Задача командного тура

В командной части заключительного этапа участники должны разработать децентрализованное приложение на базе платформы Ethereum, для взаимодействия автономных электромобилей и автоматических станций сервисного обслуживания, предоставляющих услуги по замене разряженных тяговых аккумуляторных батарей. Приложение включает в себя логику проверки батарей на контрафакт, а также экономическую систему мотивации участия и защиты от мошенничества.

Командная часть заключительного этапа имеет продолжительность 3,5 дня (всего 24 астрономических часа), которые включают работу по настройку среды разработки на базе узла сети Ethereum, разработку частей приложения на языке Python и Solidity, создание интеграционных тестов, а также приемка решения в системе автоматической проверки.

9.1. Легенда

Наступило ближайшее будущее автобаны и дороги в городах наводнены электромобилями! Свободный рынок позволяет каждому производителю автомобилей, существующему еще с 20 века, выпускать широкий модельный ряд устройств, сильно отличающихся от своих предшественников на органическом топливе. Огромное количество крупных и мелких компаний снабжают автозаводы всеми необходимыми комплектующими начиная от дворников и брызговиков, заканчивая электромоторами и тяговыми аккумуляторными батареями.

На тяговом аккумуляторе автомобиль может проехать относительно небольшое расстояние — в среднем, 400-500 километров. Затем, ему нужно перемещаться к зарядной станции или центру сервисного обслуживания.

На зарядной станции можно не спеша зарядить батарею за 45-50 минут. На автоматизированном центре сервисного обслуживания аккумуляторное устройство могут быстро заменить за 3-4 минуты. Естественно, центры обслуживания пользуются заслуженной популярностью.

Однако, при большом количестве производителей аккумуляторных батарей, огромном количестве электромобилей и сотнях различных компаний, держащий центры сервисного обслуживания возникает проблема доверия: действительно ли в обмене между транспортным средством и центром обслуживания участвуют только качественные батареи, ведь получив батарею с израсходованным ресурсом зарядов автомобиль рискует застрять где-то в пути, а станция сервисного обслуживания будет терпеть убытки, связанные с необходимостью обновления своих складских запасов.

Поэтому необходимо написать программное обеспечение для использования участ-

никами обмена, которое бы позволяло гарантированно проверять качество аккумуляторных батарей, автоматизировало бы процесс ведения сделки по оказанию услуг сервисным центром, а также обеспечивало вы финансовое сопровождение всех необходимых операций без привлечения человека: ведь не только станция может быть автоматической, но и электромобиль может оказаться беспилотным.

Чтобы исключить возможность злонамеренных действий со стороны каждой из сторон участвующей в системе экономических взаимоотношений: компании производителя ПО, производителя батарей, электромобиля и центра сервисного обслуживания, программное обеспечение должно быть разработано на базе платформы распределенного реестра. Это позволит решению обладать следующими преимуществами:

- Нет единой точки компрометации - код децентрализованного приложения будет выполняться на узлах сети блокчейн - валидаторах сети, а изменения в данных, принятые в ходе исполнения кода, будут многократно перепроверяться всеми другими узлами.
- Нет единой точки хранения данных - код приложения и данные хранятся одновременно на всех узлах сети блокчейн, контролируемые криптографической связностью данных. Это, вместе с предыдущим пунктом, позволяет гарантировать, что никакой из участников системы не сможет поменять свои договоренности "задним числом" или, подменив код приложения, заставить взаимодействовать участников по новым, неизвестным для них правилам.
- Нет единой точки входа, что позволяет участникам избегать атак, построенных по принципу man-in-middle.
- опирающаяся на безопасность и простоту сети блокчейн экономическая модель будет поддерживать мотивацию участников системы к честному поведению и продолжению взаимодействия.

Участники взаимодействия

При разработке программного обеспечения должно учитываться взаимодействие следующих сущностей:

- **производитель программного обеспечения** - выполняет развертывание системы: распространение скриптов между участниками системы, регистрацию смарт-контрактов, а также настройку систему. Он является выгодоприобретателем при подключении новых участников системы. Производитель ПО отвечает за введение в оборот расчетных токенов, обеспечение механизмов распределения расчетных токенов между участниками (не рассматривается в данной задаче).

  Все действия, происходящие от имени производителя ПО, выполняются через скрипт `setup.py`.
- **производитель тяговых аккумуляторных батарей** - оплачивает свою регистрацию в системе. Выпускает новые батареи и регистрирует их в системе. Первые 1000 батарей регистрируются за счет первоначальной оплаты, за регистрацию каждой следующей батареи производитель должен вносить дополнительную сумму. Сумма оплаченная за регистрацию производителя батарей и за регистрацию самих батарей перечисляется производителю программного обеспечения. После регистрации аккумуляторные батареи могут

быть распределены производителем батарей между сервисными центрами и электромобилями.

Все действия, происходящие от имени производителя батарей, выполняются через скрипт `vendor.py`.

- **тяговые аккумуляторные батарей** - объект реального мира, состоящий из элементов, аккумулирующих электрический заряд, электронной управляющей системы с постоянной памятью. В рамках данной задачи считается, что нельзя изменить физико-химические характеристики батареи без выведения из строя электронной управляющей системы. В самом простом случае управляющая система включает в себя подсистему подсчитывающую количество циклов заряда аккумуляторной батареи, аппаратного ключа защиты, которые хранит приватный ключ батареи, он генерируется при создании батареи и уникальный для каждого устройства, и содержит код для создания цифровой подписи по запросу от управляющей системы. Также батарея имеет специальный цифровой порт, через который можно получить так называемый статус батареи: количество циклов заряда, временную метку и соответствующие им цифровую подпись. За счет того, что временная метка изменяется от одного считывания к другому, цифровая подпись будет все время отличаться даже при одинаковом значении циклов заряда.

  В данном решении электронная управляющая система батареи будет эмулироваться скриптом на языке `Python`.

- **сервисный центр по замене тяговых аккумуляторных батарей** - регистрируется в системе (регистрация бесплатная), после чего получает право отвечать на запросы электромобилей для замены батарей. Подключается к батарее, установленной в электромобиле для получения информации о статусе батареи, находит подходящую заряженную тяговую батарею, оформляет контракт на сделку по замене батарей. После замены батарей, сервисный центр получает от электромобиля компенсацию за свою работу в виде определенного количества расчетных токенов. Изначально получает батареи от производителя батарей.

  Все действия, происходящие от имени сервисного центра, выполняются через скрипт `scenter.py`.

- **электромобиль** - регистрируется в системе (регистрация бесплатная), после чего получает право быть участником в сделках по замене батарей. Получает информацию о контракте по замене батарей от сервисного центра. После ознакомления с условиями контракта соглашается с ними, переводя расчетные токены на счет контракта. После замены батарей, проверяет статус вновь установленной батареи и завершает контракт, что инициирует перевод расчетных токенов на счет сервисного центра.

  Все действия, происходящие от имени электромобиля, выполняются через скрипт `car.py`.

9.2. Набор заданий

Решение командной задачи было разбито на подзадачи, сгруппированные в 3 набора.

Каждая подзадача (*user story*) формулировала необходимый функционал, ко-

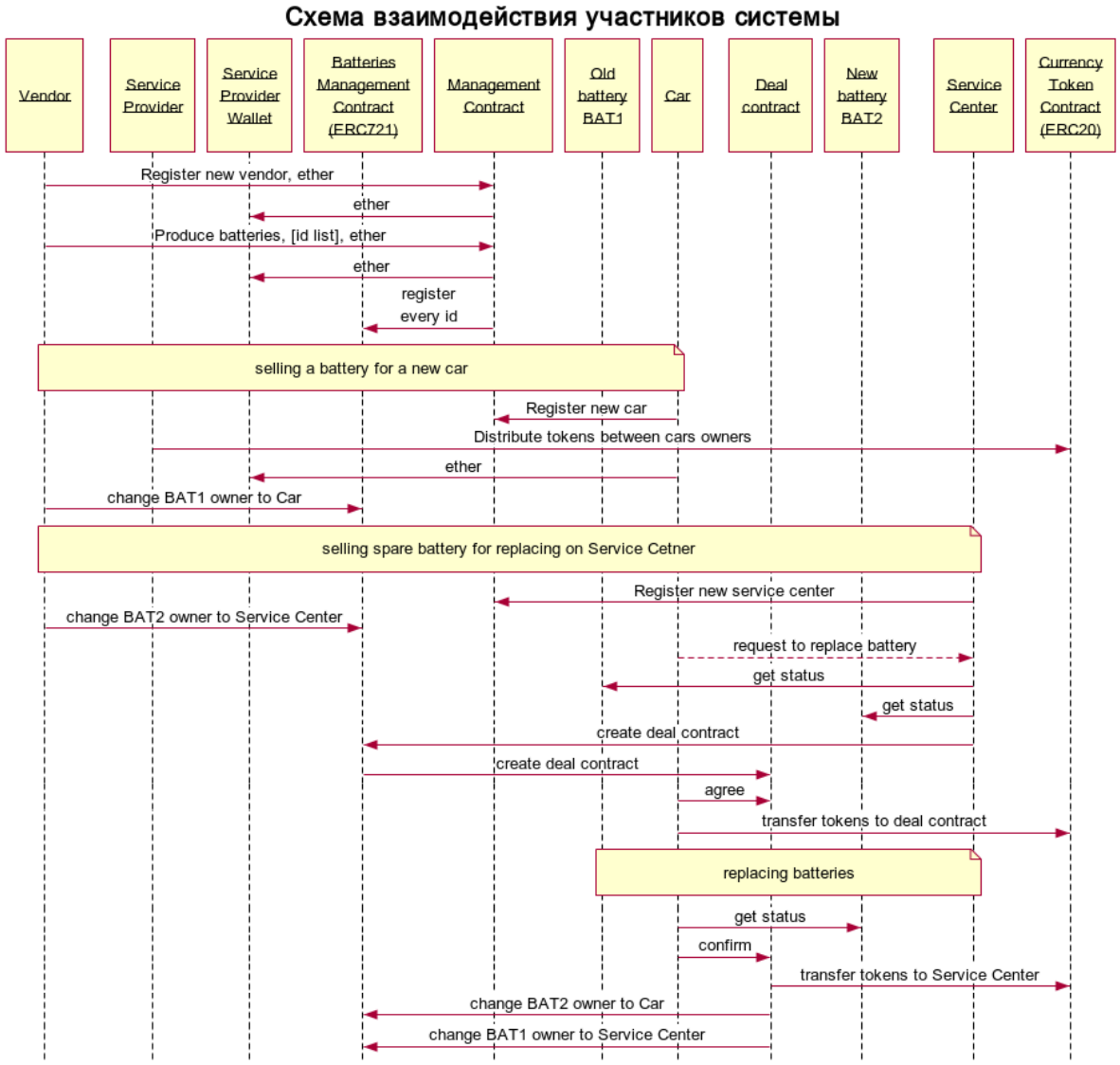

Рис. 9.1: Диаграмма взаимодействия участников системы

торый должен был быть реализован командой, а также набор приемочных тестов (*acceptance criteria*), позволявших проверить в полном ли объеме решена данная подзадача.

Каждый набор подзадач предлагалось решать в отдельном этапе (*итерации*). Первые два этапа итеративно подводили участников к решению полной финальной задачи, осуществляемому во время последнего третьего этапа. На каждом этапе решение подзадач проверялось с помощью системы автоматической системы оценивания, которая запускала решение участников с теми или иными параметрами в соответствии с приемочными тестами.

Решение командной задачи было разбито на подзадачи, сгруппированные в 3 набора.

Каждая подзадача (*user story*) формулировала необходимый функционал, который должен был быть реализован командой, а также набор приемочных тестов (*acceptance criteria*), позволявших проверить в полном ли объеме решена данная подзадача.

Каждый набор подзадач предлагалось решать в отдельном этапе (*итерации*).

Первые два этапа итеративно подводили участников к решению полной финальной задачи, осуществляемому во время последнего третьего этапа. На каждом этапе решение подзадач проверялось с помощью системы автоматической системы оценивания, которая запускала решение участников с теми или иными параметрами в соответствии с приемочными тестами.

Первая итерация

Участникам необходимо разработать базовый функционал для скриптов `setup.py`, `vendor.py`, `scenter.py`, `car.py`, реализовать Ethereum-контракты `Management Contract` и `Battery Management Contract`, тем самым решая следующие подзадачи:

user story (*)	**acceptance criteria**
US-001 Регистрация аккаунта компании производителя ПО	все
US-002 Регистрация контрактов	все
US-005 Создание аккаунта производителя батарей	все
US-007 Регистрация нового производителя	AC-007-01
US-010 Поштучная регистрация производимых батарей	все
US-020 Создание аккаунта сервисного центра	все
US-021 Регистрация аккаунта сервисного центра	все
US-022 Создание аккаунта электромобиля	все
US-023 Получение аккаунта электромобиля	все
US-024 Регистрация аккаунта электромобиля	все

(*) формулировки задач приведены в разделе "Подробное описание подзадач".

Решение подзадач было направлено на проверку следующих компетенций:

- настройка приватной сети Ethereum для тестирования решения в ходе процесса разработки;
- настройка окружения разработки программного обеспечения для использования системы ведения версий `Git`, Python-библиотек для взаимодействия с узлами блокчейн платформы Ethereum и компиляции и регистрации Ethereum контрактов;
- написание и отладка программ на языке Python, работающих с параметрами командной строки;
- обеспечение отправки JSON-RPC запросов (включая платежные транзакции и транзакции на вызов методов Ethereum контрактов) из программ на языке Python к провайдерам `Web3`;
- реализация алгоритмов работы с приватными ключами для подписи Ethereum транзакций;
- написание и отладка Ethereum контрактов на языке Solidity.

Вторая итерация

Участникам необходимо завершить работу над минимально достаточным функционалом децентрализованного приложения, который бы позволял производить проверку тяговых аккумуляторных батарей на контрафакт, регистрировать новую сделку, подтверждать и завершать ее, обеспечивал бы экономическое взаимодействие

участников для стандартного сценария поведения. Данный функционал покрывался следующими подзадачами:

user story (*), все acceptance criteria
US-003 Установка сборов за регистрацию одной батареи
US-006 Получение информации о необходимом депозите за регистрацию производителя батарей
US-007 Регистрация нового производителя
US-008 Защита от повторной регистрации нового производителя
US-009 Получение информации о сборе за регистрацию одной батареи
US-011 Регистрация батарей с закрепленным сбором за регистрацию батарей
US-012 Получение размера остатка по депозиту
US-013 Пакетная регистрация производимых батарей
US-015 Поштучная продажа новых батарей
US-017 Генерация прошивок аппаратных ключей батарей
US-018 Изменение количества заряда батареи
US-019 Получение информации о батарее из прошивки
US-025 Проверка подлинности батареи сервисным центром
US-026 Проверка подлинности батареи электромобилем
US-027 Получение адреса контракта для замены батареи
US-029 Получение информации, что контракт замены в ожидании подтверждения
US-030 Получение условий контракта
US-031 Подтверждение согласия с условиями контракта
US-032 Подтверждение готовности к физической замене батарей
US-033 Подтверждение контракта
US-037 метод setBatteryManagementContract()
US-038 метод registerVendor()
US-039 метод registerBatteries()
US-040 метод setFee()
US-041 метод registerServiceCenter()
US-042 метод registerCar()
US-043 метод createBattery()
US-044 метод transfer() (с указанием одного идентификатора батареи)
US-045 метод delegatedApprove()
US-046 метод initiateDeal()
US-048 метод agreeToDeal()

(*) формулировки задач приведены в разделе "Подробное описание подзадач".

Решение подзадач было направлено на проверку следующих компетенций:

- разработка архитектуры простого децентрализованного приложения;
- реализация хранилища уникальных цифровых объектов и алгоритмов изменения их свойств на базе стандарта ERC721 (non-fungible tokens);
- реализация хранилища расчетных токенов и алгоритмов передачи прав владения на базе стандарта ERC20;
- реализация алгоритмов:
  - упаковки цифровой информации;
  - проверки истинности информации с использованием цифровой подписи;
- выстраивание процесса интеграционного тестирования компонент децентрализованного приложения.

Третья итерация

Работа участников должна быть направлена на наращивание функционала децентрализованного приложения и оптимизацию его работы, а также разработку инструмента сбора статистики. После заверешения итерации у участников должно быть 5 Python-скриптов и 4 Ethereum контракта, закрывающие последнии набор подзадач:

user story (*), все acceptance criteria
US-004 Регистрация контрактов с постоянными адресами
US-014 Безопасная пакетная регистрация производимых батарей
US-016 Пакетная продажа новых батарей
US-028 Отправка запроса на отмену сделки
US-034 Установка времени запрета разблокировки токенов
US-035 Запрос на разблокирование токенов
US-036 Отсутствие подтверждения замены батарей для завершения сделки
US-047 метод setTimeoutThreshold()
US-050 метод releaseTokensByCar()
US-051 метод releaseTokensByServiceCenter()
US-052 метод cancelDeal()
US-053 Передача прав на батарею новому владельцу
US-054 Отложенная передача прав владения
US-055 Регистрация прав владения
US-056 Оптимизация по использованию вычислительных ресурсов
US-057 Оптимизация комиссии за валидацию транзакций
US-058 Сбор статистики

(*) формулировки задач приведены в разделе "Подробное описание подзадач".

Решение подзадач было направлено на проверку следующих компетенций:

- оптимизация работы Ethereum контрактов;
- организация взаимодействия с ненадежными источниками данных;
- реализация обхода цепочки блоков с последующим извлечением информации из транзакций для сбора статистики о работе децентрализованного приложения.

9.3. Условия проведения

- В первые день соревнований все члены команд получают ноутбук со следующим набором установленного программного обеспечения:
  - клиент сети Ethereum `geth`;
  - компилятор языка Solidity `solc`;
  - набор Python инструментария Anaconda с установленными библиотеками `web3.py` и `py-solc`;
  - среды разработки PyCharm и WingIDE;
  - клиент системы ведения версий `git`.
  - Интернет-браузер Chrome.

  Каждый ноутбук имеет возможность выхода в сеть Интернет. Команды могут использовать собственные ноутбуки.

