть надавати запити на ціни незалежним сторонам («оракулам») з метою перевірки достовірності даних. І, нарешті, ми представимо «розумний контракт» (смарт-контракт) на платформі Етереуму для виконання замовлень з використанням блокчейну Етереум.

Зміст

Вступ

Кількість цифрових активів на платформі Етереуму за останні дванадцять місяців різко зростає, оскільки дедалі більша кількість сценаріїв взаємодії реалізується як смарт-контракти. Наша теза полягає в тому, що ця тенденція триватиме і в майбутньому; тому ми вважаємо, що це зростання призведе до збільшення попиту на операції обміну (свопінгу) активами між користувачами, котрі переходять від одного способу взаємодії до іншого або перебалансовують свої портфелі токенів. Обмінам, заснованим на реєстрі замовлень в блокчейні, притаманні певні системні обмеження, багато з котрих можна подолати за допомогою дизайнерських рішень, окреслених у цьому документі. Ми намагаємося запропонувати альтернативу реєстрам замовлень у блокчейні, визначивши набір протоколів, які розблоковують ліквідність активів та сприяють безперешкодному розвиткові екосистеми Етереум без таких обмежень.

Реєстри замовлень (Order Books)

Реєстри замовлень – це високоавтоматизований спосіб узгодження пропозиції та попиту на певний актив, що підлягає вільному торговельному обміну. За традицією, вони є централізованими та використовуються у поєднанні з протоколом виконанням замовлень, що дозволяє створювати, виконувати та скасовувати замовлення шляхом зв'язку з центральним джерелом достовірної інформації. Реєстри замовлень були перероблені в дусі децентралізації таким чином, аби ними можна було користуватися у блокчейнах. Проте, запровадження реєстру замовлень у блокчейні містить в собі декілька обмежень.

Реєстри замовлень у блокчейні не масштабуються. Виконання коду з використанням блокчейну призводить до витрат, тому вартість автоматичного циклу замовлення-скасування замовлення-замовлення швидко зростає і зводить нанівець дієвість реєстру замовлень як високопродуктивної автоматичної системи узгодження попиту і пропонування. І насправді, якщо такий алгоритм узгодження виконується в блокчейні, то сторона, що розміщує замовлення, понесе витрати на виконання, що суттєво збільшуватимуться із зростанням обсягу реєстру замовлень.

Реєстри замовлень у блокчейні є загальнодоступними. Оскільки процесінг транзакцій по створенню замовлень у блокчейні здійснюється майнерами, ці майнери дізнаються по існування замовлення раніше, ніж воно публікується в реєстрі. Це створює можливість укладання угоди по транзакції на базі інформації, ще не доступної іншим сторонам, що може суттєво вплинути на первинне замовлення. Окрім того, оскільки доступ до опублікованих замовлень є вільним, ціна виконання замовлення є однаковою для всіх, що виключає можливість постачальника адаптувати ліквідність залежно від попиту.

Реєстри замовлень в блокчейні не гарантують рівні права. Фізично розподіленим системам притаманне запізнення зв'язку між їхніми вузлами. Оскільки майнери розподілені географічно, більш оснащені сторони можуть бути в змозі розміщувати,

виявляти замовлення та випереджувати затримку, притаманну блокчейнам, тобто ефективно діяти на основі інформації про замовлення раніше, ніж це зроблять інші сторони. Така інформаційна асиметрія може цілком ймовірно позбавити бажання у менш оснащених сторін взагалі брати участь в екосистемі.

Одноранговий протокол (P2P)

Як альтернатива, система однорангової торгівлі дозволяє окремим сторонам торгувати одна з одною безпосередньо. Більшість транзакцій, які ми виконуємо щоденно, є одноранговими – наприклад, купівля кави в кафе, продаж взуття на інтернет-аукціоні eBay або купівля котячого харчу на веб-сайті Amazon. Оскільки це є приватними транзакціями між окремими особами чи організаціями, кожна сторона знає і, в остаточному рахунку, вибирає, з ким здійснювати транзакції.

Однорангова торгівля масштабується. Замовлення передаються від однієї сторони до іншої і виконуються одна за одною, на відміну від замовлень на публічній біржі без гарантії їх повного виконання. Тому скасування в реєстрі замовлень стає регулярним явищем, в той час як замовлення в одноранговій мережі, як правило, виконуються, оскільки вони пропонуються і надаються сторонам, які вже висловили свою зацікавленість. Окрім того, узгодження пропозиції та попиту в одноранговому режимі можна забезпечити за допомогою досить простого виявлення однорангових партнерів, на відміну від алгоритмічної координації інтересів сторін, що вимагає великих витрат – як в блокчейні, так і поза блокчейном.

Однорангова торгівля забезпечує конфіденційність даних. Після того як дві сторони вирішили торгувати одна з одною, участь у переговорах від третіх сторін не вимагається. Зв'язок між цими сторонами залишається приватним до кінця переговорів, що позбавляє інші сторони можливості отримувати переваги за рахунок процесу реєстрації замовлень. До замовлення буде відкритий доступ лише тоді, коли воно буде подано до виконання.