- Команды могут устанавливать дополнительное программное обеспечение, но после согласование с членами жюри.
- Участники во время командного этапа финального тура могут использовать интернет и заранее подготовленные библиотеки для решения задачи.
- Участники должны использовать язык программирования Python для написания программ, использующих командную строку. Для написания Ethereum контрактов участники могут использовать любой язык программирования.
- Участники не могут использовать помощь тренера, сопровождающего лица или привлекать третьих лиц для решения задачи.
- Финальная задача формулируется участникам в первый день финального тура, но участники выполняют решение задачи поэтапно. Критерии прохождения каждого этапа формулируются для каждого дня финального тура. За подзадачи, решенные в конкретном этапе начисляются, баллы. Баллы за подзадачи можно получить только в день, закрепленный за конкретным этапом.
- В начале первого дня состязаний участники каждой команды получают доступ к репозиторию на серверах GitLab.com. Каждая команда имеет свой собственный репозиторий. Члены других команд не имеют доступ к чужим репозиториям.
- В начале первого дня состязний участникам выдется описание интерфейсов контрактов `Management Contract`, `Battery Management Contract`, `Deal contract`, `Currecncy token contract`, также участникам выдается исходный код `Service provider wallet` контракта, поскольку не является предметом оценки.
- В течение дня не ведется учет количества изменений, которые команды регистрируют в Git-репозитории.
- В конце каждого дня финального этапа жюри проверяет решение участников на соответствие приемочным тестам для каждой подзадачи, входящей в набор для соответствующей итерации.
- Баллы за все подзадачи, для которых прошло приемочное тестирование, определяют баллы, набранные командой в данный день соревнований.
- После выставления баллов, командам предоставляется доступ к системе автоматического тестирования, ответственной за проведение приемочных тестов в конкретный день состязаний, так что члены команды могут ознакомиться с логикой проверки и подать апелляцию, если не согласны с корректностью проведения тестов.
- После рассмотрения сути апелляции, жюри вправе провести тестирование вручную и назначить команде баллы за соответствующие подзадачи.
- В начале следующего дня состязаний жюри выдает всем командам свое решение подзадач предыдущего дня, которое команды могут использовать для того, чтобы решать следующий набор подзадач.

9.4. Процедура проведения приемочного тестирования и критерии оценки

Для каждого дня соревнований (для каждой итерации) справедлива следующая процедура приемочного тестирования:

- В конце каждого дня финального этапа команды должны сформировать запрос на слияние (*Merge Request*) из своей ветки исходного кода в основную ветку (*master*) в Git-репозитории.
- Команда ответственна за то, чтобы в запросе на слияние не должно быть конфликтов. Запрос на слияние с конфликтами может не рассматриваться жюри для выполнения приемочного тестирования.
- После того, как все команды отправили запросы на слияние, жюри одобряет все запросы и приступает к приемочному тестированию, для тех подзадач, которые входят в соответствующую итерацию. Для этого исходный код приложения команды копируется на сервер жюри, и прогоняются автоматические тесты на соответствие решения участников требованиям к приемочным тестам (*acceptance criteria*).
- Если все приемочные тесты для данной подзадачи пройдены успешно, команда получает баллы за данную подзадачу. Если хотя бы один тест не проходит, то баллы за данное подзадачу не начисляются.

Приемочные тесты для каждой подзадачи описаны в разделе "Подробное описание подзадач".

Дальше перечислены баллы, которые получала команда за решение подзадач в каждой итерации.

Максимальное количество баллов, которое могла набрать команда за решение всех подзадач — 400.

Первая итерация

user story	**баллы**
US-001 Регистрация аккаунта компании производителя ПО	2
US-002 Регистрация контрактов	12
US-005 Создание аккаунта производителя батарей	2
US-007 Регистрация нового производителя	8
US-010 Поштучная регистрация производимых батарей	6
US-020 Создание аккаунта сервисного центра	2
US-021 Регистрация аккаунта сервисного центра	6
US-022 Создание аккаунта электромобиля	2
US-023 Получение аккаунта электромобиля	2
US-024 Регистрация аккаунта электромобиля	6

Максимальное количество баллов за итерацию — 48.

Вторая итерация

user story	баллы
US-003 Установка сборов за регистрацию одной батареи	6
US-006 Получение информации о необходимом депозите за регистрацию производителя батарей	4
US-007 Регистрация нового производителя	4
US-008 Защита от повторной регистрации нового производителя	4
US-009 Получение информации о сборе за регистрацию одной батареи	4
US-011 Регистрация батарей с закрепленным сбором за регистрацию батарей	4
US-012 Получение размера остатка по депозиту	4
US-013 Пакетная регистрация производимых батарей	6
US-015 Поштучная продажа новых батарей	6
US-017 Генерация прошивок аппаратных ключей батарей	4
US-018 Изменение количества заряда батареи	3
US-019 Получение информации о батарее из прошивки	8
US-025 Проверка подлинности батареи сервисным центром	9
US-026 Проверка подлинности батареи электромобилем	9
US-027 Получение адреса контракта для замены батареи	16
US-029 Получение информации, что контракт замены в ожидании подтверждения	4
US-030 Получение условий контракта	8
US-031 Подтверждение согласия с условиями контракта	16
US-032 Подтверждение готовности к физической замене батарей	4
US-033 Подтверждение контракта	16
US-037 метод setBatteryManagementContract()	2
US-038 метод registerVendor()	4
US-039 метод registerBatteries()	3
US-040 метод setFee()	2
US-041 метод registerServiceCenter()	2
US-042 метод registerCar()	2
US-043 метод createBattery()	3
US-044 метод transfer() с указанием одного идентификатора батареи	2
US-045 метод delegatedApprove()	4
US-046 метод initiateDeal()	6
US-048 метод agreeToDeal()	6

Максимальное количество баллов за итерацию — 175.

Третья итерация

user story	баллы
US-004 Регистрация контрактов с постоянными адресами	20
US-014 Безопасная пакетная регистрация производимых батарей	6
US-016 Пакетная продажа новых батарей	6
US-028 Отправка запроса на отмену сделки	8
US-034 Установка времени запрета разблокировки токенов	6
US-035 Запрос на разблокирование токенов	10
US-036 Отсутствие подтверждения замены батарей для завершения сделки	10
US-047 метод setTimeoutThreshold()	2
US-050 метод releaseTokensByCar()	4
US-051 метод releaseTokensByServiceCenter()	4
US-052 метод cancelDeal()	4
US-053 Передача прав на батарею новому владельцу	8
US-054 Отложенная передача прав владения	12
US-055 Регистрация прав владения	8
US-056 Оптимизация по использованию вычислительных ресурсов	33
US-057 Оптимизация комиссии за валидацию транзакций	6
US-058 Сбор статистики	30

Максимальное количество баллов за итерацию — 177.

9.5. Подробное описание подзадач

Единой точкой взаимодействия с децентрализованным приложением для всех компонент является `Management Contract`. Практически для любого сценария, за исключением сценариев создания аккаунтов и регистрации (deployment) контрактов в сети блокчейн, компоненты должны для получения адреса `Management Contract` обращаться к файлу в формате `JSON` `database.json`, находящемуся в текущей рабочей директории:

```
{"mgmtContract": "0x00360d2b7D240Ec0643B6D819ba81A09e40E5bCd"}
```

Если это явно не прописано в описании функционала соответствующей компоненты, она не должна оставлять после себя никаких файлов-данных. Аналогично и с получением данных - чтение данных из файла должно происходить только в нескольких случаях

- для получения адреса `Management Contract`,
- для получения `Application Binary Interface` контракта или его байткода,
- или если это явно указано в описании компоненты,

в остальных случаях данные должны быть получены из блокчейн.

Все компоненты, отправляющие транзакции в блокчейн, должны уметь обрабатывать ситуации, когда:

- недоступно подключение к узлу Ethereum (сообщение об ошибке `''No connection to node''`);
- на счету аккаунта, от чьего имени выполняется транзакция, недостаточно средств, чтобы покрыть все затраты по использованному газу (сообщение об ошибке `''No enough funds to send transaction''`);
- транзакция может не включаться в блок продолжительное время в зависимости от нагрузки на сеть (сообщение об ошибке `''Transaction is not validated too long''`).

При возникновении данных событий в терминал должно выдаваться сообщение понятное пользователю, а не stack trace.

Подключение к узлу Ethereum, там где это необходимо, должно происходить с помощью автоматического определения Web3 провайдера. Для этого гарантируется, что перед запуском той или иной компоненты децентрализованного приложения будет установлена переменная окружения `WEB3_PROVIDER_URI` (`http://web3py.readthedocs.io/en/latest/quickstart.html#provider-uri`).

Настройка децентрализованного приложения

Развертывание и настройка децентрализованного приложения выполняется компанией производителем программного обеспечения (тем, кто разрабатывает данный проект). Предполагается, что в потенциальные пользователи проекта (производители батарей, владельцы станции сервисного обслуживания и производители электромобилей) могут ознакомиться с опубликованными в открытом доступе исходными кодами контрактов децентрализованного приложения, принять решение на основе этого, доверять ли этому разработчику и начать пользоваться сервисом.

Фиксация контрактов в блокчейн гарантирует, что приозводитель не сможет изменить логику работы приложения позднее, т.е. условия участия в системе, с которыми ознакомились пользователи, не поменяется.

Функционал децентрализованного приложения должен быть доступен сразу, минуя фазы настройки, когда децентрализованное приложение развернутое в блокчейн, но еще находится в нерабочем состоянии. Поэтому первичная настройка базовых параметров по максимуму должна выполняться вовремя инициализации контрактов во время их регистрации (deploy).

Имя скрипта
`setup.py <command> [options]`

В тех случаях, когда скрипту необходимо отправлять транзакции для изменения состояния блокчейн, аккаунт и пароль, которые будут использоваться для доступа к приватному ключу для подписи транзакций, должны браться из файла в формате JSON `account.json`:

```
{
    "account": "0x5338d77b5905cdeea7c55a1f3a88d03559c36d73",
    "password": "s0me_p455w0rd"
}
```

US-[3]USCounter Регистрация аккаунта компании производителя ПО

Использование скрипта
`setup.py --new <password>`
Подключается к локальному узлу Ethereum и создает новый аккаунт с паролем, заданным в строке ввода. Также создает файл базы данных `account.json`, в котором сохраняется аккаунт и пароль для дальнейшего использования данным производителем. В терминал выводится адрес аккаунта.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# setup.py --new 's0me_p455w0rd'` `0x009e2Ec03B40360d64A00E5bCd6D819ba817D24b`
Комментарий: На локальном узле создан пользователь. В текущей рабочей директории создается файл `account.json`, в котором сохраняется аккаунт и пароль в формате `JSON` для дальнейшего использования данным производителем.

EPIC-[2]EpicCounter Регистрация и конфигурация контрактов

US-[3]USCounter Регистрация контрактов

Использование скрипта
`setup.py --setup <service fee>`
Подключается к узлу Ethereum и регистрирует контракты в сети блокчейн: • `Service Provider Wallet Contract` • `Management Contract` • `Battery Management Contract` • `Currency Token Contract` Параметр `<service fee>` используется для того, чтобы при создании контракта `Management Contract` указать величину сборов за регистрацию одной батареи. В терминал выводится адрес `Management Contract`, `Service Provider Wallet Contract` и `Currency Token Contract`.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# setup.py --setup 0.01`
`Management contract: 0x00360d2b7D240Ec0643B6D819ba81A09e40E5bCd`
`Wallet contract: 0xcb3ad4765bc9056a6bb09d24bb5ef3306532db9a`
`Currency contract: 0x2380943c84daa97b5d3b4be6d77692b2eb486491`

Комментарий: В блокчейн сеть отправляются транзакции для регистрации и первичной настройке контрактов. Транзакции успешно включены в один или несколько блоков. `Management Contract` можно проинспектировать на предмет установки: • суммы сборов за регистрацию одной батареи; • адреса `Service Provider Wallet Contract`; • адреса `Battery Management Contract`. `Battery Management Contract` можно проинспектировать на предмет установки: • адреса `Management Contract`; • адреса `Currency Token Contract`. `Currency Token Contract` можно проинспектировать на наличия на балансе компании производителя ПО 10^{20} токенов, необходимых для экономического взаимодействия участников системы. В текущей рабочей директории создается файл в формате `JSON` `database.json`, в котором указан адрес `Management Contract`.

US-[3]USCounter Установка сборов за регистрацию одной батареи

Несмотря на то, что величина сборов за регистрацию одной батареи устанавливается во время регистрации контрактов, должна быть возможность сменить данную величину, например, из-за требований рынка: сбор оказался в самом начале слишком высокий или, наоборот, инфляция требует поднять величину сбора, чтобы сделать проект экономически целесообразным.

Использование скрипта
`setup.py --setfee <service fee>`
Подключается к узлу Ethereum и отправляет запрос к `Management Contract` на изменение сумму сборов за регистрацию одной батареи.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# setup.py --setfee 0.01` `Updated successfully`
Комментарий: В блокчейн сеть отправляется транзакция. Транзакция успешно включена в блок. `Management Contract` можно проинспектировать на предмет изменений.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# setup.py --setfee 0.01` `No permissions to change the service fee`
Комментарий: Если попытка изменения суммы сборов происходит не от аккаунта компании производителя ПО, то выдается ошибка. Транзакция на регистрацию производителя не отправляется в блокчейн.

US-[3]USCounter Регистрация контрактов с постоянными адресами

Одним из конкуретных преимуществ приложения может быть то, что адреса контрактов, по развернуты контракты - заранее известные. Это позволит повысить доверие к системе (адреса заранее указаны в контрактах, а не передаются на этапе инициализации) и упростить процедуру подключения новых пользователей (адрес контракта не нужно настраивать через конфигурационные файлы).

В реальной жизни данный функционал особенно востребован в случае присутствия какого-то контракта (например, контракта токена) в разных, так называемых, side chains, когда функционал контракта распределяется между отдельными блокчейн сетями: например при наличии контракта non-fungible token его сложная бизнес-логика изменения его свойств или генерации других контрактов, реализована в одном блокчейн, где стоимость вычислений мала, а более легковесная логика перехода прав владения, связанная со сделками купли-продажи с использованием ликвидной криптовалюты, - в другом. Тогда выглядит логичным вариант, когда в разных блокчейн сетях данный контракт имеет один и тот же адрес.

Поскольку, подразумевается, что исходный код скрипта `setup.py` доступен каждому, то в нем не должно храниться приватного ключа для подписи транзакций регистрации контрактов. Также приватный ключ не должен извлекаться из других файлов, поскольку их также пришлось бы выкладывать в открытый доступ для максимального доверия. В рамках данной US предлагается найти способ, который требует наличия только одного заранее аккаунта, созданного через `setup.py -new`, информация о котором передается через `account.json`.

Использование скрипта
`setup.py --setup <service fee>`
Подключается к узлу Ethereum и регистрирует контракты в сети блокчейн: • `Service Provider Wallet Contract` • `Management Contract` • `Battery Management Contract` • `Currency Token Contract` Вне зависимости от того какой аккаунт принадлежит компании производителю ПО и вне зависимости от того, в блокчейн с каким идентификатором сети происходит регистрация контрактов, каждый из контрактов будет иметь один и тот же адрес.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# echo $WEB3_PROVIDER_URI`
`http://127.0.0.1:8545`
`# setup.py --setup 0.01`
`Management contract: 0x519D75381DfD9e8e1128698C6B6f9f76FC242404`
`Wallet contract: 0xC1Af0c06cAe672B3514Be9683663c88d8Db920bf`
`Currency contract: 0xbFD04af48C978cC0d9Bc5E06d9593Cb4fb7f6f98`
`# setup.py --new 'secr3tw0rd'`
`# env WEB3_PROVIDER_URI=http://127.0.0.1:7545 setup.py --setup 0.01`
`Management contract: 0x519D75381DfD9e8e1128698C6B6f9f76FC242404`
`Wallet contract: 0xC1Af0c06cAe672B3514Be9683663c88d8Db920bf`
`Currency contract: 0xbFD04af48C978cC0d9Bc5E06d9593Cb4fb7f6f98`

Комментарий: Транзакции после вызова первой команды `setup.py` отправляются на узел, принадлежащий одной блокчейн сети. А транзакции после вызова второй команды отправляются на узел, принадлежащий другой блокчейн сети. Адреса всех четырех контрактов, получившихся после регистрации в одной и в другой сети - одинаковые: адрес `Management Contract` в одной сети, равен адресу `Management Contract` в другой сети, адрес `Service Provider Wallet Contract` в одной сети, равен адресу `Service Provider Wallet Contract` в другой сети и т.д.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# setup.py --setup 0.01` `At least one address already occupied.`
Комментарий: Если при регистрации контрактов в сети обнаруживается, что один из адресов под которым контракты должны зарегистрироваться уже занят под адрес пользователя или адрес контракта, то выдается ошибка. Транзакции на регистрацию контрактов не отправляется в блокчейн. *Примечание:* данный критерий будет проверяться только при успешной приемке предыдущего критерия.

Производитель батарей

Имя скрипта
`vendor.py <command> [options]`

В тех случаях, когда скрипту необходимо отправлять транзакции для изменения состояния блокчейн, аккаунт и пароль, которые будут использоваться для доступа к приватному ключу для подписи транзакций, должны браться из файла в формате JSON `account.json`:

```
{
    "account": "0x5338d77b5905cdeea7c55a1f3a88d03559c36d73",
    "password": "s0me_p455w0rd"
}
```

EPIC-[2]EpicCounter Регистрация нового производителя батарей

US-[3]USCounter Создание аккаунта производителя батарей

Использование скрипта
`vendor.py --new <password>`
Подключается к локальному узлу Ethereum и создает новый аккаунт с паролем, заданным в строке ввода. Также создает файл базы данных `account.json`, в котором сохраняется аккаунт и пароль для дальнейшего использования данным производителем. В терминал выводится адрес аккаунта.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# vendor.py --new 's0me_p455w0rd'` `0x00360d2b7D240Ec0643B6D819ba81A09e40E5bCd`
Комментарий: На локальном узле создан пользователь. В текущей рабочей директории создается файл `account.json`, в котором сохраняется аккаунт и пароль в формате `JSON` для дальнейшего использования данным производителем.

US-[3]USCounter Получение информации о необходимом депозите за регистрацию производителя батарей

Использование скрипта
`vendor.py --regfee`
Подключается к узлу Ethereum и выполняет **локальный** вызов функции получения размера регистрационного взноса у `Management Contract`. В терминал выводится полученная информация (в *ether*) в виде десятичной дроби с точностью до 6 знаков после запятой, незначащие завершающие нули не выводятся.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# vendor.py --regfee` `Vendor registration fee: 0.5`
Комментарий: Информация успешно получена и выведена в терминал. Нет транзакций в блокчейн для `Management Contract`.

US-[3]USCounter Регистрация нового производителя

Необходим депозит для предотвращения нежелательных регистраций.

Рис. 9.2: Диаграмма взаимодействия компонент

Использование скрипта
`vendor.py --reg <vendor name> <fee>`
Подключается к узлу Ethereum и отправляет запрос к `Management Contract` на регистрацию нового производителя батарей, название которого передается в качестве первого параметра к команде, а отправитель запроса ассоциируется как представитель данного производителя. Со счета отправителя запроса списывается сумма указанная в качестве второго параметра в качестве депозита за предоставляемый сервис. Сумма указывается в *ether*. В терминал выводится статус выполнения команды и идентификатор производителя батареи, который является шестнадцатеричной записью первых четырех байт от хэш-суммы ($keccak256$) адреса, наименования производителя и номера блока, в который включена транзакция на регистрацию производителя, объединенных вместе. *Примечание:* Поскольку частью процесса регистрации производителя является внесение депозита, то на счету аккаунта, от имени которого происходит регистрация должна быть достаточная для депозита сумма.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# vendor.py --reg 'Tesla Gigafactory' 0.5`
`Success.`
`Vendor ID: ______`

Комментарий: В блокчейн сеть отправляется транзакция на регистрацию производителя, транзакция успешно включена в блок, `Management Contract` можно проинспектировать на предмет добавленных данных (имя производителя ассоциировано с адресом отправителя, за данным производителем закреплен депозит). Баланс `Service Provider Wallet Contract` увеличивается на сумму депозита.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# vendor.py --reg 'Nissan Energy' 0.1` `Failed. Payment is low.`
Комментарий: Если при попытке регистрации производителя батарей указывается сумма недостаточная для депозита, то выдается ошибка. Транзакция на регистрацию производителя не отправляется в блокчейн.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# vendor.py --reg 'Nissan Energy' 0.5` `Failed. No enough funds to deposit.`
Комментарий: Если при попытке регистрации производителя батарей на счету аккаунта, который регистрирует производителя, недостаточно средств, то выдается ошибка. Транзакция на регистрацию производителя не отправляется в блокчейн.

US-[3]USCounter Защита от повторной регистрации нового производителя

Должна обеспечиваться только одна ассоциативная связь между аккаунтом производителя и его названием. В перспективе могут быть использованы алгоритмы машинного обучения для обеспечения защиты уникальности названия.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# vendor.py --reg 'Tesla Gigafactory' 0.5` `Failed. The vendor name is not unique.`
Комментарий: При попытке регистрации производителя батарей с названием, которое уже зарегистрировано, выдается ошибка. Транзакция на регистрацию производителя не отправляется в блокчейн.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# vendor.py --reg 'Nissan Energy' 0.5` `Failed. The vendor address already used.`
Комментарий: При попытке регистрации производителя батарей из под аккаунта, с которым уже регистрировался производитель, выдается ошибка. Транзакция на регистрацию производителя не отправляется в блокчейн.

EPIC-[2]EpicCounter Регистрация новых батарей

При регистрации новых батарей владельцем каждой батареи в `Battery Management Contract` назначается производитель данной батареи.

US-[3]USCounter Получение информации о сборе за регистрацию одной батареи

Использование скрипта
`vendor.py --batfee`
Подключается к узлу Ethereum и выполняет **локальный** вызов функции получения размера взноса за одну батарею у `Management Contract`. Размер взноса должен быть возвращен тот, который был установлен на момент регистрации производителя в системе. В терминал выводится полученная информация (в *ether*) в виде десятичной дроби с точностью до 6 знаков после запятой, незначащие завершающие нули не выводятся.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# vendor.py --batfee` `Production fee per one battery: 0.01`
Комментарий: Информация успешно получена и выведена в терминал. Нет транзакций в блокчейн для `Management Contract`.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# setyp.py --setfee 0.005`
`Updated successfully`
`# vendor.py --batfee`
`Production fee per one battery: 0.01`

Комментарий: Информация успешно получена и выведена в терминал. Нет транзакций в блокчейн для `Management Contract`.

US-[3]USCounter Поштучная регистрация производимых батарей

При положительном балансе депозита производителя батарей в `Management Contract` можно регистрировать батарею без перечисления средств. Иначе, при регистрации батареи необходимо пополнять депозит до суммы, достаточной для регистрации.

Использование скрипта
`# vendor.py --bat <number> [<deposit>]`
Подключается к узлу Ethereum и отправляет запрос к `Management Contract` на регистрацию новой батареи. Для определения производителя данной батареи в `Management Contract` используется аккаунт отправителя транзакции. С депозита производителя списывается сумма равная сбору за регистрацию одной, зафиксированному на момент регистрации производителя батарей и умноженному на количество батарей, которые предлагается зарегистрировать. В командной строке может быть передана сумма для пополнения депозита (сумма указывается в *ether*). В терминал выводится идентификатор батареи, который формируется по тому же принципу, что и адрес Ethereum. Приватный ключ, которому соответствует данный идентификатор, генерируется (псевдо)случайным образом в диапазоне от 0 до $2^{256} - 1$. *Примечание:* Поскольку частью процесса регистрации батареи является пополнение депозита, то на счету аккаунта, от имени которого происходит регистрация должна быть достаточная для пополнения сумма.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# vendor.py --bat 1` `Created battery with ID: d65bc90bb030472db9a6e9d653cb3a24bb556a3f`
Комментарий: В блокчейн сеть отправляется транзакция на регистрацию батареи, транзакция успешно включена в блок, `Battery Management Contract` можно проинспектировать на предмет добавленных данных. Депозит производителя батарей в `Management Contract` уменьшается на сумму, необходимую для регистрации одной батареи.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# vendor.py --bat 1` `Failed. No enough funds to register object.`
Комментарий: Если при попытке регистрации батареи депозит на `Management Contract` не содержит достаточно средств, то выдается ошибка. Транзакция на регистрацию батареи не отправляется в блокчейн.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# vendor.py --bat 1 0.0005` `Created battery with ID: 915d3b4be6d776923824daa97bb2eb4860943c84`
Комментарий: В блокчейн сеть отправляется транзакция на регистрацию батареи, транзакция успешно включена в блок, `Battery Management Contract` можно проинспектировать на предмет добавленных данных. Баланс `Service Provider Wallet Contract` увеличивается на сумму пополнения депозита. Депозит производителя батарей в `Management Contract` изменяется на сумму, необходимую для регистрации одной батареи с учетом средств, перечисленных для пополнения депозита.

US-[3]USCounter Регистрация батарей с закрепленным сбором за регистрацию батарей

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# vendor.py --batfee`
`Production fee per one battery: 0.01`
`# setyp.py --setfee 0.005`
`Updated successfully`
`# vendor.py --bat 1`
`Created battery with ID: 44bf53f1bda83c64f29a61ed671cfc8ff7fa3b58`

Комментарий: В блокчейн сеть отправляется транзакция на регистрацию батареи, транзакция успешно включена в блок, `Battery Management Contract` можно проинспектировать на предмет добавленных данных. Депозит производителя батарей в `Management Contract` уменьшается на 0.1 *ether*.

US-[3]USCounter Получение размера остатка по депозиту

Использование скрипта
`vendor.py --deposit`
Подключается к узлу Ethereum и выполняет **локальный** вызов функции `Management Contract` получения остатка по депозиту для аккаунта, от которого выполняется вызов.. В терминал выводится полученная информация (в *ether*) в виде десятичной дроби с точностью до 6 знаков после запятой, незначащие завершающие нули не выводятся.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# vendor.py --deposit` `Deposit: 0.095`
Комментарий: Информация успешно получена и выведена в терминал. Нет транзакций в блокчейн для `Management Contract`.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# vendor.py --deposit` `Vendor account is not registered.`
Комментарий: При попытке получения остатка депозита от аккаунта, который не зарегистрирован как производитель батарей, выдается ошибка. Нет транзакций в блокчейн для `Management Contract`.

US-[3]USCounter Пакетная регистрация производимых батарей

При положительном балансе депозита производителя батарей в `Management Contract` регистрировать батареи можно в количестве, соответствующем остатку депозита. Иначе, при регистрации батареи необходимо пополнять депозит до суммы, достаточной для регистрации.

Примечание: Данные сценарии проверяются только в случае успешного прохождения тестирования *"Поштучная регистрация производимых батарей"*.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# vendor.py --bat 3`
`Created battery with ID: 1e6d7ef161b777d3f7422bec456bc251b459e77`
`Created battery with ID: c64a6b0267dc5fc156592b958078644a1946f8e`
`Created battery with ID: a8faf5e89eb2107a74bdb5d29141bf5e4615cd8`

Комментарий: В блокчейн сеть отправляется **одна** транзакция на регистрацию трех батарей, транзакция успешно включена в блок, `Battery Management Contract` можно проинспектировать на предмет добавленных данных. Депозит производителя батарей в `Management Contract` уменьшается на сумму, необходимую для регистрации трех батарей.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# vendor.py --bat 3` `Failed. No enough funds to register object.`
Комментарий: Если при попытке регистрации батарей депозит на `Management Contract` не содержит достаточно средств, то выдается ошибка. Транзакция на регистрацию батарей не отправляется в блокчейн.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# vendor.py --bat 4 0.001`
`Created battery with ID: 8069dfa9abaef192224c962f81a81f6fdf84241f`
`Created battery with ID: cfbc3f5d56753efa436251d382a74b607b3241b0`
`Created battery with ID: 11d87f2214deb3f901fc91acfee053e5b9f1a7c4`
`Created battery with ID: 17a51f0cf04f3544410a079da96327a10e8acd04`

Комментарий: В блокчейн сеть отправляется **одна** транзакция на регистрацию четырех батарей, транзакция успешно включена в блок, `Battery Management Contract` можно проинспектировать на предмет добавленных данных. Баланс `Service Provider Wallet Contract` увеличивается на сумму пополнения депозита. Депозит производителя батарей в `Management Contract` изменяется на сумму, необходимую для регистрации четырех батарей с учетом средств, перечисленных для пополнения депозита.

US-[3]USCounter Безопасная пакетная регистрация производимых батарей

При пакетной регистрации батарей может сложится такая ситуация, что потребляемый при обработке транзакции газ превысит максимальное количество газа, допустимое для транзакций, включенных в блок. В этом случае, регистрация батарей должна происходить в несколько транзакций. Количество идентификаторов батарей, включенных в транзакцию не должно быть предварительно задано, вместо этого оно должно вычисляться автоматически в зависимости от настроек блокчейн — максимальное количество газа, на сколько могут потребить все транзакции, включенные в один блок).

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# vendor.py --bat 5800`
`Created battery with ID: ae318f3b81c25acf9d8eb9cf004b7dc8d3980f7`
`Created battery with ID: ae8eff9801b40a5e25af5212d50e2c7d8ca683f`
`Created battery with ID: c51eb5db9fefc2497203a8f19a74f449d24f4ba`
`. . .`
`Created battery with ID: 121759890801155f310e26b541c861ed5dfc423`
`Created battery with ID: dbcdd9074c66705051cb8645c892080c3c742ed`
`Created battery with ID: e42474a90e439aef1c5a7d872eee5f62b9d8c19`

Комментарий: В блокчейн сеть отправляется **две** транзакции на регистрацию 5800 батарей, обе транзакции успешно включены в **разные** блоки, `Battery Management Contract` можно проинспектировать на предмет добавленных данных. Депозит производителя батарей в `Management Contract` уменьшается на сумму, необходимую для регистрации 5800 батарей.

EPIC-[2]EpicCounter Дистрибьюция батареи

Дистрибьюция батарей происходит в двух случаях:

- установка новой батареи в автомобиль на заводе производителе;
- продажа батарей на замену сервисным центрам и партнерам.

И в том, и в другом случае должен смениться владелец батареи в `Battery Management Contract`.

US-[3]USCounter Поштучная продажа новых батарей

Поскольку сценариев того, как батарея может отпускаться от производителя новым владельца достаточно много, то экономическая составляющая этого процесса не будет покрываться данным сценарием.

Использование скрипта
`# vendor.py --owner <battery_id> <new_owner>`
Подключается к узлу Ethereum и отправляет запрос к `Battery Management Contract` на смену владельца батареи. Для определения текущего владельца батареи в `Management Contract` используется адрес отправителя транзакции. В терминал выводится статус выполнения команды.

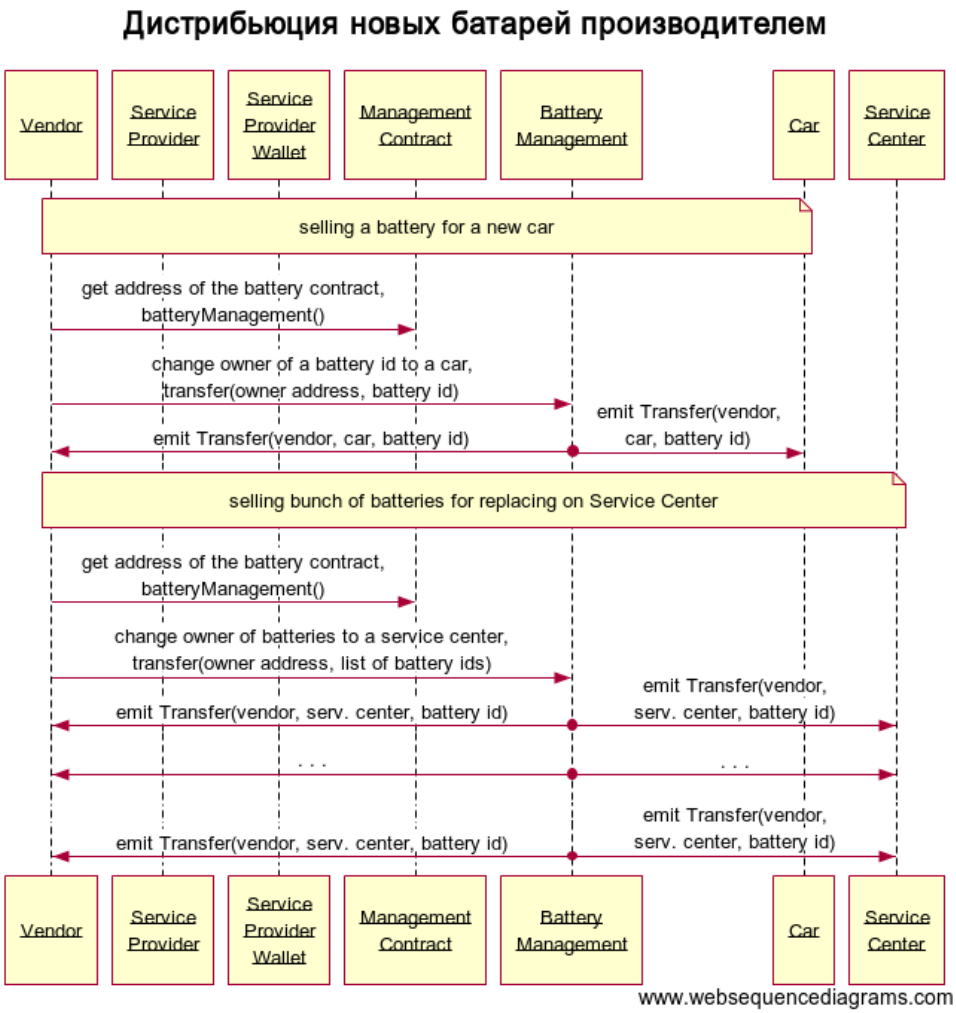

Рис. 9.3: Диаграмма взаимодействия компонент

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# vendor.py --owner ae318f3b81c25acf9d8eb9cf004b7dc8d3980f7 \`
`0x50dfbef376d01e05e134a9b3a322afef50111ec1`
`Success`

Комментарий: В блокчейн сеть отправляется транзакция на изменение владельца батареи, транзакция успешно включена в блок, `Battery Management Contract` можно проинспектировать на предмет изменения владельца батареи.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# vendor.py --owner 121759890801155f310e26b541c861ed5dfc423 \`
`0x0dcd2f752394c41875e259e00bb44fd505297caf`
`Failed. Not allowed to change ownership.`

Комментарий: При попытке смены владельца у батареи, для которой право владения перемещено другому, выдается ошибка. Транзакция на изменения владельца батареи не отправляется в блокчйен. Нет локальных вызов методов контракта для проверки текущего владельца батареи.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# vendor.py --owner ce2e25258e041d07bc3c4906f2b45d68c9aa3420 \`
`0xbe428c3867f05dea2a89fc76a102b544eac7f772`
`Failed. Not allowed to change ownership.`

Комментарий: При попытке смены владельца у батареи на владельца, который не зарегистрирован ни как сервисный центр, ни как электромобиль, выдается ошибка. Транзакция на изменения владельца батареи не отправляется в блокчйен. Нет локальных вызов методов контракта для проверки текущего владельца батареи.

US-[3]USCounter Пакетная продажа новых батарей

Если вместо идентификатора батареи передается путь до файла, то происходит смена владельца для всех батарей, чьи идентификаторы перечислены в файле.

В файле каждый новый идентификатор, записанный в шестнадцатеричном виде и начинающийся с `0x`, находится в новой строке.

Путь до файла может быть, как абсолютным, так и относительным.

Примечание: Данный сценарий проверяются только в случае успешного прохождения тестирования *"Поштучная продажа новых батарей"*.

Подразумевается, что для всех батарей из списка текущий владелец один и тот же. Сценарии, когда у какой-то из батарей нельзя поменять владельца из-за ошибки владения, проверяться не будет.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# vendor.py --owner /path/to/file 0x50dfbef376d01e05e134a9b3a322afef50111ec1` `Success`
Комментарий: В блокчейн сеть отправляется **одна** транзакция на изменение владельца батарей, перечисленных в файле, доступном по пути `/path/to/file`. Транзакция успешно включена в блок, `Battery Management Contract` можно проинспектировать на предмет изменения владельца батарей.

Аппаратный ключ защиты (HASP)

Поскольку, в случае олимпиадной задачи, не предоставляется работать с реальным аппаратным ключом защиты, то функционал ключа будет эмулироваться скриптом на языке Python, он же будет файлом прошивки аппаратного ключа, содержащим статические (read-only memory, хранящая неизменяемые данные специфичные для конкретного аппаратного ключа).

Изменяемые данные (data storage) аппаратного ключа будут эмулироваться через файл, в котором информация хранится в форматае JSON.

EPIC-[2]EpicCounter Генерация прошивок

US-[3]USCounter Генерация прошивок аппаратных ключей батарей

Результатом регистрации батареи будет не только внесение изменений в блок-

чейн, но и генерация файла, содержащего прошивку аппаратного ключа конкретной батареи.

Примечание: Данный сценарий проверяется только в случае успешного прохождения тестирования *"Поштучная регистрация производимых батарей"* или *"Множественная регистрация производимых батарей"*.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# vendor.py --bat 3`
`Created battery with ID: 842d20af461195aaa13f248df340b5e0b3d4042`
`Created battery with ID: ee4d92e6d4183caac6e45b2c31f74e37440ef11`
`Created battery with ID: 6244345b1459bde1cc14be06d11d00421dab3b5`

Комментарий: В директории `firmware`, находящейся в той же директории, что и скрипт `vendor.py`, добавляются три файла `842d20af.py`, `ee4d92e6.py`, `6244345b.py`. Если директории `firmware` не существует, то она автоматически создается. В каждом файле есть глобальная переменная `_privkey`, содержащая набор байт - приватный ключ соответствующей батареи. Приватный ключ может быть проверен на совпадение с батарей, если из него получить Ethereum адрес. Первые четыре байта адреса, записанные в шестнадцатеричном виде, должны совпадать с именем файла. *Примечание:* Если сценарий выполняется с запросом на регистрацию только одной батареи, то соответственно в директории должен появиться только один новый файл.

EPIC-[2]EpicCounter Операции с аппаратным ключом

Имя скрипта
`<battery_id>.py <command>`

Аппаратный ключ работает полностью автономно и не требует никаких данных из блокчейн.

US-[3]USCounter Изменение количества заряда батареи

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# 842d20af.py --charge`
Комментарий: Увеличивает значение в файле, эмулирующем data storage, значение на единицу. Если такого файла не было, то он создается и в него записывается 1. Имя файла с хранилищем данных должно совпадать с именем файла-прошивкой аппаратного ключа. Расширение файла `.json` - файл содержит информацию в формате JSON.