Однорангова торгівля забезпечує рівні права. Оскільки замовлення створюються та передаються безпосередньо однією стороною іншій, сторонні учасники не можуть мати жодних переваг. Оскільки вони працюють з кількома незалежними сторонами, учасники можуть отримувати ціни, гідні порівняння або більш вигідні ніж ті, які їм могли б запропонувати на біржі. Окрім того, ті, хто встановлює ціни за виконання замовлень, можуть робити це агресивно, не боячись, що інші зможуть скористатися з автоматизованих стратегій отримання переваг за рахунок затримки зв'язку.

Обмеження масштабованості, конфіденційності та справедливості, що накладають реєстри замовлень у блокчейні, викликали потребу в альтернативі. Сьогоднішня екосистема Етереуму потребує відкритого однорангового рішення питання обміну активами.

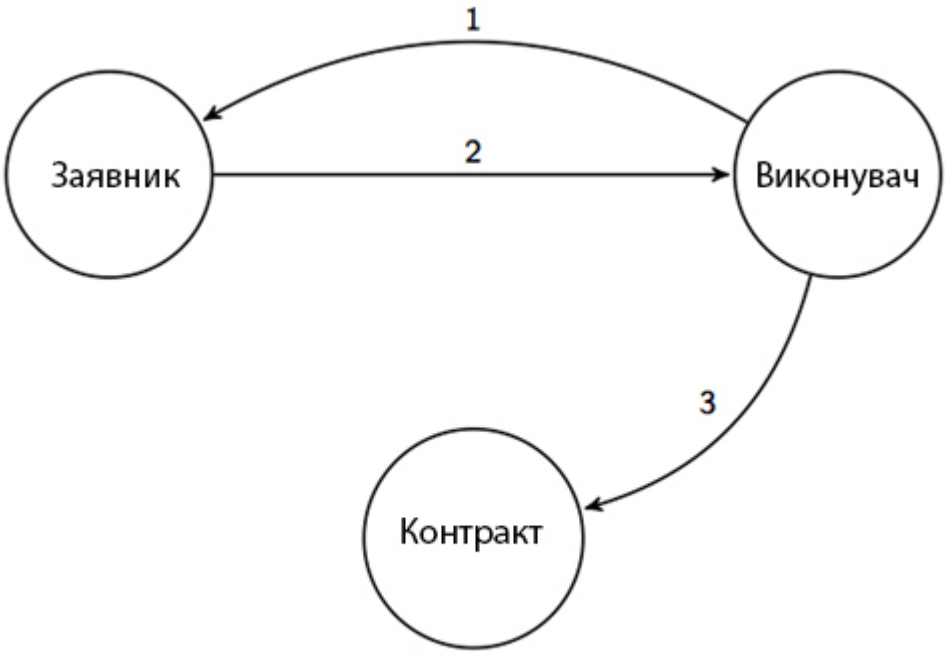
Ознайомлення з протоколом Своп

Своп – це протокол, що сприяє впровадженню надійної однорангової екосистеми торгівлі токенами у блокчейні Етереуму. Надаємо нижче опис розширюваної специфікації, яка підтримує ефективне виявлення кореспондентів і проведення переговорів між ними. Ці протоколи покликані стати основою екосистеми торгівлі активами та прискорити розширення екосистеми Етереуму. Публікуючи цей документ і пропонуючи його обговорення, ми сподіваємося отримати зауваження від сторін, зацікавлених у розвитку екосистеми, з метою розробки високоякісних протоколів, що дозволить створити широке розмаїття прикладних програм для застосування у реальному середовищі.

Протокол зв'язку між одноранговими вузлами

При наявності лише декількох повідомлень, що кореспонденти передають один одному, торги можна обговорювати швидко, на рівноправній основі та в умовах конфіденційності. Для цілей цього документа «Заявником» (Maker) є сторона, яка надає замовлення, а «Виконувачем» (Taker) – стороною, яка виконує його. Оскільки сторони користуються вузлами одного і того самого рангу, кожна із сторін може в будь-який час взяти на себе роль Заявника або Виконувача. Токени в наданому нижче описі є сумісними із стандартом ERC20, і будь-який токен, який відповідає цьому стандарту, може бути об'єктом торгу з використанням цього протоколу.

Надаємо нижче схему послідовності операцій, що виконуються за основним протоколом. Заявник і Виконувач узгоджують умови транзакції поза блокчейном. «Контракт», що згадується нижче – це смарт-контракт на платформі Етереума, який викликається Виконувачем тоді, коли він готовий виконати замовлення у блокчейні.



Мал. 1. Запит, надання і виконання замовлення

1. Виконувач викликає Заявника – *getOrder*.
2. У відповідь Заявник надає замовлення.
3. Виконувач викликає Контракт – _fillOrder()_.

Прикладний інтерфейс програмування замовлень (Order API)

Наступні прикладні інтерфейси програмування замовлень (API) – це віддалені виклики процедур (remote procedure calls, RPC) без прийняття до уваги транспортного рівня, які використовуються для встановлення зв'язку між одноранговими вузлами мережі та сервісами. В прикладах замість адрес використовуються так звані «тікери», тобто скорочені позначення токенів, але для фактичних викликів потрібні адреси токенів, що відповідають стандарту ERC20. Підписи викликів, що надаються нижче, вживаються з метою обговорення, оскільки подальші технічні деталі будуть опубліковані в окремому документі.

Прикладний інтерфейс програмування замовлень використовується поза блокчейном і визначає асинхронні виклики, якими обмінюються кореспонденти під час узгодження умов транзакції. Виконавець може за бажанням вибрати цикл «запит-надання замовлення» як синхронний «запит-відповідь». Оскільки замовлення підписується Заявником, Виконувач може пізніше подати його для виконання на смарт-контракт.

getOrder (запит на надання замовлення)

getOrder(makerAmount, makerToken, takerToken, takerAddress)

getOrder(сума Заявника, токен Заявника, токен Виконувача, адреса Виконувача)

Виконувач викликає Заявника і дає запит на замовлення торгувати токенами.

```
Приклад: “I want to buy 10 GNO using BAT.”
«Я хочу купити 10 токенів GNO, використовуючи BAT».
getOrder(10, ‘GNO’, ‘BAT’, <takerAddress>)
getOrder(10, ‘GNO’, ‘BAT’, <адреса Виконувача>)
```

provideOrder (надання замовлення)

provideOrder(makerAddress, makerAmount, makerToken, takerAddress, takerAmount, takerToken, expiration, nonce, signature)
provideOrder(адреса Заявника, сума Заявника, токен Заявника, адреса Виконувача, сума Виконувача, токен Виконувача, дата спливання терміну дії, відмітка поточного часу, підпис)

Виклик Виконувача Заявником з наданням підписаного замовлення, готового для виконання.

```
Приклад: “I’ll sell you 10 GNO for 5 BAT.”
«Я продам вам 10 токенів GNO за 5 токенів BAT».
provideOrder(<makerAddress>, 10, ‘GNO’, <takerAddress>, 5, ‘BAT’, <expiration>, <nonce>, <signature>)
provideOrder(<адреса Заявника>, 10, ‘GNO’, <адреса Виконувача>, 5, ‘BAT’, <спливання терміну дії>, <відмітка поточного часу>, <підпис>)
```

Прикладний інтерфейс програмування котирування (Quote API)

Котирування призначаються для надання сторонами інформації про ціну і не підлягають виконанню. Котирування можна пізніше перетворити на замовлення, якщо умови задовольняють обох кореспондентів.

getQuote (запит на надання котирування)

getQuote(makerAmount, makerToken, takerTokens)
getQuote(сума Заявника, токен Заявника, токени Виконувача)

Виконувач викликає Заявника і дає запит на котирування по певним токенам.

```
Приклад: “How much would it cost to buy 10 GNO using BAT?”
«Скільки коштуватиме покупка 10 токенів GNO в токенах BAT?»
getQuote(10, ‘GNO’, [‘BAT’])
```

provideQuote (надання котирування)

provideQuote(makerAmount, makerToken, takerAmounts)
provideQuote(сума Заявника, токен Заявника, суми Виконувача)

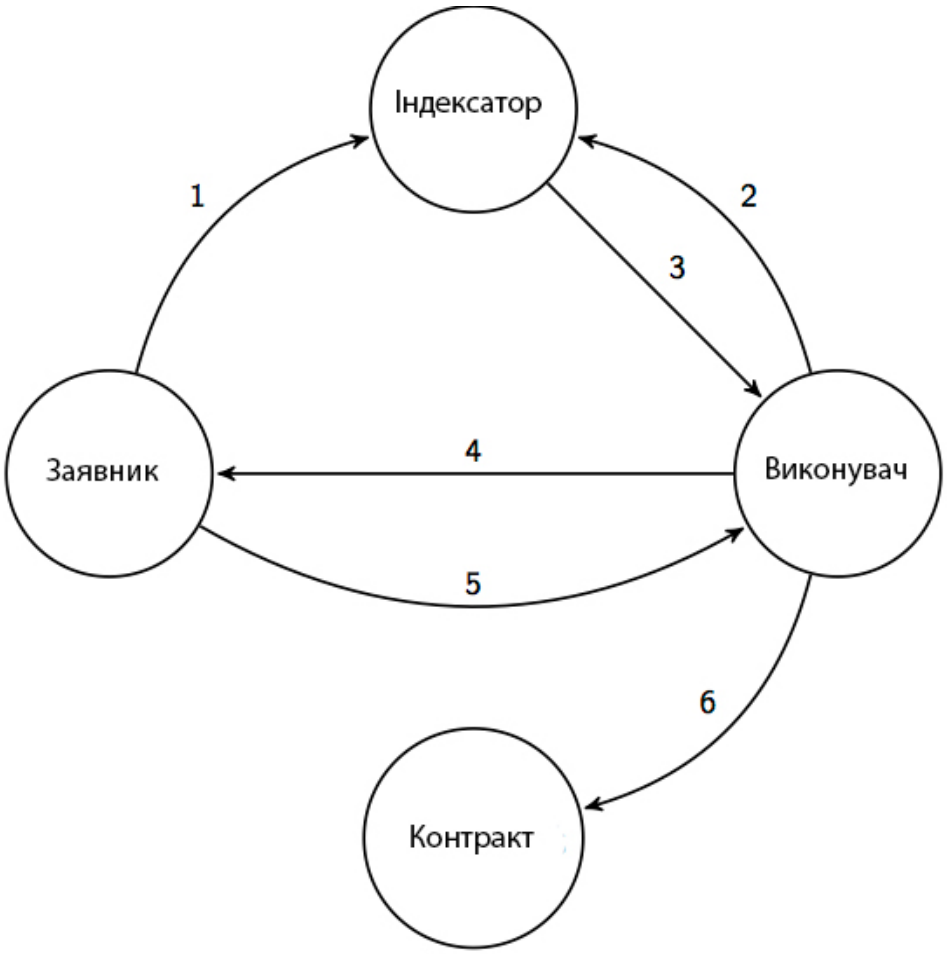
Виконувач викликає Заявника і надає котирування в токенах Виконувача.

```
Приклад: “It will cost you 5 BAT for 10 GNO.”
«Покупка 10 токенів GNO коштуватиме вам 5 токенів BAT".
provideQuote(10, ‘GNO’, { ‘BAT’: 5 })
```