US-[3]USCounter Получение информации о батарее из прошивки

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# 842d20af.py --get`
`4`
`1515748372`
`27`
`8f8f14002e48267c85323a89f0d1c34deda1af13ef39e0aae851cdff0bdde18b`
`3197422d66ae3f1b11950dfbef376d01e05e134a9b3a322afef50111ec9bd656`

Комментарий: Выводит в терминал пять строк (статус батареи): 1. n - текущее количество проведенных зарядов батареи. Берется из файла, эмулирующего data storage, или 0, если такого файла нет. Целое число, максимум $2^{32} - 1$, записано в десятичном формате. 2. t Текущее время в формате unix time - количество секунд с 1 января 1970 года. Целое число, максимум $2^{32} - 1$, записано в десятичном формате. 3. v, r, s - компоненты цифровой подписи, где v - целое число, максимум $2^{32} - 1$, записано в десятичном формате; r, s - наборы байт длиной 32, записанный в шестнадцатеричном формате. Цифровая подпись формируется с использованием приватного ключа, соответствующего данной батареи. Подпись берется для хэш-суммы ($keccak256$) от объединенных вместе количества зарядов батарей и текущего времени ($m = n \cdot 2^{32} + t$). Действительность цифровой подписи может быть проверена через генерацию Ethereum адреса для данных количества зарядов батарей и текущего времени. Первые восемь шестнадцатеричных символа адреса должны совпадать с именем файла прошивки.

Регистрация участников системы

Для замены батарей взаимодействуют два участника системы: сервисный центр и электромобиль. Подразумевается, что электромобиль может быть полностью автономный, именно поэтому он выступает самостоятельным участником.

В общем виде вызов скрипта, отвечающего за операции выполняемые от имени сервисного центра, выглядит следующим образом:

Имя скрипта
`scenter.py <command> [options]`

В тех случаях, когда скрипту необходимо отправлять транзакции для изменения состояния блокчейн, аккаунт и пароль, которые будут использоваться для доступа к приватному ключу для подписи транзакций, должны браться из файла в формате JSON `scenter.json`:

```
{
    "account": "0x5338d77b5905cdeea7c55a1f3a88d03559c36d73",
    "password": "s0me_p455w0rd"
}
```

Скрипт, отвечающего за операции выполняемые от имени электромобиля:

Имя скрипта
`car.py <command> [options]`

Поскольку хранение информации о всей блокчейн в памяти электромобиля не оправдано, то должна предоставляться возможность для подготовки транзакций или подписи данных, передаваемых вне блокчейн. Для этого приватный ключ должен браться из файла в формате JSON `car.json`:

```
{
    "key": "338d77b5905cda7c55a1f3a885338d77b5905cdeea7c55a1f3a88d03559c36d73"
}
```

Все транзакции должны отправляться в сеть блокчейн с использованием данного ключа.

EPIC-[2]EpicCounter Регистрация сервисного центра

US-[3]USCounter Создание аккаунта сервисного центра

Использование скрипта
`scenter.py --new <password>`
Подключается к локальному узлу Ethereum и создает новый аккаунт с паролем, заданным в строке ввода. Также создает файл базы данных `scenter.json`, в котором сохраняется аккаунт и пароль для дальнейшего использования данным производителем. В терминал выводится адрес аккаунта.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# scenter.py --new 's0me_p455w0rd'` `0x5338d77b5905cdeea7c55a1f3a88d03559c36d73`
Комментарий: На локальном узле создан пользователь. В текущей рабочей директории создается файл `scenter.json`, в котором сохраняется аккаунт и пароль в формате `JSON` для дальнейшего использования данным сервисным центра.

US-[3]USCounter Регистрация аккаунта сервисного центра

Использование скрипта
`scenter.py --reg`
Подключается к локальному узлу Ethereum и отправляет запрос к `Management Contract` на регистрацию аккаунта отправителя транзакции в качестве сервисного центра.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`scenter.py --reg` `Registered successfully`
Комментарий: В блокчейн сеть отправляется транзакция на регистрацию аккаунта сервисного центра. Транзакция успешно включена в блок. `Management Contract` можно проинспектировать на предмет изменений.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`scenter.py --reg` `Already registered`
Комментарий: Если при попытке аккаунта сервисного центра в `Management Contract` уже есть такой аккаунт в качестве сервисного центра или электромобиля, то выдается ошибка. Транзакция на регистрацию батареи не отправляется в блокчейн.

EPIC-[2]EpicCounter Регистрация электромобиля

US-[3]USCounter Создание аккаунта электромобиля

Использование скрипта
`car.py --new`
Случайным образом выбирается приватный ключ и записывается в файл `car.json`. В терминал выводится адрес аккаунта.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --new` `0x24a1c06ab6fe0c2ab55858678ee69b1972cbe5c3`
Комментарий: Файл `car.json` создан в текущей рабочей директории.

US-[3]USCounter Получение аккаунта электромобиля

Использование скрипта
`car.py --account`
Выводит в терминал адрес аккаунта автомобиля на основе приватного ключа из файла `car.json`.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --account` `0x24a1c06ab6fe0c2ab55858678ee69b1972cbe5c3`
Комментарий: Адрес аккунта соответствует приватному ключу, из файл `car.json`, расположенному в текущей рабочей директории.

US-[3]USCounter Регистрация аккаунта электромобиля

Использование скрипта
`car.py --reg`
Подключается к локальному узлу Ethereum и отправляет запрос к `Management Contract` на регистрацию аккаунта отправителя транзакции в качестве электромобиля.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`car.py --reg` `Registered successfully`
Комментарий: В блокчейн сеть отправляется транзакция на регистрацию аккаунта электромобиля. Транзакция успешно включена в блок. `Management Contract` можно проинспектировать на предмет изменений.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`car.py --reg` `Already registered`
Комментарий: В блокчейн сеть отправляется транзакция на регистрацию аккаунта электромобиля. Транзакция успешно включена в блок, выясняется, есть такой аккаунт в качестве сервисного центра или электромобиля. Локальных запросов для проверки аккаунта перед отправкой транзакции не создается.

Замена батарей

В данном разделе подробно рассматриваются действия, которые должны выполнить участники во время замены батарей. При этом действия не учитывают нежелательное поведение участников, когда участник сознательно отказывается от сделки или пытается мошенничать. Такие сценарии будут рассмотрены в следующих разделах.

EPIC-[2]EpicCounter Проверка подлинности батареи

Каждый участник системы может проверить батарею на подлинность, если имеет физический доступ к батарее.

US-[3]USCounter Проверка подлинности батареи сервисным центром

Использование скрипта
`scenter.py --verify <path>`
Обращается к файлу с прошивкой батареи и получает информацию о количестве циклов заряда, текущее время и цифровую подпись (статус батареи). После этого подключается к узлу Ethereum и выполняет **локальный** вызов функции проверки данных от батареи. В терминал выводится статус проверки, количество выполненных циклов заряда, идентификатор и название производителя батареи.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# scenter.py --verify firmware/ee4d92e6.py`
`Verified successfully.`
`Total charges: 3`
`Vendor ID: ______`
`Vendor Name: 'Tesla Gigafactory'`

Комментарий: Проверка выполнена успешно. Нет транзакций в блокчейн для проверки батареи с идентификатором ee4d92e6d4183caac6e45b2c31f74e37440ef11.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# scenter.py --verify firmware/00360d2b.py` `Verification failed. Probably the battery forged.`
Комментарий: Проверка показывает, что данные от батареи не проходят валидацию в блокчейн - такая батарея не зарегистрирована. Нет транзакций в блокчейн для проверки батареи с идентификатором 00360d2b7d240ec0643b6d819ba81a09e40e5bcd.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# scenter.py --verify firmware/6244345b.py` `Battery with the same status already replaced. Probably the battery forged.`
Комментарий: Проверка показывает, что батарея с идентификатором 6244345b1459bde1cc14be06d11d00421dab3b5 и данным статусом уже участвовала в замене. Это указывает на возможную подмену прошивки, а, значит, мошенничество. Нет транзакций в блокчейн для проверки батареи с идентификатором 6244345b1459bde1cc14be06d11d00421dab3b5.

US-[3]USCounter Проверка подлинности батареи электромобилем

Использование скрипта
`car.py --verify <path>`
Обращается к файлу с прошивкой батареи и получает информацию о количестве циклов заряда, текущее время и цифровую подпись (статус батареи). После этого подключается к узлу Ethereum и выполняет **локальный** вызов функции проверки данных от батареи. В терминал выводится статус проверки, количество выполненных циклов заряда, идентификатор и название производителя батареи.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# car.py --verify firmware/ee4d92e6.py`
`Verified successfully.`
`Total charges: 3`
`Vendor ID: ______`
`Vendor Name: 'Tesla Gigafactory'`

Комментарий: Проверка выполнена успешно. Нет транзакций в блокчейн для проверки батареи с идентификатором ee4d92e6d4183caac6e45b2c31f74e37440ef11.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --verify firmware/00360d2b.py` `Verification failed. Probably the battery forged.`
Комментарий: Проверка показывает, что данные от батареи не проходят валидацию в блокчейн - такая батарея не зарегистрирована. Нет транзакций в блокчейн для проверки батареи с идентификатором 00360d2b7d240ec0643b6d819ba81a09e40e5bcd.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --verify firmware/6244345b.py` `Battery with the same status already replaced. Probably the battery forged.`
Комментарий: Проверка показывает, что батарея с идентификатором 6244345b1459bde1cc14be06d11d00421dab3b5 и данным статусом уже участвовала в замене. Это указывает на возможную подмену прошивки, а, значит, мошенничество. Нет транзакций в блокчейн для проверки батареи с идентификатором 6244345b1459bde1cc14be06d11d00421dab3b5.

EPIC-[2]EpicCounter Подготовка к замене батарей

Когда электромобиль прибывает в сервисный центр для замены аккумуляторной батареи, то сервисный центр должен получить информацию о батарее, которая уже

установлена в автомобиле. На основании информации о батареях, участвующих в замене, сервисный центр составляет контракт, в рамках которого будет происходить выполнение работ по замене и дальнейшее экономическое взаимодействие.

US-[3]USCounter Получение адреса контракта для замены батареи

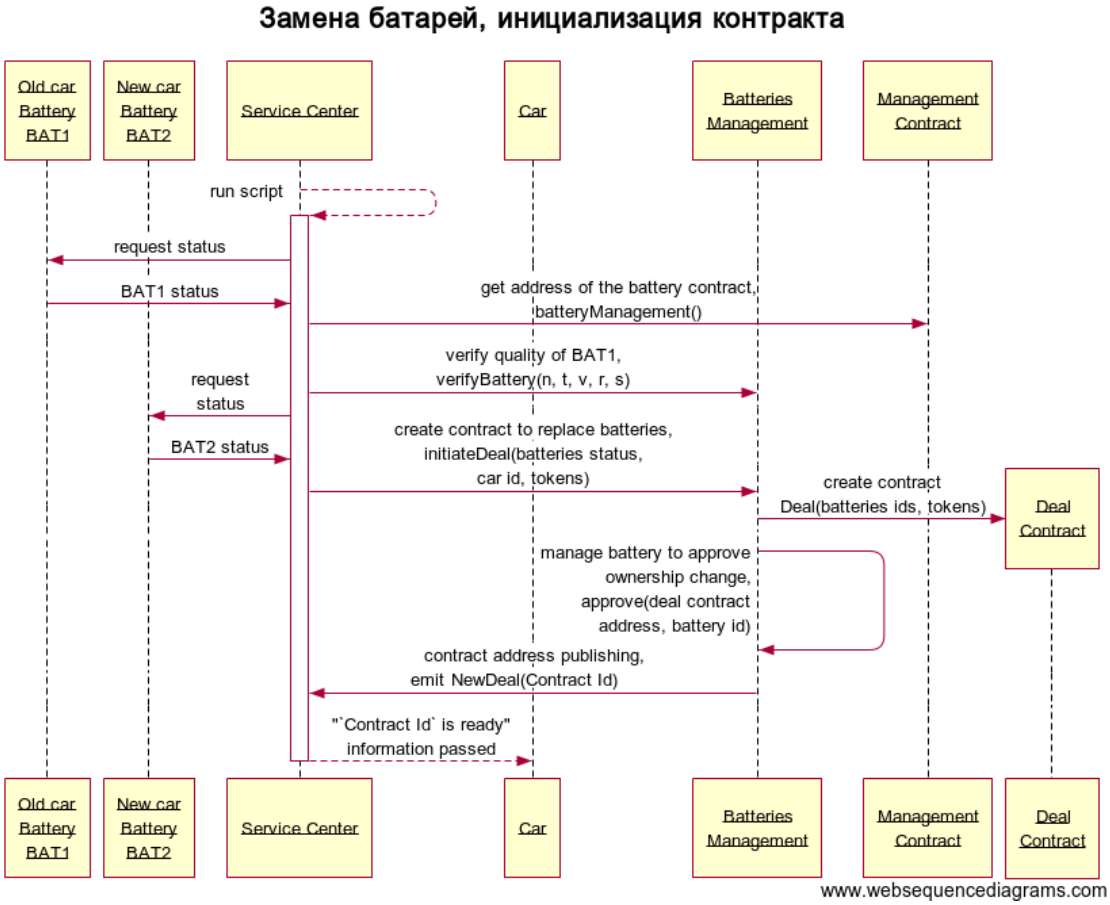

Рис. 9.4: Диаграмма создания контракта на замену батарей

Использование скрипта
`scenter.py --contract <new battery path> <old battery path> <car> <value>`
Обращается к файлам с прошивками батарей и получает информацию о статусе батарей. После этого подключается к узлу Ethereum и отправляет запрос на создание контракта на замену указанных батарей с указанием стоимости работ по замене батарей, заданный через `value` в виде целого числа - количества расчетных токенов. Параметр `car` позволяет ограничить ограничить доступ к контракту только для данного электромобиля. В терминал выводится адрес заключаемого контракта.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

```
# scenter.py --contract firmware/00360d2b.py firmware/ee4d92e6.py \
             0x24a1c06ab6fe0c2ab55858678ee69b1972cbe5c3 100
Deal: 0x3597bfD533a99c9aa083587B074434E61Eb0A258
```

Комментарий: В блокчейн сеть отправляется транзакция на создание контракта, в транзакцию передаются статусы обеих батарей, стоимость работ. Транзакция успешно включена в блок, в ходе ее обработки создается контракт на замену батарей, контракт можно проинспектировать на предмет идентификаторов батарей, участвующих в замене. Нельзя создать еще один контракт на замену, который бы содержал одну из использованных в данном контракте батарей. Результатом проверки того же самого статуса каждой батареи, что использовался в при создании контракта, будет ошибка, что транзакция с таким статусом уже была зарегистрирована в блокчейн. Локальных запросов для проверки статусов батарей перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

```
# scenter.py --contract firmware/00360d2b.py firmware/ee4d92e6.py  \
             0x24a1c06ab6fe0c2ab55858678ee69b1972cbe5c3 100
Deal was not created.
```

Комментарий: Если при отправке транзакции на создание контракта сделки определяется, что такой батареи не зарегистрировано в системе, то выдается ошибка. Контракт на замену батарей не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

```
# scenter.py --contract firmware/00360d2b.py firmware/ee4d92e6.py  \
             0x24a1c06ab6fe0c2ab55858678ee69b1972cbe5c3 100
Deal was not created.
```

Комментарий: Если при попытке создания контракта сделки определяется, что батарея с данным идентификатором в текущий момент уже участвует в каком-то контракте замены, то выдается ошибка. Контракт на замену батарей не создается. Локальных запросов для проверки статусов батарей перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

```
# scenter.py --contract firmware/ee4d92e6.py firmware/00360d2b.py \
             0x24a1c06ab6fe0c2ab55858678ee69b1972cbe5c3 100
Deal was not created.
```

Комментарий: Если при попытке создания контракта сделки определяется, что сервисный центр не является владельцем батареи, которую собирается использовать для замены, то выдается ошибка. Контракт на замену батарей не создается. Локальных запросов для определения прав владения батареей перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

```
# scenter.py --contract firmware/ee4d92e6.py firmware/00360d2b.py \
             0x24a1c06ab6fe0c2ab55858678ee69b1972cbe5c3 100
Deal was not created.
```

Комментарий: Если при попытке создания контракта сделки определяется, что аккаунт электромобиля, использующийся в контракте, не является текущим владельцем батареи, которую собирается использовать для замены, то выдается ошибка. Контракт на замену батарей не создается. Локальных запросов для проверки статусов батарей перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# scenter.py --contract firmware/ee4d92e6.py firmware/00360d2b.py \`
`0x2a0c0dbecc7e4d658f48e01e3fa353f44050c208 100`
`Deal was not created.`

Комментарий: Если при попытке создания контракта сделки определяется, что аккаунт отправитель транзакции не принадлежит сервисному центру, зарегистрированному в системе, то выдается ошибка. Контракт на замену батарей не создается. Локальных запросов для проверки статусов батарей перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# scenter.py --contract firmware/ee4d92e6.py firmware/00360d2b.py \`
`0x16332b24b07b2ee0ad9203587f9e5ffd0b30b851 100`
`Deal was not created.`

Комментарий: Если при попытке создания контракта сделки определяется, что несмотря на то, что аккаунт отправителя транзакции зарегистрирован как сервисный центр, ни один производитель батарей никогда не передавал ему права на владение батареями, то выдается ошибка. Контракт на замену батарей не создается. Локальных запросов для проверки статусов батарей перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# scenter.py --contract firmware/ee4d92e6.py firmware/00360d2b.py \`
`0x38c3c36677ed76d39efa17ab575e387818ec75d7 100`
`Deal was not created.`

Комментарий: Если при попытке создания контракта сделки определяется, что адрес, участвующий в сделке в качестве получателя новой батареи, не принадлежит электромобилю, зарегистрированному в системе, то выдается ошибка. Контракт на замену батарей не создается. Локальных запросов для проверки статусов батарей перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# scenter.py --contract firmware/ee4d92e6.py firmware/00360d2b.py \`
`0xdcfd11e4546551a09bc5c8529490347a9840162c 100`
`Deal was not created.`

Комментарий: Если при попытке создания контракта сделки определяется, что несмотря на то, что адрес, участвующий в сделке в качестве получателя новой батареи, зарегистрирован в системе как электромобиль, ни один производитель батарей никогда не передавал ему права на владение батареями, то выдается ошибка. Контракт на замену батарей не создается. Локальных запросов для проверки статусов батарей перед отправкой транзакции не создается.

US-[3]USCounter Отправка запроса на отмену сделки

И сервисный центр может отменить сделку сделку после создания соответствующего контракта, но до того, как электромобиль послал запрос на согласие.

Использование скрипта
`scenter.py --canceldeal <contract>`
Подключается к узлу Ethereum и посылает транзакцию для отмены контракта.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# scenter.py --canceldeal 0xa650c24c4a9aadbb05e6047ca3598c93f5889be1` `Deal canceled successfully`
Комментарий: В блокчейн сеть отправляется транзакция на отмену контракта. Транзакция успешно включена в блок. `Battery Management Contract` можно проинспектировать, что владелец старой и новой батареи остались прежними. С этого момента можно создать еще один контракт на замену, который бы содержал любую из использованных в данном контракте батарей. Контракт сделки недоступен для проведения дальнейших операций

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# scenter.py --canceldeal 0xa030017e8f8d12620ae4ad13066b0df905f91d30` `Cannot cancel the deal`
Комментарий: Если при попытке отмены сделки определяется, что электромобиль уже подтвердил согласие со сделкой, то выдается ошибка. Транзакция на отмену сделки не отправляется в блокчейн.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# scenter.py --canceldeal 0xa030017e8f8d12620ae4ad13066b0df905f91d30` `Cannot cancel the deal`
Комментарий: Если при попытке отмены сделки определяется, что данная сделка была отменена (сервисным центром или электромобилем) или завершена, то выдается ошибка. Транзакция на отмену сделки не отправляется в блокчейн.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# scenter.py --canceldeal 0x71819a042e604ba76ffd54cc1776f4fd3d18704f` `Cannot cancel the deal`
Комментарий: Если при попытке отмены сделки определяется, что аккаунт, который использовался для подписи отправленной транзакции, не соответствует адресу сервисного центра, зарегистрированного контракте сделки, то выдается ошибка. Транзакция на отмену сделки не отправляется в блокчейн.

US-[3]USCounter Получение информации, что контракт замены в ожидании подтверждения

Использование скрипта
`scenter.py --status <contract>`
Подключается к узлу Ethereum и выполняет **локальный** вызов функции получения статуса контракта. В терминал выводится строка, обозначающая в какой фазе находится контракт.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# scenter.py --status 0x3597bfD533a99c9aa083587B074434E61Eb0A258` `Waiting for agreement`
Комментарий: Статус определился успешно. В блокчейн нет транзакций для контракта 0x3597bfD533a99c9aa083587B074434E61Eb0A258.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# scenter.py --status 0x87B074434E61Eb0A2583597bfD533a99c9aa0835` `Not found`
Комментарий: Статус не может определиться по причине того, что указанный адрес либо не является адресом контракта на замену батарей, либо был отменен, либо был завершен, либо вообще не существует в локальной копии блокчейн. В блокчейн нет транзакций для контракта 0x87B074434E61Eb0A2583597bfD533a99c9aa0835.

EPIC-[2]EpicCounter Принятие условий контракта на замену батарей

После создания контракта на замену батарей, электромобиль может ознакомиться с условиями контракта, где указывается:

- Какие батареи участвуют в замене: идентификатор и название производителя, количество циклов заряда каждой батареи.
- Количество расчетных токенов, определяющее компенсацию за разность амортизации батарей.
- Количество расчетных токенов, определяющих оплату выполнения работ по замене батареи.

Если условия контракта принимаются, электромобиль подтверждает согласие на замену перечислением необходимого количества токенов на счет контракта на замену батареи.

Компенсация за разность амортизации батарей определяется контрактом по следующим правилам:

Таблица 9.1: Определение компенсации при замене батарей

	$0 \leqslant N_{new} < 50$	$50 \leqslant N_{new} < 150$	$150 \leqslant N_{new} < 300$	$N_{new} > 300$
$0 \leqslant N_{old} < 50$	Δ	N/A	N/A	N/A
$50 \leqslant N_{old} < 150$	$\Delta \cdot 1,003^{\Delta}$	Δ	N/A	N/A
$150 \leqslant N_{old} < 300$	$\Delta \cdot 1,0035^{\Delta}$	$\Delta \cdot 1,003^{\Delta}$	Δ	N/A
$N_{old} > 300$	1000	1000	1000	0

где

- N_{old} - количество циклов заряда, полученных из статуса батареи, установленной в автомобиле;
- N_{new} - количество циклов заряда, полученных из статуса батареи, подготовленной в сервисном центре для замены;
- $\Delta = N_{old} - N_{new}$;
- N/A - поскольку подразумевается, что сервисный центр всегда предлагает такие батареи, чей количество циклов заряда выше или равно циклам заряда батареи из электромобиля, то другие сценарии не рассматриваются.

Итоговое количество токенов должно быть представлено числом, округленным посредством отбрасывания дробной части.

US-[3]USCounter Получение условий контракта

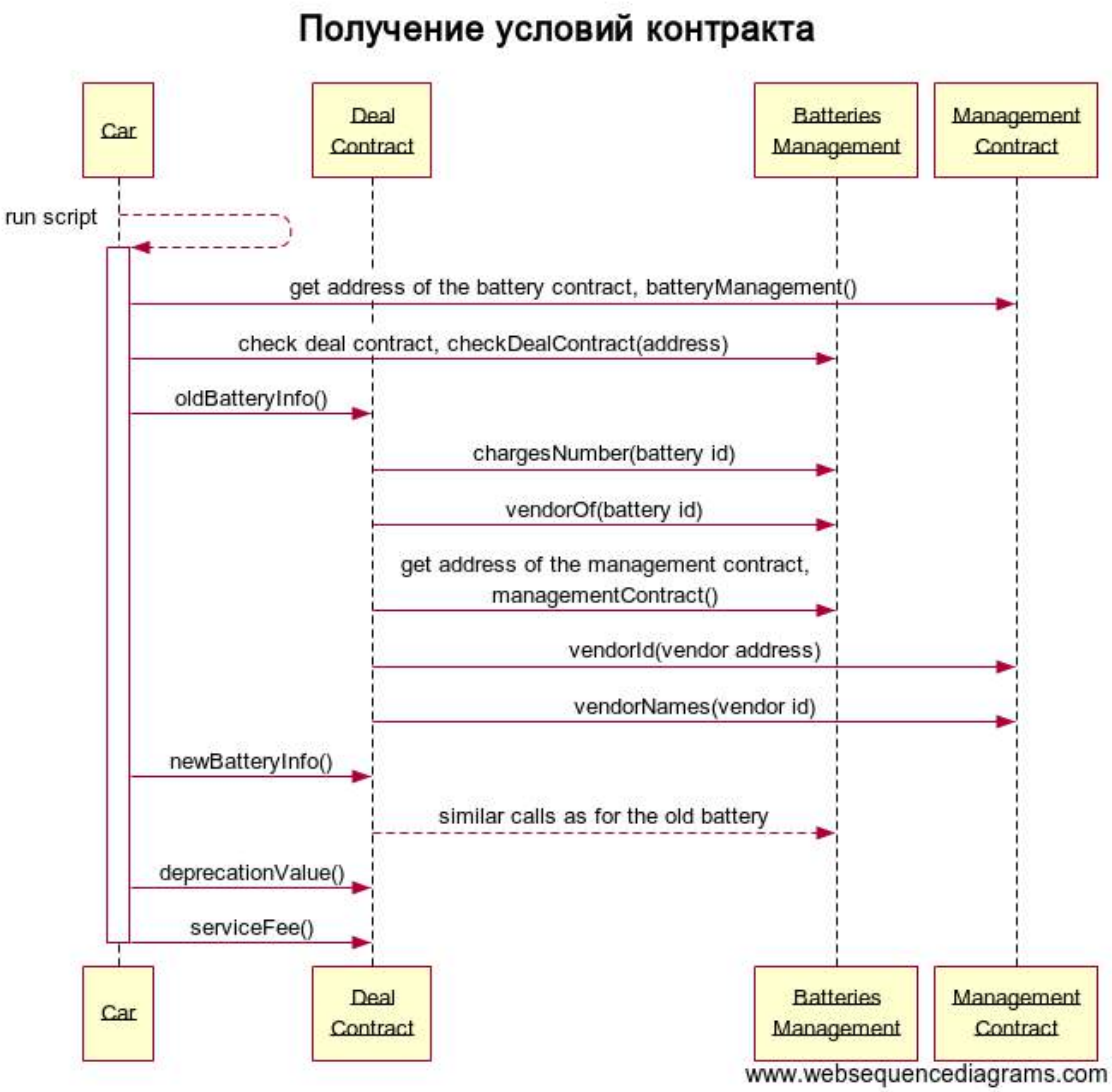

Рис. 9.5: Диаграмма получения информации о контракте

Использование скрипта
`car.py --info <contract>`
Подключается к узлу Ethereum и выполняет **локальный** вызов функции получения данных о сделке из контракта. Данные о сделке выводятся в терминал.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

`# car.py --info 0x3597bfD533a99c9aa083587B074434E61Eb0A258`
`OLD BATTERY:`
`Total charges: 27`
`Vendor ID: ______`
`Vendor Name: 'Nissan Energy'`
`NEW BATTERY:`
`Total charges: 3`
`Vendor ID: ______`
`Vendor Name: 'Tesla Gigafactory'`
`WEAR COMPENSATION: 24`
`REPLACEMENT WORK: 100`

Комментарий: Информация о сделке выведена в терминал. Нет транзакций в блокчейн для получения информации о сделке.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --info 0x86fa049857e0209aa7d9e616f7eb3b3b78ecfdb0` `No deal registered.`
Комментарий: Если данная сделка не зарегистрирована в `Battery Management Contract`, то выдается ошибка.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --info 0x4092678e4e78230f46a1534c0fbc8fa39780892b` `Cannot check the deal.`
Комментарий: Если данная сделка была отменена или завершена, то выдается ошибка.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --info 0xaef53aa490e632bd4870296599efeec241fac96a` `Cannot check the deal.`
Комментарий: Если аккаунт автомобиля, выполняющий запрос, не является владельцем батаери, участвующей данной сделке, то выдается ошибка.

US-[3]USCounter Подтверждение согласия с условиями контракта

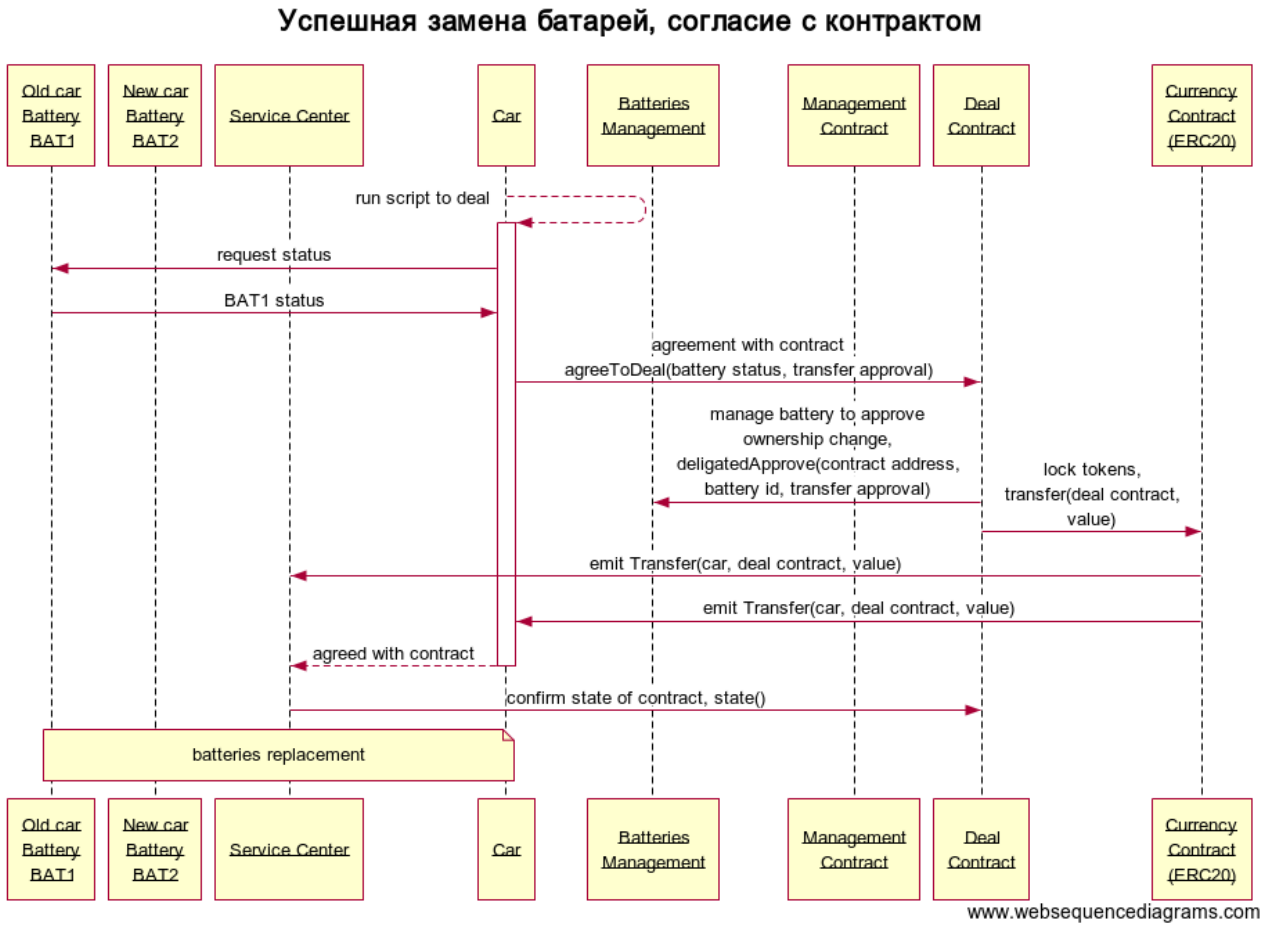

Рис. 9.6: Диаграмма подтверждения контракта

Использование скрипта
`car.py --deal <old battery path> <contract>`
Обращается к файлу с прошивкой батарей и получает информацию о статусе батарей. Подключается к узлу Ethereum и посылает транзакцию для подтверждения согласия с условиями контракта.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

```
# car.py --deal firmware/ee4d92e6.py 0x3597bfD533a99c9aa083587B074434E61Eb0A258
Confirmation of 124 tokens lock commited.
```

Комментарий: В блокчейн сеть отправляется транзакция на согласие с контрактом. Транзакция успешно включена в блок. Контракт расчетных токенов может быть проинспектирован на изменение баланса контракта на замену батарей. Результатом проверки того самого же статуса батареи, что использовался в при согласии с условиями контракта, будет ошибка, что транзакция с таким статусом уже была зарегистрирована в блокчейн. Нельзя создать еще один контракт на замену, который бы содержал одну из использованных в данном контракте батарей.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

```
# car.py --deal firmware/411b6845.py 0x3597bfD533a99c9aa083587B074434E61Eb0A258
Not confirmed.
```

Комментарий: В блокчейн сеть отправляется транзакция на согласие с контрактом. Транзакция успешно включена в блок, но в ходе ее обработки определяется отсутствие достаточного количества расчетных токенов на балансе электромобиля. Изменение баланса контракта на замену батарей в контракте токена не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки баланса токенов перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

```
# car.py --deal firmware/ee4d92e6.py 0x852A0bE16E3597bfD533a99c9aa083587B074434
Not confirmed.
```

Комментарий: В блокчейн сеть отправляется транзакция на согласие с контрактом. Транзакция успешно включена в блок, но в ходе ее обработки определяется, что аккаунт электромобиля, использующийся для подтверждения согласия с условиями контракта, не указан в качестве участника контракта (в качестве электромобиля). Изменение баланса контракта на замену батарей в контракте токена не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки участников контракта перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

```
# car.py --deal firmware/11ef4074.py 0x852A0bE16E3597bfD533a99c9aa083587B074434
Not confirmed.
```

Комментарий: В блокчейн сеть отправляется транзакция на согласие с контрактом. Транзакция успешно включена в блок, но в ходе ее обработки определяется, что цифровая подпись статуса батареи, указывает на батарею, отличающуюся от того, что была использована для контракта. Изменение баланса контракта на замену батарей в контракте токена не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки батареи перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --deal firmware/49b2523c.py 0x52bc44d5378309ee2abf1539bf71de1b7d7be3b5` `Not confirmed.`
Комментарий: В блокчейн сеть отправляется транзакция на согласие с контрактом. Транзакция успешно включена в блок, но в ходе ее обработки определяется, что временная метка, включенная в статус батареи, установленной в электромобиле, меньше либо равна той, что была использована для создания контракта. Изменение баланса контракта на замену батарей в контракте токена не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки батареи перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --deal firmware/e45b2c31.py 0x3597bfD533a99c9aa083587B074434E61Eb0A258` `Not confirmed.`
Комментарий: В блокчейн сеть отправляется транзакция на согласие с контрактом. Транзакция успешно включена в блок, но в ходе ее обработки определяется, что батарея с данным идентификатором не принадлежит данному электромобилю. Изменение баланса контракта на замену батарей в контракте токена не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --deal firmware/2c4fcd3f.py 0x80a7e048f37a50500351c204cb407766fa3bae7f` `Not confirmed.`
Комментарий: В блокчейн сеть отправляется транзакция на согласие с контрактом. Транзакция успешно включена в блок, но в ходе ее обработки определяется, что данная сделка была отменена (сервисным центром или электромобилем) или завершена. Изменение баланса контракта на замену батарей в контракте токена не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

US-[3]USCounter Подтверждение готовности к физической замене батарей

Как только электромобиль подтвердил свое согласие с контрактом замены, сервисный центр должен иметь возможность получить информацию об этом, чтобы начать физическую замену батарей.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# scenter.py status 0x3597bfD533a99c9aa083587B074434E61Eb0A258` `Battery replacement confirmed`
Комментарий: Статус определился успешно. В блокчейн нет транзакций для контракта 0x3597bfD533a99c9aa083587B074434E61Eb0A258.