Протокол зв'язку з Індексатором

Індексатор є сервісом поза блокчейном, що групує та узгоджує однорангові вузли зв'язку з урахуванням їхнього наміру торгувати, тобто того, чи мають бажання потенційні Заявники та Виконувачі купувати чи продавати токени. Індексатори є сервісам, що функціонують поза блокчейном, які накопичують інформацію про такий «намір торгувати» та сприяють узгодженню однорангових вузлів зв'язку з урахуванням їхнього наміру купувати чи продавати ті чи інші токени. Багато потенційних Заявників можуть повідомляти про намір торгувати, і коли Виконувач надсилає запит Індексатору знайти підходящих кореспондентів, він може отримати декілька результатів цього запиту. Після того, як Виконувач знайде Заявника, з яким він хотів би торгувати, розпочинаються переговори з використанням вищезазначеного Протоколу зв'язку між одноранговими вузлами. Після досягнення домовленості між Заявником і Виконувачем замовлення виконується шляхом виклику смарт-контракту.

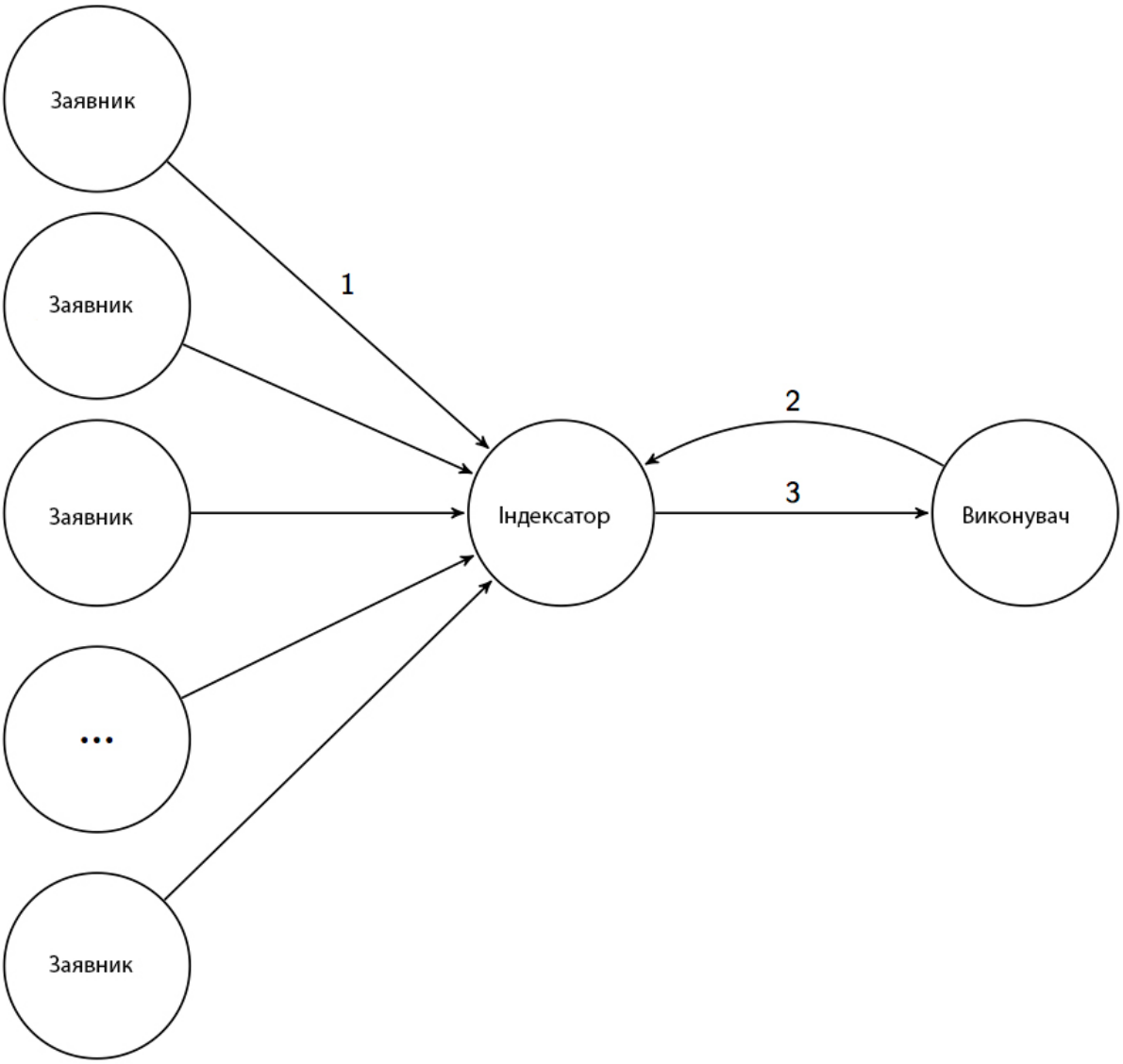
Взаємодії між Заявником, Виконувачем та Індексатором показані на схемі, що надається нижче. Всі з них – Заявник, Виконувач та Індексатор – працюють поза блокчейном і спілкуються через будь-яке середовище передачі повідомлень за своїм вибором.