EPIC-[2]EpicCounter Подтверждение выполнения условий контракта после замены батарей

После физической замены батарей автомобиль проверяет то, что новая батарея

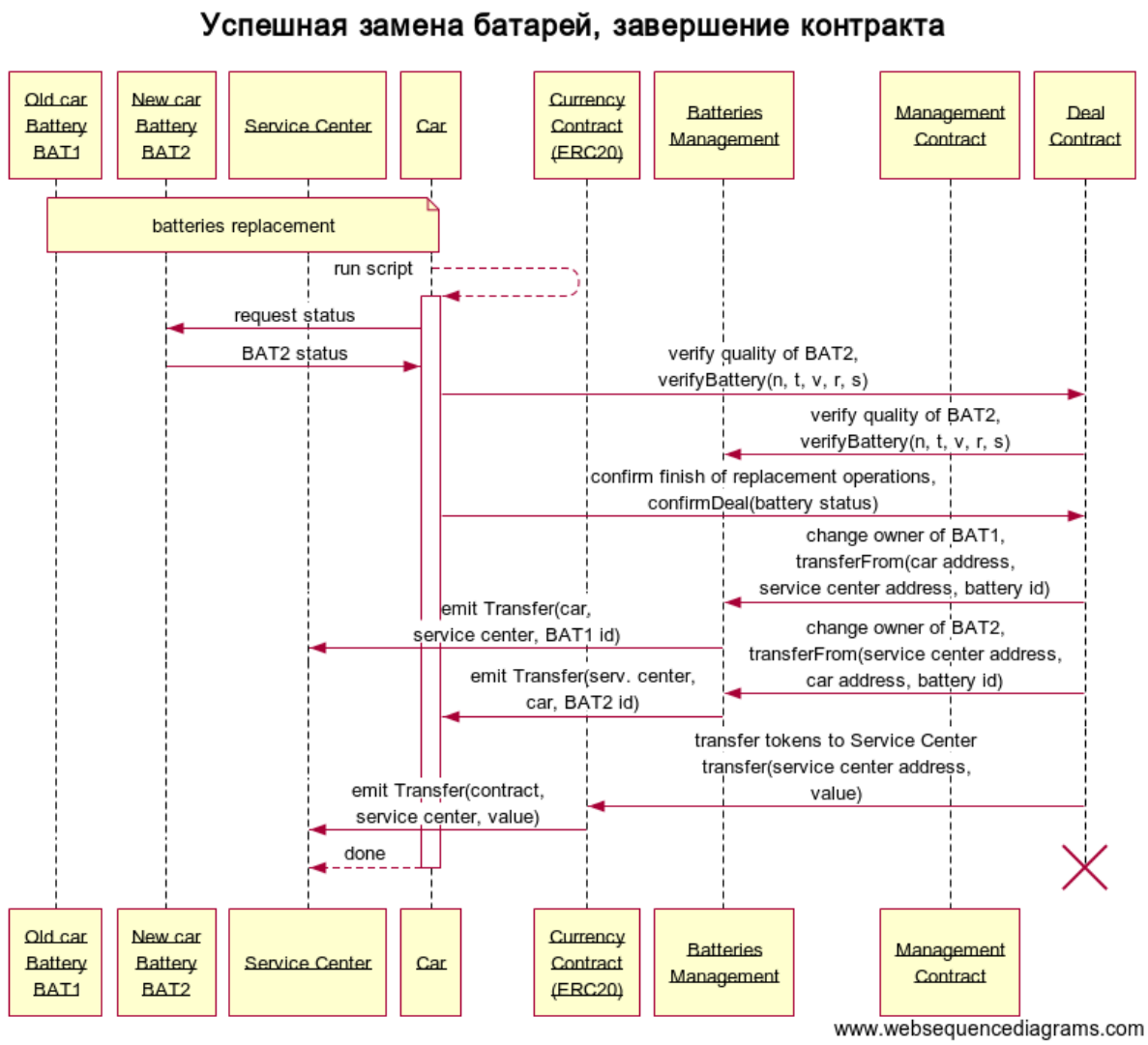

Рис. 9.7: Диаграмма завершения контракта

совпадает с той, что указана в контракте, и подтверждает завершение платежа к сервисному центру. Данное действие также меняет владельца батареи в блокчейн.

Использование скрипта
`car.py --confirm <new battery path> <contract>`
Обращается к файлу с прошивкой батарей и получает информацию о статусе батарей. Подключается к узлу Ethereum и посылает транзакцию для завершения оплаты по контракту замены батарей.

US-[3]USCounter Подтверждение контракта

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

```
# car.py --confirm firmware/00360d2b.py 0x3597bfD533a99c9aa083587B074434E61Eb0A258
124 tokens paid for battery replacement.
```

Комментарий: В блокчейн сеть отправляется транзакция на согласие с завершением работ по контракту. Транзакция успешно включена в блок. Контракт расчетных токенов может быть проинспектирован на увеличение баланса сервисного центра. `Battery Management Contract` можно проинспектировать, что владелец старой и новой батареи изменились. Результатом проверки того же самого статуса батареи, что использовался в при подтверждении контракта, будет ошибка, что транзакция с таким статусом уже была зарегистрирована в блокчейн. С этого момента можно создать еще один контракт на замену, который бы содержал любую из использованных в данном контракте батарей.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

```
# car.py --confirm firmware/d819ba81.py 0x3597bfD533a99c9aa083587B074434E61Eb0A258
Not confirmed.
```

Комментарий: В блокчейн сеть отправляется транзакция на согласие с завершением работ по контракту. Транзакция успешно включена в блок, но в ходе ее обработки определяется, что данная батарея изначально не была указана для участия в замене. Изменение баланса в контракте токенов не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

```
# car.py --confirm firmware/d819ba81.py 0x3597bfD533a99c9aa083587B074434E61Eb0A258
Not confirmed.
```

Комментарий: В блокчейн сеть отправляется транзакция на согласие с завершением работ по контракту. Транзакция успешно включена в блок, но в ходе ее обработки определяется, что батарея с данным идентификатором и с таким же статусом уже участвовала в контракте замены. Изменение баланса в контракте токенов не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

```
# car.py --confirm firmware/94018902.py 0x0d17628ea8a4f14c94142b18761a1b26d42d1546
Not confirmed.
```

Комментарий: В блокчейн сеть отправляется транзакция на согласие с завершением работ по контракту. Транзакция успешно включена в блок, но в ходе ее обработки определяется, что временная метка, включенная в статус новой батареи, установленной в электромобиле, меньше либо равна той, что была использована для создания контракта. Изменение баланса в контракте токенов не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --confirm firmware/d819ba81.py 0x3597bfD533a99c9aa083587B074434E61Eb0A258` `Not confirmed.`
Комментарий: В блокчейн сеть отправляется транзакция на согласие с завершением работ по контракту. Транзакция успешно включена в блок, но в ходе ее обработки определяется, что аккаунт электромобиля, использующийся для подтверждения контракта не указан в качестве участника контракта (в качестве электромобиля). Изменение баланса в контракте токенов не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --confirm firmware/9cc3bf7a.py 0x6f7ded7500aff3be1015caa65c063bb3f764c581` `Not confirmed.`
Комментарий: В блокчейн сеть отправляется транзакция на согласие с завершением работ по контракту. Транзакция успешно включена в блок, но в ходе ее обработки определяется, что данная сделка была отменена (сервисным центром или электромобилем) или завершена. Изменение баланса в контракте токенов не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --confirm firmware/5444fbbc.py 0x32e90a83c861b9d81379ac13d063efe70e0dc327` `Not confirmed.`
Комментарий: В блокчейн сеть отправляется транзакция на согласие с завершением работ по контракту. Транзакция успешно включена в блок, но в ходе ее обработки определяется, что электромобиль еще не послал подтверждение на согласие со сделкой. Изменение баланса в контракте токенов не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

Защита интересов участников сделки

Нельзя полагаться на то, что сделка будет всегда проходить по успешному сценарию. Если бы это было так, то и решение на платформе блокчейн было бы не нужно.

Следовательно, нужно предусмотреть такие варианты событий, когда участники по стечению обстоятельств (сбой системы питания электромобиля) или по злому умыслу (мошенничество) отходят от предписанного сценария.

EPIC-[2]EpicCounter Отказ сервисного центра в замене батареи

Возможна такая ситуация, когда электромобиль согласился со сделкой, но сервисный центр не производит замену. Это приводит к тому, что средства автомобиля (расчетные токены) блокируются на счету контракта сделки. Следовательно, нужно предоставить возможность электромобилю сделать запрос на их разблокирование.

Гарантией того, что замены не было, является новый статус батареи, которая все еще установлена в электромобиле. Чтобы указать, что замены произведено не было, нужно подтвердить контракту владение батарей через предъявление статуса, отличающегося от того, который использовался, чтобы подтвердить согласие с контрактом.

После разблокирования токенов, контракт считается отмененным.

При этом возможно мошенничество, когда автомобиль получает два статуса от батареи: один, чтобы подтвердить согласие со сделкой, и сразу же второй, чтобы даже при отсутствии старой батареи, можно было бы вернуть себе расчетные токены вместо завершения сделки. Следовательно, поскольку статус включает в себя временную метку, можно предусмотреть такую логику обработки запроса на разблокирование, когда между соседними статусами, пройдет не меньше определенного промежутка времени. Данное пороговое значение устанавливается в значение по-умолчанию (3600 секунд) при регистрации контракта `Battery Management Contract` и передается потом в контракт сделки при ее создании. `Battery Management Contract` должен предоставлять способ для изменения значения временного промежутка, в течение которого запросы на разблокирование токенов не будет применяться. Данное значение будет применяться только для вновь созданных сделок.

US-[3]USCounter Установка времени запрета разблокирование токенов

Использование скрипта
`setup.py --release <timeout>`
Подключается к узлу Ethereum и отправляет запрос к `Battery Management Contract` на изменение временного промежутка запрета разблокирования расчетных токенов.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# setup.py --release 600` `Updated successfully`
Комментарий: В блокчейн сеть отправляется транзакция. Транзакция успешно включена в блок. Сделки, созданные после этой транзакции, используют данный промежуток времени для разрешения разблокирования расчетных токенов.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# setup.py --release 600` `No permissions to change token release timeout`
Комментарий: Если попытка изменения временного промежутка запрета разблокирование токенов происходит не от аккаунта компании производителя ПО, то выдается ошибка. Транзакция на регистрацию производителя не отправляется в блокчейн.

US-[3]USCounter Запрос на разблокирование токенов

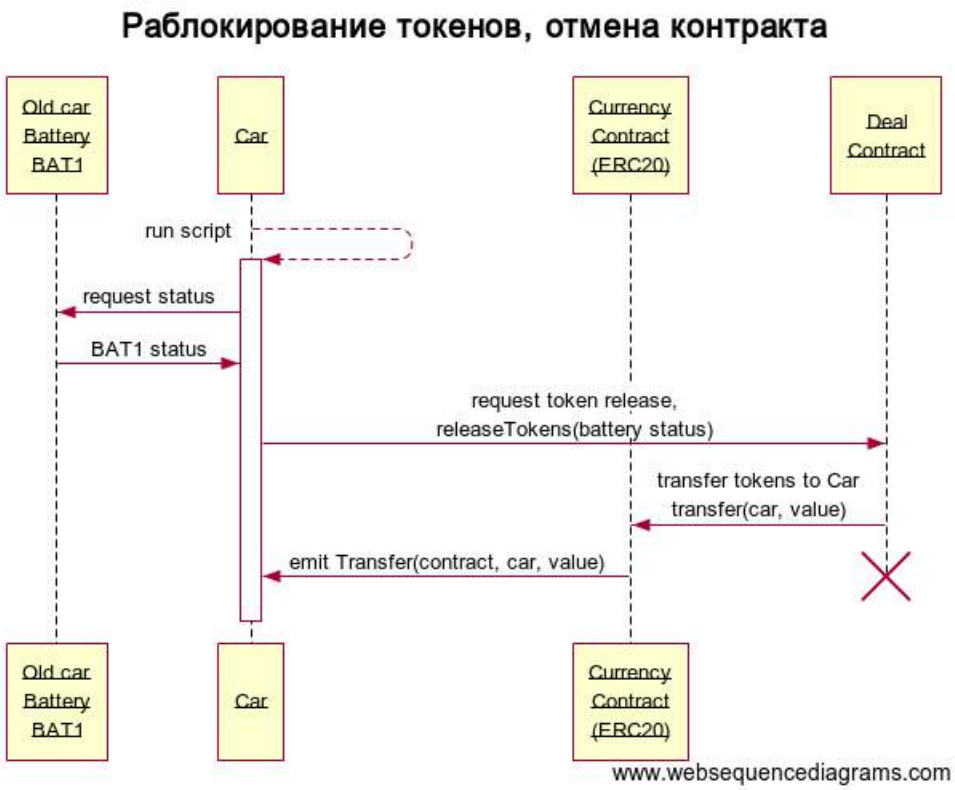

Рис. 9.8: Диаграмма отмены контракта

Использование скрипта
`car.py --release <old battery path> <contract>`
Обращается к файлу с прошивкой батарей и получает информацию о статусе батарей. Подключается к узлу Ethereum и посылает транзакцию для разблокирование токенов. Выводит на экран количество разблокированных расчетных токенов.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --release firmware/10052a52.py 0x9d2d7931b0edf8709260e3f7ffce74f2b3e569d6` `124 tokens unlocked successfully.`
Комментарий: В блокчейн сеть отправляется транзакция на разблокирование токенов. Транзакция успешно включена в блок. Контракт токенов может быть проинспектирован на увеличение баланса электромобиля. `Battery Management Contract` можно проинспектировать, что владелец старой и новой батареи не изменились. Результатом проверки того же самого статуса батареи, что использовался в при разблокировании токенов, будет ошибка, что транзакция с таким статусом уже была зарегистрирована в блокчейн. С этого момента можно создать еще один контракт на замену, который бы содержал любую из использованных в данном контракте батарей.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

```
# car.py --release firmware/2339a242.py 0x356dc7a098f26e720db68a7b1b0c62993585c504
Not confirmed.
```

Комментарий: В блокчейн сеть отправляется транзакция на разблокирование токенов. Транзакция успешно включена в блок, но в ходе ее обработки определяется, что данная батарея изначально не была указана для участия в замене. Изменение баланса в контракте токенов не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

```
# car.py --release firmware/5c3cde51.py 0xe010407d8e423e7a374f6fc81136fa124e3b3d57
Not confirmed.
```

Комментарий: В блокчейн сеть отправляется транзакция на разблокирование токенов. Транзакция успешно включена в блок, но в ходе ее обработки определяется, что батарея с данным идентификатором и с таким же статусом уже участвовала в контракте замены. Изменение баланса в контракте токенов не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

```
# car.py --release firmware/7db4ff5b.py 0x40d6ea55a436f814de4210d0de444fe52fef3946
Not confirmed.
```

Комментарий: В блокчейн сеть отправляется транзакция на разблокирование токенов. Транзакция успешно включена в блок, но в ходе ее обработки определяется, что временная метка, включенная в статус батареи, меньше, чем ожидаемая с учетом временного промежутка запрета разблокирования токенов. Изменение баланса в контракте токенов не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

```
# car.py --release firmware/4998c293.py 0xca8d895baf2bf90810dc65507d2625c509b16c22
Not confirmed.
```

Комментарий: В блокчейн сеть отправляется транзакция на согласие с завершением работ по контракту. Транзакция успешно включена в блок, но в ходе ее обработки определяется, что аккаунт электромобиля, использующийся для разблокирования токенов, не указан в качестве участника контракта (в качестве электромобиля). Изменение баланса в контракте токенов не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py --release firmware/b3853f71.py 0xd43d5d0cc33f2832da514476441098abad1e241e` `Not confirmed.`
Комментарий: В блокчейн сеть отправляется транзакция на согласие с завершением работ по контракту. Транзакция успешно включена в блок, но в ходе ее обработки определяется, что данная сделка была отменена (сервисным центром или электромобилем) или завершена. Изменение баланса в контракте токенов не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

EPIC-[2]EpicCounter Разрешение споров после замены батарей

Следующая возможная ситуация, которая может возникнуть - это установка сервисным центром не той батареи, которая заявлялась при создании сделки, как неумышленно, так и злонамеренно. Причем это может и не оказаться правдой, когда злоумышленником выступает электромобиль, пытаясь сформировать отрицательную репутацию сервисного центра.

Тогда, после сообщения электромобилем о возникшей проблеме, возможны четыре возможных сценария:

1. Сервисный центр соглашается поставить верную батарею. Тогда сделка дальше движется по запланированному сценарию.
2. Сервисный центр признает свою ошибку и соглашается вернуть старую батарею. В этом случае замены не происходит, покрывать затраты сервисного центра не нужно. А электромобиль отправляет запрос на разблокирование токенов.
3. Электромобиль формирует атаку *Denial of Service* и блокирует работу сервисного центра, ложно сообщая о неправильной батарее (для этого годятся любые данные) и не отправляя запрос о завершении сделки. Один из возможных способов решения данной проблемы - введение страховых депозитов и системы скоринга.

   Тогда электромобиль за некорректное поведение получает снижение скоринга независимыми арбитрами (например, блокировка ведения бизнеса может рассматриваться полицией), что приводит к удорожанию страхового депозита электромобиля.

   Данный функционал не будет рассматриваться в рамках данной задачи.
4. Сервисный центр отказывается менять батарею, что может привести к тому, что из-за разряженной поставленной батареи, электромобиль не будет иметь возможность уехать из сервисного центра. Поскольку данная ситуация расценивается, как мошенничество со стороны сервисного центра, то к решению проблемы привлекаются независимые арбитры (например, полиция).

   Экономической защитой от такого поведения, может также служить система страховых депозитов при создании сделки и ведение скоринга всех сервисных центров. Снижение скоринга из-за неприемлемого поведения конкретного участника приводило бы к увеличению страхового депозита.

   Данный функционал не будет рассматриваться в рамках данной задачи.

US-[3]USCounter Отсутствие подтверждения замены батарей для завершения сделки

Еще одной ситуацией является отсутствие подтверждения согласия с заменой батарей от электромобиля для заверешения сделки.

Это приводит к тому, что на счет сервисного центра не перечисляются расчетные токены. Тогда, по аналогии с запросом на разблокирование токенов электромобилем, должен быть запрос на разблокирование токенов сервисным центром.

Разблокирование токенов сервисным центром также не должно позваляться в течение определенного промежутка времени. Поэтому для проверки времени запроса на разблокирование будет использоваться те же самые данные, что и для варианта с электромобилем.

После разблокирования токенов, контракт считается отмененным.