Мал. 2. Пошук кореспондента і укладення угоди

1. Заявник викликає Індексатора – *addIntent*.
2. Виконувач викликає Індексатора – *findIntent*.
3. Індексатор викликає Виконувача – *foundIntent(maker)*.
4. Виконувач викликає Заявника – *getOrder*.
5. Заявник надає у відповідь замовлення.
6. Виконувач викликає Контракт – *fillOrder(order).*

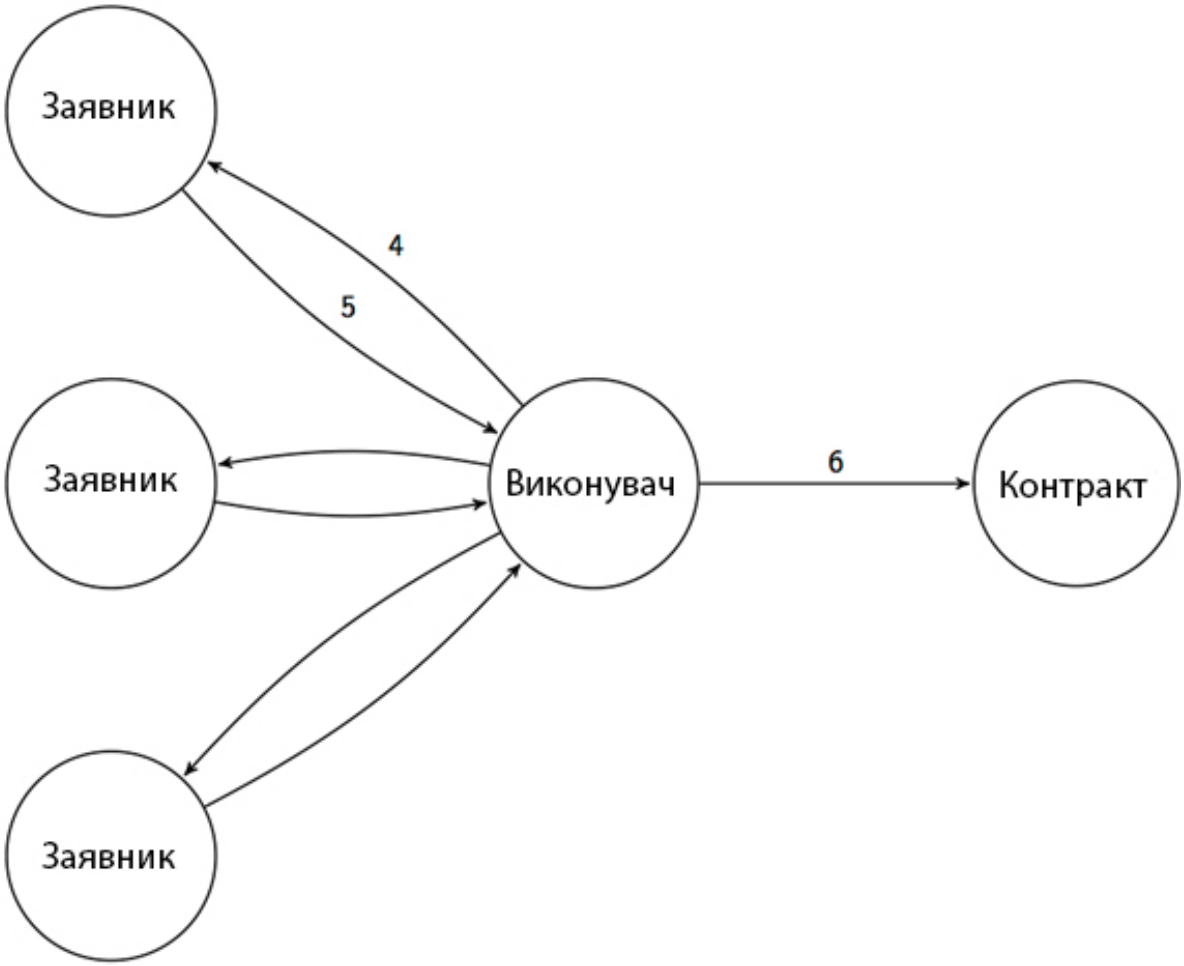
Взаємодію між декількома Заявниками, Виконувачем та Індексатором показано на схемі нижче. Кожен Заявник самостійно оголошує свій намір. Виконувач дає запит на пошук Заявників з певним наміром, а Індексатор відповідає, надаючи список адрес та даних в системі Етереум.



Мал. 3. Заявники надсилають запит addIntent.
Виконувач насилає Індексатору запит findIntent

1. Декілька Заявників викликають Індексатора і надсилають запит *addIntent*.
2. Виконувач викликає Індексатора і надсилає запит *findIntent*.
3. Індексатор викликає Виконувача і надає список Заявників, котрі повідомили про відповідні наміри – *foundIntent(maker)*.

Після того, як Виконувач знаходить відповідних Заявників, він зможе скористатися з інтерфейсу програмування замовлень, щоб подати запит кожному із Заявників на замовлення, аби порівняти їх одне з одним. Якщо Виконувач вирішив виконати те чи інше замовлення, він виконає його (fillOrder) шляхом виклику смарт-контракту.



Мал. 4. Виконувач надсилає Заявнику запит про надання замовлень (getOrder). Виконувач викликає Контракт і надсилає запит про виконання замовлення (fillOrder)

1. Виконувач надсилає декільком із Заявників запит на надання замовлень – *getOrder*.
2. Заявники надають у відповідь замовлення.
3. Виконувач вибирає замовлення і викликає Контракт**,** щоб виконати замовлення – *fillOrder(order).*

Прикладний інтерфейс програмування Індексатора (Indexer API)

Прикладний інтерфейс програмування Індексатора здійснює керівництво інформацією про намір торгувати, якою обмінюються однорангові вузли. Однорангові вузли надають один одному такі виклики:

addIntent (повідомлення про намір торгувати)

addIntent(makerToken, takerTokens)
addIntent(токен Заявника, токени Виконувача)

Повідомлення про намір купити чи продати певну кількість токенів.

```
Приклад: “I want to trade GNO for BAT.”
«Я хочу продати токени GNO за токени BAT».
addIntent(‘GNO’, [‘BAT’])
```

removeIntent (видалення повідомлення про намір торгувати)

removeIntent(makerToken, takerTokens)
removeIntent(токен Заявника, токени Виконувача)

Видалення повідомлення про намір торгувати токенами.

```
Приклад: “I am no longer interested in trading GNO for BAT.”
«Мене більше не зацікавить покупка токенів GNO за токени BAT».
removeIntent(‘GNO’, [‘BAT’])
```

getIntent (отримання повідомлення про намір торгувати)

getIntent(makerAddress)
getIntent(адреса Заявника)

Вказати дійсні повідомлення про намір торгувати, пов'язані з адресою.

```
Приклад: “List the tokens that [makerAddress] wants to trade.”
«Вказати токени, котрими [адреса Заявника] хоче торгувати».
getIntent(<makerAddress>)
getIntent(<адреса Заявника>)
```

findIntent (пошук повідомлень про намір торгувати)

findIntent(makerToken, takerToken)
findIntent(токен Заявника, токен Виконувача)

Пошук бажаючого торгувати певними токенами.

```
Приклад: “Find someone trading GNO for BAT.”
«Знайти бажаючого купити токени GNO за токени BAT».
findIntent(‘GNO’, ‘BAT’)
```

foundIntent (бажаючого торгувати токенами знайдено)

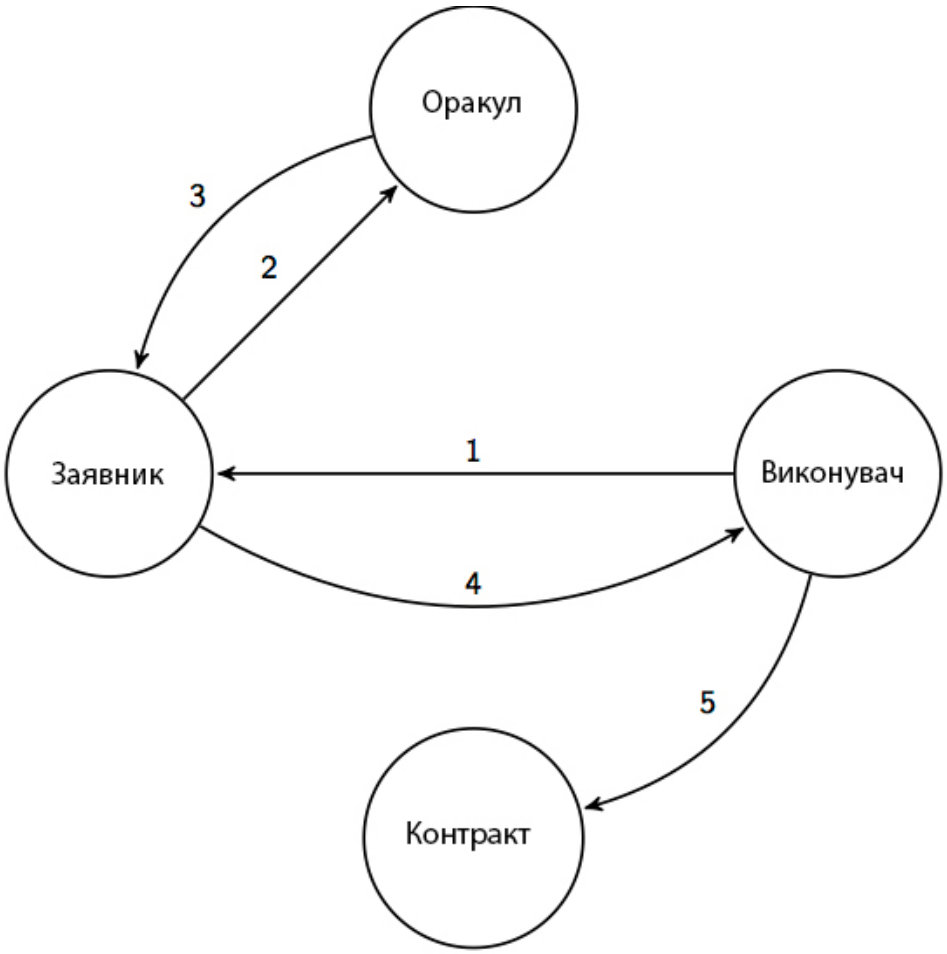
foundIntent(makerAddress, intentList)
foundIntent(адреса Заявника, список бажаючих торгувати)