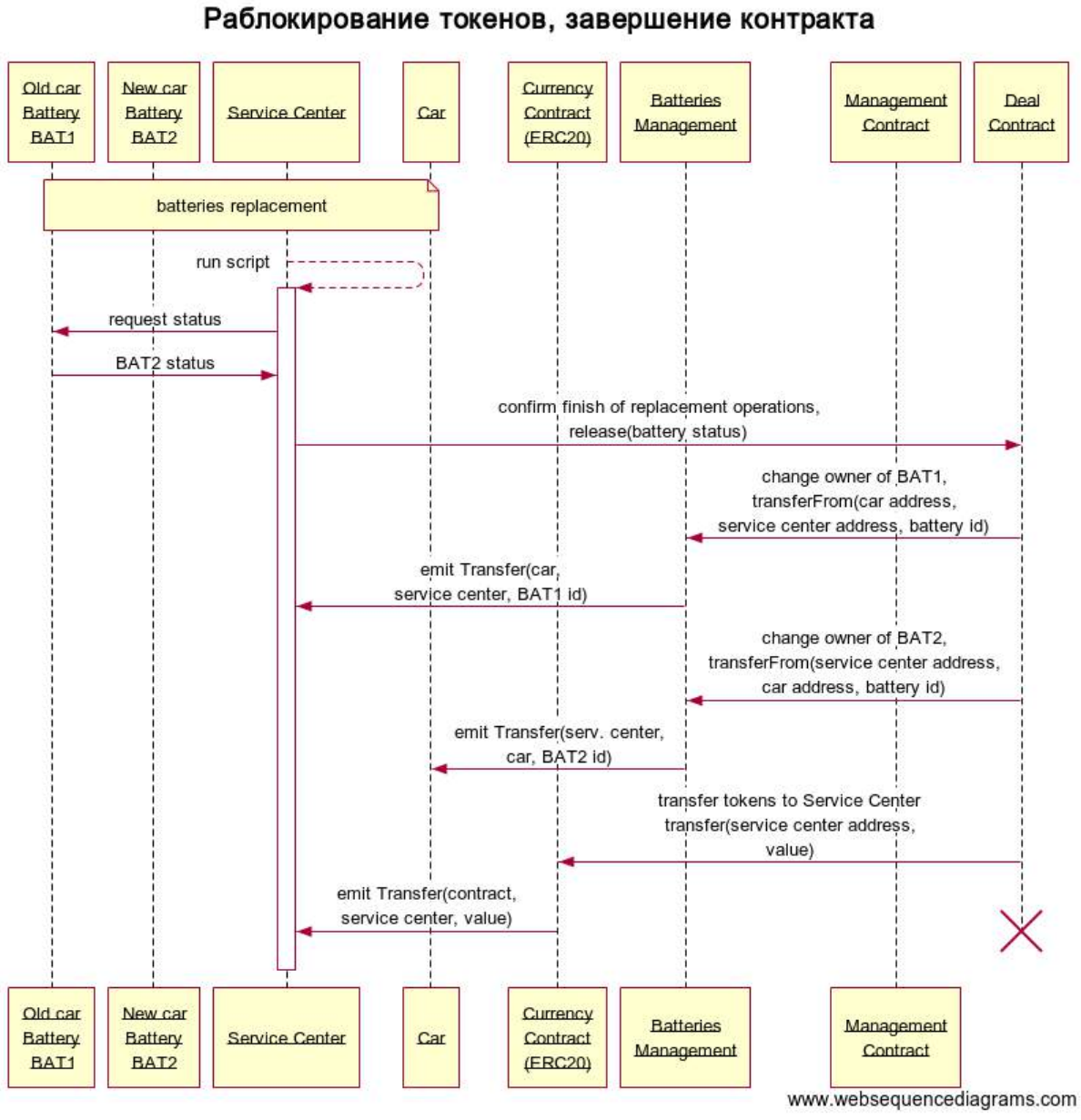

Рис. 9.9: Диаграмма разблокирования токенов и завершения контракта

Использование скрипта

```
scenter.py --release <old battery path> <contract>
```

Обращается к файлу с прошивкой батарей и получает информацию о статусе батарей. Подключается к узлу Ethereum и посылает транзакцию для разблокирование токенов. Выводит на экран количество разблокированных расчетных токенов.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

```
# scenter.py --release firmware/c412dbe6.py 0x9bb1fd0466a1a77714f7e0ce75c69740468ee56f
124 tokens unlocked successfully.
```

Комментарий: В блокчейн сеть отправляется транзакция на разблокирование токенов. Транзакция успешно включена в блок. Контракт токенов может быть проинспектирован на увеличение баланса сервисного центра. `Battery Management Contract` можно проинспектировать, что на изменение прав владения у старой и новой батарей. Результатом проверки того же самого статуса батареи, что использовался в при разблокировании токенов, будет ошибка, что транзакция с таким статусом уже была зарегистрирована в блокчейн. С этого момента можно создать еще один контракт на замену, который бы содержал любую из использованных в данном контракте батарей.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

```
# scenter.py --release firmware/2339a242.py 0x2e5990ecb06d4ef63400d7a4888db09c7a31d0b5
Not confirmed.
```

Комментарий: Если при попытке разблокирования расчетных токенов определяется, что данная батарея изначально не была указана для участия в замене, то выдается ошибка. Изменение баланса в контракте токенов не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

```
# scenter.py --release firmware/dca16490.py 0x9d2d7931b0edf8709260e3f7ffce74f2b3e569d6
Not confirmed.
```

Комментарий: Если при попытке разблокирования расчетных токенов определяется, что батарея с данным идентификатором и с таким же статусом уже участвовала в контракте замены, то выдается ошибка. Изменение баланса в контракте токенов не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

```
# scenter.py --release firmware/7db4ff5b.py 0xb1d1969e3932335eec67830d24268b0622ed7910
Not confirmed.
```

Комментарий: Если при попытке разблокирования расчетных токенов определяется, что временная метка, включенная в статус батареи, меньше, чем ожидаемая с учетом временного промежутка запрета разблокирования токенов, то выдается ошибка. Изменение баланса в контракте токенов не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# scenter.py --release firmware/5fa8d151.py 0xca8d895baf2bf90810dc65507d2625c509b16c22` `Not confirmed.`
Комментарий: Если при попытке разблокирования расчетных токенов определяется, что аккаунт сервисного центра, использующийся для разблокирования токенов, не указан в качестве участника контракта (в роли сервисного центра), то выдается ошибка. Изменение баланса в контракте токенов не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# scenter.py --release firmware/7db4ff5b.py 0x40d6ea55a436f814de4210d0de444fe52fef3946` `Not confirmed.`
Комментарий: Если при попытке разблокирования расчетных токенов определяется, что данная сделка была отменена (сервисным центром или электромобилем) или завершена, то выдается ошибка. Изменение баланса в контракте токенов не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# scenter.py --release firmware/4998c293.py 0xca8d895baf2bf90810dc65507d2625c509b16c22` `Not confirmed.`
Комментарий: Если при попытке разблокирования расчетных токенов определяется, что электромобиль еще не послал подтверждение на согласие со сделкой, то выдается ошибка. Изменение баланса в контракте токенов не происходит. Изменений в `Battery Management Contract` не происходит. Локальных запросов для проверки статуса батареи перед отправкой транзакции не создается.

Ethereum контракты

Одно из требований к децентрализованному приложению - предоставлять ожидаемый интерфейс взаимодейтсвия. Это нужно потому, что клиентская часть приложения может быть реализована не обязательно в виде `Pyhton` скриптов. Она может быть выполнена, как приложение для десктопа, web-приложение или мобильное приложение.

Разработка всех частей может вестись независимо друг от друга в один и тот же момент, поэтому важно заранее описать интерфейсы взаимодействия и не менять их при дальнейшей разработке. Иначе неизбежны задержки в разработке ПО, срыв сроков, дополнительные расходы и поздний вывод продукта на рынок.

Ниже представлены задания, ответственные за реализацию отдельных методов Ethereum контрактов. Каждое задание будет считаться принятым, если совпадают все указанные в нем ожидания.

Для вызова методов перечисленных ниже контрактов будет использоваться ABI, который может быть получен из описания интерфейсов контрактов, представленных ниже

EPIC-[2]EpicCounter Management contract

Описание интерфейса контракта:

```
pragma solidity ^0.4.19;

contract ManagementContractInterface {
    // Для оповещения регистрации нового производителя
    // - адрес аккаунта из-под которого проходила регистрация
    // - идентификатор производителя
    event Vendor(address, bytes4);

    // Для оповещения о создании новой батареи
    // - идентификатор производителя
    // - идентификатор батареи
    event NewBattery(bytes4, bytes20);

    // Конструктор контракта
    // - адрес контракта, ответственного за накопление криптовалюты,
    //   перечисляемой в качестве депозита за использование сервиса.
    // - сумму сбора за выпуск одной батареи
    function ManagementContractInterface(address, uint256) public {}

    // Устанавливает адрес для контракта, ответственного за
    // управление информацией о батареях.
    // Доступен только создателю management контракта
    // - адрес контракта, управляющего инфорацией о батареях
    function setBatteryManagementContract(address) public;

    // Регистрирует вендора, если при вызове метода перечисляется
    // достаточно средств.
    // - наименование производителя
    function registerVendor(bytes) public payable;

    // Возвращает размер текущий депозит вендора
    function vendorDeposit(address) public view returns(uint256);

    // Регистрирует новые батареи, если при вызове метода на балансе
    // данного производителя достаточно средств. Во время регистрации
    // батарей баланс уменьшается соответственно количеству батареи и
    // цене, установленной для данного производителя на текущий момент.
    // - идентификаторы батарей
    function registerBatteries(bytes20[]) public payable;

    // Возвращает наименование производителя по его идентификатору
    function vendorNames(bytes4) public view returns(bytes);

    // Возвращает идентификатор производителя по адресу производителя
    function vendorId(address) public view returns(bytes4);

    // Возвращает адрес контракта, управляющего информацией о батареях,
    // установленного в данный момент.
    function batteryManagement() public view returns(address);

    // Устанавливает сумму сбора за выпуск одной батареи
    // - сумма сбора в wei
    function setFee(uint256) public;

    // Возвращает сумму сбора в wei, который будет списываться с депозита
    // за каждую выпущенную батарею. Если запрос идет от зарегистрированного
    // производителя батарей, то выдается та сумма сбора, которая была на момент
    // его регистрации.
    function batteryFee() public view returns(uint256);
```

```
    // Возвращает сумму сбора в wei, которую необходимо перечислить в качестве
    // депозита при регистрации производителя батарей. Сумма депозита
    // равна сумме за регистрацию 1000 батарей, что позволит защититься от
    // мошенников, поскольку требует серьезных вложений.
    function registrationDeposit() public view returns(uint256);

    // Возвращает адрес контракта, на котором происходит накопление
    // криптовалюты.
    function walletContract() public view returns(address);

    // Возвращает истину или ложь в зависимости от того, зарегистрирован сервис
    // центра с указанным адресом в системе или нет.
    function serviceCenters(address) public view returns(bool);

    // Регистрирует в системе адрес отправителя транзакции, как сервис центр.
    // Регистрация происходит только если данный адрес уже не был зарегистрирован
    // в качестве сервис центра или электромобиля.
    function registerServiceCenter() public;

    // Возвращает истину или ложь в зависимости от того, зарегистрирован
    // электромобиль с указанным адресом в системе или нет.
    function cars(address) public view returns(bool);

    // Регистрирует в системе адрес отправителя транзакции, электромобиль.
    // Регистрация происходит только если данный адрес уже не был зарегистрирован
    // в качестве сервис центра или электромобиля.
    function registerCar() public;
}
```

US-[3]USCounter метод setBatteryManagementContract()

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если отправитель транзакции - аккаунт компании разработчика ПО.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова этого метода будет проверяется вызовом `batteryManagement()` значения должны совпадать.

US-[3]USCounter метод registerVendor()

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если сумма транзакции отличная от нуля, если производитель батарей с таким названием еще не зарегистрирован, если с аккаунта отправителя транзакции не регистрировался еще не один производитель.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Генерируется событие `Vendor` с указанием адреса отправителя транзакции и идентификатора производителя

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Проверяется вызовом `vendorId()` и `vendorNames()`, которые позволяют получить адрес производителя по его идентификатору и название производителя по его адресу. Наименование производителя должно совпадать с тем, что было передано в метод `registerVendor()` при его вызове.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Депозит отправителя транзакции в `Management Contract` становится равным сумме транзакции. Результат можно проверить `vendorDeposit()`.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Изменяется баланс счета *Servide Provider Wallet* на сумму, указанную в транзакции

US-[3]USCounter метод registerBatteries()

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если аккаунт отправитель уже зарегистрирован, как предводитель батарей, если депозит данного производителя достаточен для регистрации нужного количества батарей с учетом суммы транзакции.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
С депозита отправителя транзакции списывается сумма равная количеству батарей умноженному на сумму сбора за регистрацию одной батареи, которая была доступна на момент регистрации данного производителя. Результат можно проверить `vendorDeposit()`.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Для каждой батареи из переданного списка генерирует событие `NewBattery` с указанием идентификатора производителя и идентификатора батареи

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Каждый идентификатор батареи может провериться вызовом `ownerOf()`, `vendorOf()` в `Battery Management`, которые позволяют получить адрес владельца батареи и адрес ее производителя. Оба значения должны совпадать с адресом отправителя транзакции, вызвавшей `registerBatteries()`

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Для каждой зарегистрированной батареи можно сформировать вызовом скрипта, эмулирующего аппаратный ключ, такие данные, что `verifyBattery()` в `Battery Management` вернет 0.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Если в транзакции вызывающей метод была указана сумма транзакции, то баланс счета *Servide ProviderWallet* изменяется на эту сумму.

US-[3]USCounter метод setFee()

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если отправитель транзакции - аккаунт компании разработчика ПО.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова этого метода будет проверяется вызовом `batteryFee()` значения должны совпадать, при вызове из-под аккаунта, который не зарегистрирован в качестве производителя батареи

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
При вызове из-под аккаунта, который зарегистрирован в качестве производителя батареи, то указывается сумма сборов, которая была установлена на момент регистрации производителя

US-[3]USCounter метод registerServiceCenter()

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если отправитель транзакции не еще не зарегистрирован ни как сервисный центр, ни как элекромобиль.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова этого метода будет проверяется вызовом `serviceCenters()` значения должны совпадать.

US-[3]USCounter метод registerCar()

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если отправитель транзакции не еще не зарегистрирован ни как сервисный центр, ни как элекромобиль.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова этого метода будет проверяется вызовом `cars()` значения должны совпадать.

EPIC-[2]EpicCounter Battery Management

Описание интерфейса контракта:

```
pragma solidity ^0.4.19;

contract BatteryManagementInterface {
    // Для оповещения о передаче прав владения батареей новому владельцу
    // - адрес предыдущего владельца
    // - адрес нового владельца
    // - индентификатор батареи
    event Transfer(address indexed, address indexed, bytes20);

    // Для оповещения, что какому-то аккунту делегировано право менять владельца
    // батареи
    // - владелец батареи
    // - адрес далегата
    // - идентификатор батареи, которой может распоряжаться делегат
    event Approval(address indexed, address indexed, bytes20);

    // Для оповещения после создания контракта на замену батарей
    // - адрес контракта
    event NewDeal(address);

    // Конструктор контракта
    // - адрес контракта, управляющего списком вендоров.
    // - адрес контракта, управляющего токенами, в которых будет
    //   происходить расчет за замену батарей.
    function BatteryManagementInterface(address, address) public {}

    // Создает новую батарею
    // Владельцем батареи назначается его текущий создатель.
    // Создание нового батареи может быть доступно только
    // management контракту
    // - адрес производителя батареи
    // - идентификатор батареи
```

```
function createBattery(address, bytes20) public;

// Меняет владельца батареи. Может быть вызвана только
// текущим владельцем
// - адрес нового владельца
// - идентификатор батареи
function transfer(address, bytes20) public;

// Меняет владельца для всех батарей перечисленных в списке. Может быть вызвана
// только текущим владельцем
// - адрес нового владельца
// - список идентификаторов батарей
function transfer(address, bytes20[]) public;

// Определяет по статусу батареи ее идентификатор, затем текущего владельца.
// И меняет владельца батаереи. Может быть вызвана только текущим владельцем
// - запакованные данные, хранящие статус батареи и нового получателя. Упаковка
//   данных используется для уменьшения используемого газа за данные,
//   передаваемые в транзакции в метод контракта. Метод упаковки:
//   p = n*(2**124) + t*(2**192) + v*(2**160) + address
// - r компонента подписи для данных батареи
// - s компонента подписи для данных батареи
function transfer(bytes32, bytes32, bytes32) public;

// Делегирует право аккунту изменять владельца батареи, которой на текущий
// момент владеет аккаунт-отправитель транзакции. Может выполняться, если
// аккаунт-отправитель тразанкции действительно владеет данной батареей. Для
// одной батареи может быть только один делегат. Для отзыва права необходимо
// послать 0 в качестве адреса.
// Генерирует событие Approval.
// - адрес делегата
// - идентификатор батареи
function approve(address, bytes20) public;

// Делегирует право аккунту изменять владельца батареи, которым на текущий
// момент владеет аккаунт, адрес которого определяется в ходе проверки цифровой
// подписи. Может выполняться, если аккаунт, восстанавливаемый из цифровой
// подписи, действительно владеет данной батареей. Для одной батареи может
// быть только один делегат. Для отзыва права необходимо
// послать 0 в качестве адреса.
// Цифровая подпись проверяется относительно адреса отправителя транзакции
// (msg.sender), поскольку подразумевается, что тем самым владелец батареи
// подтверждает, что делегирование права может быть выполнено с данного адреса.
// Генерирует событие Approval.
// - адрес делегата
// - идентификатор батареи
// - v, r, s компоненты цифровой подписи
function delegatedApprove(address, bytes20, uint8, bytes32, bytes32) public;

// Меняет владельца батареи. Может выполняться, только если отправителю
// транзакций делегировано право изменения владельца для данной батареи.
// Генерирует событие Transfer.
// - адрес владельца батареи
// - адрес получателя
// - идентификатор батареи
function transferFrom(address, address, bytes20) public;

// Возвращает адрес текущего зарегистрированного владельца батареи
// - индентификатор батареи
function ownerOf(bytes20) view public returns(address);
```

```
// Возвращает адрес производителя батареи
// - идентификатор батареи
function vendorOf(bytes20) view public returns(address);

// Проверяет цифровую подпись для данных и возвращает результат
// проверки и адрес вендора.
// Результат проверки: 0 - результат проверки цифровой подписи, показывает
// что она сделана зарегистрированной батареей; 1 - транзакция
// с таким статусом уже отправлялась в блокчейн, что указывает на возможную
// прослушку траффика; 2 - нет соответствующей зарегистрированной батареи;
// 999 - другая ошибка.
// - число зарядов
// - временная метка
// - v, r, s компоненты цифровой подписи
function verifyBattery(uint256 n, uint256 t, uint8 v, bytes32 r, bytes32 s)
    public view returns(uint256, address);

// Формирует новый контракт на выполнение операций по замене батарей.
// В контракт передаются идентификаторы батарей, участвующие в сделке,
// адрес аккаунта автомобиля, сумма на компенсацию амортизации батарей,
// сумма за выполенние работ по замене.
// Идентификаторы батарей становятся известны в ходе проверки цифровых
// подписей для статусов батарей.
// Сумма компенсации исчисляется в зависимости от количества заряда,
// переданного в статусе батарей.
// Контракт не должен создаваться, если для батареи с такими
// идентфикаторами не зарегистрированы в контракте, если одна из батарей
// уже участвует в каком-то другом контракте замены или участники контракта
// не являются владельцами батарей.
// - запакованные данные, хранящие статус сразу двух батарей. Упаковка
//   данных используется для уменьшения используемого газа за данные,
//   передаваемые в транзакции в метод контракта. Метод упаковки:
//   p = n1*(2**160) + t1*(2**128) + v1*(2**96) + n2*(2**64) + t2*(2**32) + v2
// - r компонента подписи для данных старой батареи
// - s компонента подписи для данных старой батареи
// - r компонента подписи для данных новой батареи
// - s компонента подписи для данных новой батареи
// - адрес аккаунта автомобиля
// - количество расчетных токенов, определяющих выполнение работ по замене
function initiateDeal(uint256, bytes32, bytes32, bytes32, bytes32, address,
    uint256) public;

// Возвращает количество зарядов для конкретной батареи, зарегистрированных
// в данный момент
// - идентификатор батареи
function chargesNumber(bytes20) view public returns(uint256);

// Извлекает из переданных данных цифровую подпись текущего владельца батареи,
// адрес нового владельца батареи, ее статус. Изменяет владельца, если
// нет нарущений прав владения для восстановленного из подпсис адреса.
// - запакованные данные, хранящие статус батареи и нового получателя. Упаковка
//   данных используется для уменьшения используемого газа за данные,
//   передаваемые в транзакции в метод контракта. Метод упаковки (от старших
//   байт к младшим):
//    0..3   байт - число зарядов
//    4..7   байт - временная метка
//    8..75  байт - v, r, s компоненты цифровой подписи статуса батареи
//   76..95  байт - адрес нового владельца
//   96..163 байт - v, r, s компоненты цифровой подписи текущего владельца
```

```
    function approvedTransfer(bytes) public;

    // Проверяет переданный адрес, если он является контрактом сделки.
    // Возрващает 0 - контракт сделки существует и аккаунт, с которого
    // осуществляется запрос, является участником сделки; 1 - контракт сделки
    // существует, но аккаунт не является участников сделки; 2 - контракт
    // сделки не существует.
    function checkDealContract(address) view public returns(uint256);

    // Устанавливает верменной промежуток, после которого разрешено
    // разблокирование расчетных токенов контрактом сделки. Данная операция
    // разрешена только аккаунту производителя ПО.
    // - размер временного промежутка в секундах.
    function setTimeoutThreshold(uint256) public;

    // Возвращает временной промежуток, запрета разблокирования расчетных
    // токенов, установленный на текущий момент.
    function timeoutThreshold() view public returns(uint256);

    // Возвращает адрес контракта, который управляет регистрацией производителей
    // батарей.
    function managementContract() view public returns(address);

    // Возвращает адрес контракта, который управляет токенами, использующимися
    // в качестве внутренней валюты при совершении сделок по замене батарей.
    function erc20() view public returns(address);
}
```

US-[3]USCounter метод createBattery()

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если вызывается из `Management Contract` и если батареи с таким идентификатором не существует еще.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если вызывается из `Management Contract`.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова провериться вызовом `ownerOf()`, `vendorOf()`, которые позволяют получить адрес владельца батареи и адрес ее производителя. Оба значения должны совпадать с адресом, указанным в вызовае `createBattery()`

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Для зарегистрированной батареи можно сформировать вызовом скрипта, эмулирующего аппаратный ключ, такие данные, что `verifyBattery()` вернет 0.