Індексатор знайшов бажаючого торувати.

```
Приклад: “Found someone selling 10 GNO for BAT.”
«Знайдено бажаючого продати 10 токенів GNO за токени BAT».
foundIntent(<makerAddress>, [{ makerAmount: 10, makerToken: 'GNO', takerTokens:
['BAT'] })
foundIntent(<адреса Заявника>, [{ сума Заявника: 10, токен Заявника: 'GNO', токени
Виконувача: ['BAT'] })
```

Протокол зв'язку з Оракулом

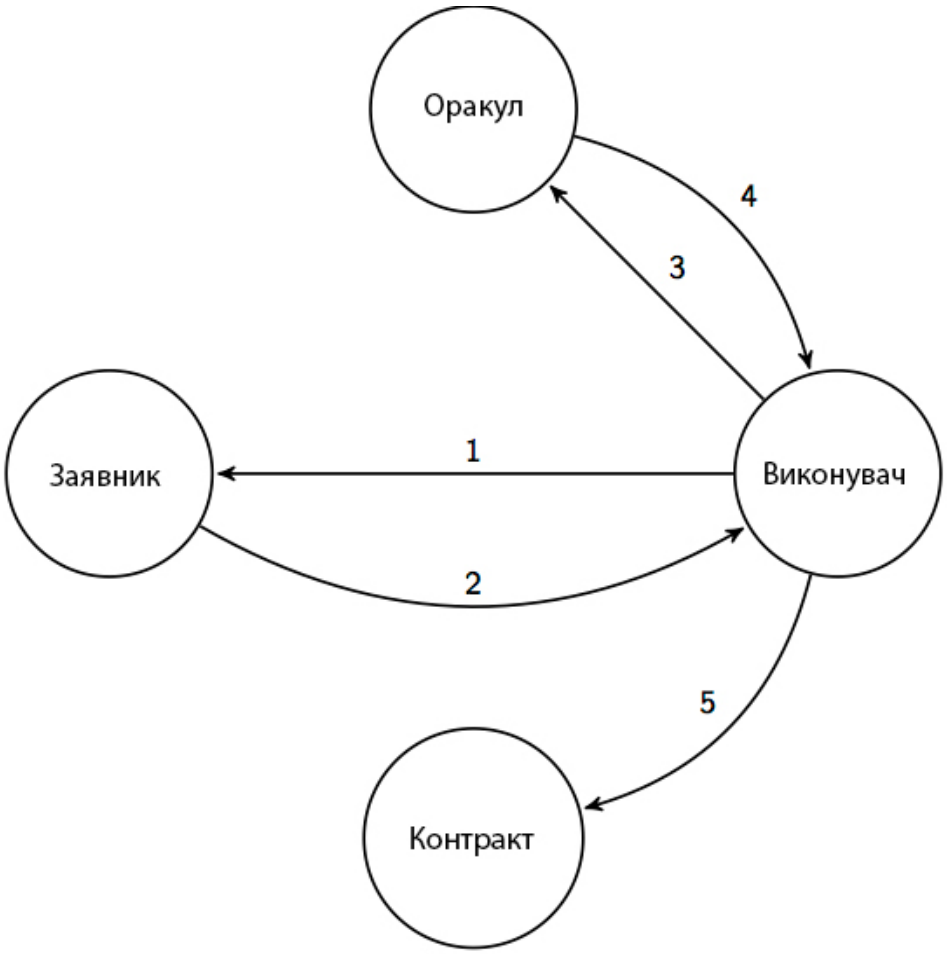
Оракул є сервісом, що знаходиться поза блокчейном і надає інформацію про ціни Заявникам і Виконувачам. Оцінюючи замовлення перш ніж надати його Виконувачу, Заявник може запитати Оракула стосовно рекомендованої справедливої ціни. Аналогічно, отримавши замовлення, Виконувач може надати запит Оракулу стосовно перевірки справедливості ціни, зазначеної в замовленні. Oracle надає цю інформацію про ціну, щоб допомогти як Заявнику, так і Виконувачу прийняти більш інформовані рішення щодо ціни та сприяти успіхові переговорів про торгівлю.



Мал. 5. Заявник надсилає Оракулу запит перед наданням замовлення

1. Виконувач викликає Заявника запитом на замовлення – *getOrder*.
2. Заявник викликає Оракула запитом на ціну – *getPrice*.
3. Оракул надає Заявнику у відповідь ціну.
4. Після аналізу цінової інформації Заявник дає у відповідь замовлення.
5. Виконувач викликає Контракт, щоб виконати замовлення – *fillOrder(order)*.

Аналогічна взаємодія відбувається між Виконувачем і Оракулом, коли Виконувач отримує замовлення.



Мал. 6. Виконувач надсилає Оракулу запит перед виконанням замовлення

1. Виконувач викликає Заявника запитом на замовлення – *getOrder*.
2. Заявник надає у відповідь замовлення.
3. Виконувач викликає Оракула запитом на ціну – *getPrice*.
4. Oracle надає Виконувачу свою ціну.
5. Після аналізу цінової інформації Виконувач викликає викликає Контракт, щоб виконати Замовлення – *fillOrder(order)*.

Прикладний інтерфейс програмування Оракула (Oracle API)

Прикладний інтерфейс програмування Оракула використовується Заявниками і Виконувачами для визначення ціни замовлення. Ціни є рекомендованими і не підлягають обов'язковому виконанню.

getPrice (запит на рекомендовані ціни)

getPrice(makerToken, takerToken)
getPrice(токен Заявника, токен Виконувача)

Виконувач або Заявник викликає Оракула, щоб отримати ціну.

```
Приклад: “What is the current price of GNO for BAT?”
«Яка поточна ціна токена GNO в токенах BAT?»
getPrice(‘GNO’, ‘BAT’)
```

providePrice (надання ціни)

providePrice(makerToken, takerToken, price)
providePrice(токен Заявника, токен Виконувача, ціна)

Оракул викликає Заявника або Виконувача, щоб надати ціну.

```
Приклад: “The current price of GNO for BAT is 0.5.”
«Поточне співвідношення ціни токена GNO до ціни токена BAT становить 0,5.»
providePrice(‘GNO’, ‘BAT’, 0.5)
```

Смарт-контракт

Смарт-контракт Етереум використовується для виконання або скасування замовлень.

fillOrder(makerAddress, makerAmount, makerToken, takerAddress, takerAmount, takerToken, expiration, nonce, signature)
fillOrder(адреса Заявника, сума Заявника, токен Заявника, адреса Виконувача, сума Виконувача, токен Виконувача, дата спливання терміну дії, відмітка поточного часу, підпис)

Одиничний обмін токенами, викликаний Виконувачем. Контракт гарантує відповідність відправника повідомлення Виконувачу і те, що зазначений термін дії не сплинув. Щоб замовлення були виконані, однорангові вузли повинні попередньо затвердити переведення на їхні рахунки токенів у кількостях, які дозволять вилучити щонайменше зазначені в контракті суми. Щоб здійснити обмін сторін токенами, контракт розсилає виклик transferFrom із запитом про передачу відповідних токенів. По успішному завершенні цієї функції в блокчейн транслюється подія Filled про виконання замовлення.

```
Приклад: “I want to fill this order of 5 GNO for 10 BAT.”
«Я хочу виконати це замовлення про покупку 5 токенів GNO за 10 токенів BAT».
fillOrder([maker], 5, ‘GNO’, [taker], 10, ‘BAT’, [expiration], [signature])
fillOrder([Заявник], 5, ‘GNO’, [Виконувач], 10, ‘BAT’, [дата спливання терміну дії], [підпис])
```

cancelOrder(makerAddress, makerAmount, makerToken, takerAddress, takerAmount, takerToken, expiration, nonce, signature)
cancelOrder(адреса Заявника, адреса Виконувача, сума Виконувача, токен Виконувача, адреса Виконувача, сума Виконувача, токен Виконувача, дата спливання терміну дії, відмітка поточного часу, підпис)

Скасування замовлення, яке вже було передано Виконувачу, але ще не виконано, викликається Заявником відповідного замовлення. Замовлення позначається, як уже виконане згідно з контрактом, тому наступна спроба виконати замовлення буде невдалою. Після успішного завершенні цієї функції в блокчейн транслюється подія Canceled про скасування замовлення.

```
Приклад: “I want to cancel this order of 5 GNO for 10 BAT.”
«Я хочу скасувати це замовлення на покупку 5 токенів GNO за 10 токенів BAT».
cancelOrder([maker], 5, ‘GNO’, [taker], 10, ‘BAT’, [expiration], [signature])
cancelOrder([Заявник], 5, ‘GNO’, [Виконувач], 10, ‘BAT’, [дата спливання терміну дії],
[підпис])
```

Замовлення на торгівлю токенами Етер (Ether)

Смарт-контракт підтримує торгівлю токенами Етер. Якщо в замовленні зазначена нульова адреса Виконувача (0x0), смарт-контракт перевірить суму в Етерах, яку було надіслано в функціональному виклику, і перерахує її Заявнику від імені Виконувача.

Висновки

Протокол Swap слугує зростаючому попитові на децентралізований обмін активами в мережі Етереум. Незважаючи на те, що Реєстри замовлень на базі блокчейну є новітніми і безумовно відповідають етичним нормам нашої екосистеми, вони мають обмеження, які, на нашу думку, в остаточному рахунку ускладнюють їхню конкуренцію з наявними в даний час централізованими рішеннями. Протокол Своп забезпечує можливість застосування метода, який є як децентралізованим, так і таким, на який не впливають ці обмеження.

Запроваджуючи цей протокол, учасники отримують доступ до регулювання ліквідності масштабованим, приватним і рівноправним способом, не поступаючись доступом до вигідних цін. Цей протокол та його інтерфейси прикладного програмування є розширюваними, і ми заохочуємо нашу спільноту створювати програмні додатки разом з нами. Ми будемо раді отримати від вас відгуки та з нетерпінням очікуємо, спільно з вами, подальшого розвитку спільноти Етереум.

З усіма запитаннями, зауваженнями та відгуками звертайтеся до нас на адресу team@swap.tech.