US-[3]USCounter метод transfer() с указанием одного идентификатора батареи

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если отправитель транзакции является владельцем батареи и если батарея с таким идентификатором существует

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если получатель права на владение батареей зарегистрирован, как электромобиль или сервисная станция

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Генерируется событие `Transfer` с указанием старого и нового владельцев и идентификатора батареи

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова провериться вызовом `ownerOf()`. Адрес возвращаемый методом должен совпадать с адресом указанным в `transfer()` для того же идентификатора батареи.

US-[3]USCounter метод delegatedApprove()

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если адрес аккаунта, восстанавливаемый из цифровой подписи, указанной в параметрах, действительно владеет батарей с указанным идентификатором.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если адрес аккаунта, который вызывает данный метод, входит в то сообщение, для которого создана цифровая подпись.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Генерируется событие `Approval` с указанием владельца батареи, того кто может передавать права владения и идентификатора батареи

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова проверяется вызовом `transferFrom()` - аккаунт вызывающий данный метод, должен совпадать с аккаунтом, которому делегировалось право смены владельца.

US-[3]USCounter метод initiateDeal()

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если отправитель транзакции, является сервисным центром, и если он является владельцем одной из батарей, чей идентификатор восстанавливается из статуса

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если аккаунт, указанный в качестве параметра, является аккаунтом электромобиля и владельцем одной из батарей, чей идентификатор восстанавливается из статуса.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если статусы обеих батарей еще не использовались при отправке транзакций в блокчейн: для создания, отмены или подтверждения сделок, для разблокирования расчетных токенов и т.п.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если батареи с такими идентификаторами не участвуют ни в какой другой сделке.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Генерируется событие `NewDeal` с указанием адреса контракта сделки.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Генерируется событие `Approval` с указанием адреса сервисного центра, адреса контракта сделки и идентификатором батареи, которую сервисный центр предлагает к замене.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить, если статусы, использовавшиеся в транзакции, отправить в `verifyBattery()` - она вернет 1.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить вызовом `chargesNumber()` с идентификатором каждой из батарей. Должно вернуться тоже значение выполненных зарядов, что и в статусе, который использовался для создания сделки.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить вызовом `oldBattery()` или `newBattery()` у контракта, расположенного по адресу, указанному в событии `NewDeal`. Методы должны вернуть те же идентификаторы батарей, что получатся при восстановлении из статусов, использовавшихся при создании сделки.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить вызовом `oldBatteryInfo()` или `newBatteryInfo()` у контракта, расположенного по адресу, указанному в событии `NewDeal`. Методы должны вернуть верную информацию о производителях батарей, участвующих в сделке.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить вызовом `deprecationValue()` и `serviceFee()` у контракта, расположенного по адресу, указанному в событии `NewDeal`. Методы должны вернуть верную информацию о количестве токенов, необходимых для сделки (сумма компенсации при замене батарей должна соответствовать значению рассчитанному по Таблице 1.1.).

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить вызовом `agreeToDeal()` или `cancelDeal()` у контракта, расположенного по адресу, указанному в событии `NewDeal`. Методы должны успешно выполняться (при соблюдении остальных условий их успешного выполнения).

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить, получив новые статусы с батарей, участвовавшие в контракте, и послав их в другой вызов `initiateDeal()`, который завершится с ошибкой из-за того, что батареи уже используются в другой сделке.

US-[3]USCounter метод setTimeoutThreshold()

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если отправитель транзакции - аккаунт компании разработчика ПО.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова этого метода будет проверяется вызовом `timeoutThreshold()` значения должны совпадать.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова этого метода будет проверяется вызовом `releaseTokensByCar()` или `releaseTokensByServiceCenter()` у контрактов сделки, созданных после установки нового временного промежутка. Разблокирование токенов должно происходить только после истечения установленного промежутка времени.

EPIC-[2]EpicCounter Deal contract

Описание интерфейса контракта:

```
pragma solidity ^0.4.19;

contract DealInterface {

    // Конструктор контракта
    // - идентификатор старой батареи
    // - идентификатор новой батареи
    // - адрес контракта расчетных токенов
    // - количество токенов, определяющих компенсацию амортизации батарей
    // - количество токенов, определяющих выполнение работ по замене
    // - временной промежуток, в ходе которого запрещено разблокирование
    //   расчетных токенов.
    function Deal(bytes20, bytes20, address, uint256, uint256, uint256) public {
    }

    // Возвращают идентификаторы батарей, вовлеченных в контракт.
    function oldBattery() view public returns(bytes20);
    function newBattery() view public returns(bytes20);

    // Возвращают информацию о батареях, вовлеченных в контракт.
    // Возвращаемые значения:
    // - количество зарядов
    // - идентификатор производителя
    // - наименование производителя
    function oldBatteryInfo() view public returns(uint256, bytes4, bytes);
    function newBatteryInfo() view public returns(uint256, bytes4, bytes);

    // Возвращает количество токенов, определяющих компенсацию амортизации
    // батарей
    function deprecationValue() view public returns(uint256);

    // Возвращает количество токенов, определяющих выполнение работ по
    // замене
    function serviceFee() view public returns(uint256);

    // Возвращает текущий статус контракта
    // 1 - Ожидание согласия со сделкой от автомобиля
    // 2 - Согласие со сделкой получено
    // 3 - Оплата работ выполнена
    function state() view public returns(uint256);

    // Проверяет, что переданный статус батареи получен от батареи, которая
    // зарегистрирована к замене (старая), что данный запрос идет от текущего
```

```
// владельца батареи, который участвует в контракте в роли получателя новой
// батареи. Если все условия выполняются, то токены идущие в оплату
// контракта переводятся на счет контракта.
// - запакованные данные, хранящие статус батареи. Упаковка
//   данных используется для уменьшения используемого газа за данные,
//   передаваемые в транзакции в метод контракта. Метод упаковки:
//   p = n*(2**64) + t*(2**32) + v
// - r компонента подписи для данных батареи
// - s компонента подписи для данных батареи
function agreeToDeal(uint256, bytes32, bytes32) public;

// Проверяет, что переданный статус батареи получен от батареи, на которую
// должна происходить замена. При этом:
// 1) либо данный запрос идет от аккаунта, который участвует в контракте в роли
//    получателя новой батареи
// 2) либо запрос идет от аккаунта, который участвует в контракте в роли того,
//    кто предоставляет сервис по замене и с момента запроса согласия с
//    контрактом  прошло 24 часа
// Если все условия выполняются, то токены идущие в оплату контракта
// переводятся на счет сервисного центра, владельцы батарей изменяются.
// - запакованные данные, хранящие статус батареи. Упаковка
//   данных используется для уменьшения используемого газа за данные,
//   передаваемые в транзакции в метод контракта. Метод упаковки:
//   p = n*(2**64) + t*(2**32) + v
// - r компонента подписи для данных батареи
// - s компонента подписи для данных батареи
function confirmDeal(uint256, bytes32, bytes32) public;

// Перепосылает аргументы в контракт BatteryManagement для проверки цифровой
// подписи. Возвращаемый результат проверки такой же, как и в оригинальной
// функции: 0 - результат проверки цифровой подписи, показывает
// что она сделана батареей, для которой существует токен; 1 - транзакция
// с таким статусом уже отправлялась в блокчейн, что указывает на возможную
// прослушку траффика; 2 - для  батареи нет соответствующего токена; 999 -
// другая ошибка.
// - число зарядов
// - временная метка
// - v, r, s компоненты цифровой подписи
function verifyBattery(uint256 n, uint256 t, uint8 v, bytes32 r, bytes32 s)
    public view returns(uint256, address);

// Обрабатывает запрос на разблокирование расчетных токенов и возвращает
// токены на счет электромобиля, владельцы батарей, участвующих в сделке,
// не изменяются.
// Выполняется только если отправитель запроса, зарегистрирован в качестве
// получателя новой батареи, если проверка подписи из статуса батареи указывает
// на то, что батарея не была заменена, и если время из статуса больше,
// чем разрешенное время разблокировки токенов.
// - запакованные данные, хранящие статус батареи. Упаковка
//   данных используется для уменьшения используемого газа за данные,
//   передаваемые в транзакции в метод контракта. Метод упаковки:
//   p = n*(2**64) + t*(2**32) + v
// - r компонента подписи для данных батареи
// - s компонента подписи для данных батареи
function releaseTokensByCar(uint256, bytes32, bytes32) public;

// Обрабатывает запрос на разблокирование расчетных токенов и перечисляет
// токены на счет сервисного центра, происходит смена владельцев батарей,
// участвующих в сделке.
// Выполняется только если отправитель запроса, зарегистрирован в качестве
```

```
    // изначального владельца новой батареи, если проверка подписи из статуса
    // батареи указывает на то, что батарея действительно была заменена, и
    // если время из статуса больше, чем разрешенное время разблокировки токенов.
    // - запакованные данные, хранящие статус батареи. Упаковка
    //   данных используется для уменьшения используемого газа за данные,
    //   передаваемые в транзакции в метод контракта. Метод упаковки:
    //   p = n*(2**64) + t*(2**32) + v
    // - r компонента подписи для данных батареи
    // - s компонента подписи для данных батареи
    function releaseTokensByServiceCenter(uint256, bytes32, bytes32) public;

    // Отменяет сделку, если она получатель новой батареи, не согласился с
    // контрактом. Выполняется только если отправитель запроса - сервисный
    // центр.
    function cancelDeal() public;
}
```

US-[3]USCounter метод agreeToDeal()

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только если отправитель транзакции, является электромобилем, и если он является владельцем батареи, чей идентификатор восстанавливается из статуса, использованного в транзакции

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только если отправитель транзакции, является участником контракта.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только если отправитель транзакции имеет достаточное количество расчетных токенов на балансе.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только если идентификатор батареи восстановленный из статуса, использующегося в транзакции, выставлен в данном контракте.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только если временная метка, включенная в статус, использованный в транзакции, больше той, что была в статусе для соответствующей батареи во время создания контракта.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если статус батареи еще не использовался при отправке транзакций в блокчейн: для создания, отмены или подтверждения сделок, для разблокирования расчетных токенов и т.п.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если сделка не отменена и не завершена.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если до этого момента метод agreeToDeal() для этой сделки не исполнялся успешно.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Генерируется событие `Approval` с указанием адреса сервисного центра, адреса контракта сделки и идентификатором батареи, которую сервисный центр предлагает к замене.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Генерируется событие `Transfer` с указанием адресов электромобиля и контракта сделки, и тем же количеством токенов, что получаются в ходе сложения значений, которые выдаются `deprecationValue()` и `serviceFee()`, вызванные у данного контракта.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить, если вызов `state()` вернет 2.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить, если статус батареи, использовавшийся в транзакции, отправить в `verifyBattery()` - она вернет 1.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить вызовом `balanceOf()` для адреса контракта сделки в `Currency Token` контракте до и после транзакции с `agreeToDeal()`. Разница между значениями равна сумме сделки.

US-[3]USCounter метод confirmDeal()

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только если отправитель транзакции, является электромобилем и он является участников контракта.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только если идентификатор батареи, восстановленный из статуса, использующегося в транзакции, указывает на то, что владельцем этой батареи является сервисный центр - участник данного контракта.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только если идентификатор батареи восстановленный из статуса, использующегося в транзакции, выставлен в данном контракте.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только если временная метка, включенная в статус, использованный в транзакции, больше той, что была в статусе для соответствующей батареи во время создания контракта.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если статус батареи еще не использовался при отправке транзакций в блокчейн: для создания, отмены или подтверждения сделок, для разблокирования расчетных токенов и т.п.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если сделка не отменена и не завершена.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если до этого момента метод agreeToDeal() для этой сделки выполнялся успешно.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если до этого момента метод confirmDeal() для этой сделки не исполнялся успешно.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Генерируется три события `Transfer` • два - с указанием адресов сервисного центра, электромобиля и идентификаторами батарей; • одно - с указанием контракта сделки, контракта сервисного центра, и суммы расчетных токенов, установленных в контракте на начало сделки.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить, если статус батареи, использовавшийся в транзакции, отправить в `verifyBattery()` - она вернет 1.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить вызовом `balanceOf()` для адреса сервисного центра сделки в `Currency Token` контракте до и после транзакции с `agreeToDeal()`. Разница между значениями равна сумме сделки.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить вызовом `ownerOf()` для идентификаторов батарей в `Battery Management` контракта до и после транзакции, вызывающей `agreeToDeal()`. Должен произойти обмен владельцев.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить вызовом `eth.getCode()` с адресом контракта сделки через JSON-RPC. Возвращаемое значение должно быть пустым.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить, получив новые статусы с батарей, участвовавших в контракте, и послав их в другой вызов `initiateDeal()`, который должен создать новую сделку (при соблюдении остальных условий их успешного выполнения).

US-[3]USCounter метод releaseTokensByCar()

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только если отправитель транзакции, является электромобилем, и если он является владельцем батареи, чей идентификатор восстанавливается из статуса, использованного в транзакции

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только если отправитель транзакции, является участником контракта.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только если идентификатор батареи восстановленный из статуса, использующегося в транзакции, выставлен в данном контракте.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только если временная метка, включенная в статус, использованный в транзакции, больше той, что была в статусе для соответствующей батареи во время создания контракта, и больше, чем временной промежуток в ходе которого запрещено разблокирование батарей.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если статус батареи еще не использовался при отправке транзакций в блокчейн: для создания, отмены или подтверждения сделок, для разблокирования расчетных токенов и т.п.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если сделка не отменена и не завершена.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если до этого момента метод agreeToDeal() для этой сделки исполнялся успешно.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если до этого момента методы `confirmDeal()` и `cancelDeal` для этой сделки не исполнялся успешно.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Генерируется событие `Transfer` с указанием адресов контракта сделки и электромобиля, и тем же количеством токенов, что получаются в ходе сложения значений, которые выдаются `deprecationValue()` и `serviceFee()`, вызванные у данного контракта.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить, если статус батареи, использовавшийся в транзакции, отправить в `verifyBattery()` - она вернет 1.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить вызовом `balanceOf()` для адреса электромобиля в `Currency Token` контракте до и после транзакции с `releaseTokensByCar()`. Разница между значениями равна сумме сделки.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить вызовом `eth.getCode()` с адресом контракта сделки через JSON-RPC. Возвращаемое значение должно быть пустым.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить, получив новые статусы с батарей, участвовавших в контракте, и послав их в другой вызов `initiateDeal()`, который должен создать новую сделку (при соблюдении остальных условий их успешного выполнения).

US-[3]USCounter метод releaseTokensByServiceCenter()

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только если отправитель транзакции, является сервисным центром и он является участников контракта.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только если идентификатор батареи, восстановленный из статуса, использующегося в транзакции, указывает на то, что владельцем этой батареи является электромобиль - участник данного контракта.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только если идентификатор батареи восстановленный из статуса, использующегося в транзакции, выставлен в данном контракте.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только если временная метка, включенная в статус, использованный в транзакции, больше той, что была в статусе для соответствующей батареи во время создания контракта, и больше, чем временной промежуток в ходе которого запрещено разблокирование батарей.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если статус батареи еще не использовался при отправке транзакций в блокчейн: для создания, отмены или подтверждения сделок, для разблокирования расчетных токенов и т.п.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если сделка не отменена и не завершена.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если до этого момента метод agreeToDeal() для этой сделки выполнялся успешно.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если до этого момента метод confirmDeal() для этой сделки не исполнялся успешно.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Генерируется три события `Transfer` • два - с указанием адресов сервисного центра, электромобиля и идентификаторами батарей; • одно - с указанием контракта сделки, контракта сервисного центра, и суммы расчетных токенов, установленных в контракте на начало сделки.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить, если статус батареи, использовавшийся в транзакции, отправить в `verifyBattery()` - она вернет 1.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить вызовом `balanceOf()` для адреса сервисного центра сделки в `Currency Token` контракте до и после транзакции с `releaseTokensByServiceCenter()`. Разница между значениями равна сумме сделки.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить вызовом `ownerOf()` для идентификаторов батарей в `Battery Management` контракта до и после транзакции, вызывающей `releaseTokensByServiceCenter()`. Должен произойти обмен владельцев.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить вызовом `eth.getCode()` с адресом контракта сделки через JSON-RPC. Возвращаемое значение должно быть пустым.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить, получив новые статусы с батарей, участвовавших в контракте, и послав их в другой вызов `initiateDeal()`, который должен создать новую сделку (при соблюдении остальных условий их успешного выполнения).

US-[3]USCounter метод cancelDeal()

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только если отправитель транзакции, является сервисным центром и он является участников контракта.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если сделка не отменена и не завершена.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если до этого момента метод agreeToDeal() для этой сделки выполнялся успешно.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Выполняется только, если до этого момента метод confirmDeal() для этой сделки не исполнялся успешно.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить вызовом `eth.getCode()` с адресом контракта сделки через JSON-RPC. Возвращаемое значение должно быть пустым.

Ожидание к выполнению метода (AC-[3]USCounter-[2]ACCounter)
Успешность вызова можно проверить, получив новые статусы с батарей, участвовавших в контракте, и послав их в другой вызов `initiateDeal()`, который должен создать новую сделку (при соблюдении остальных условий их успешного выполнения).

Обмен батарей без участия сервисного центра

Для того, чтобы не ограничивать возможные сценарии обмена батареями только между сервисным центром и электромобилем, необходимо обеспечивать передачу батареи от одного владельца к другому (например, сценарий когда автовладелец продет батарею другому владельцу).

При этом должен поддерживаться вариант, когда подключение к блокчейн сети доступно и сторона, получающая батарею, может убедиться в передаче прав владения, так и вариант, когда подключение к блокчейн невозможно, т.е. передача прав владения может быть отложенная во времени.

В во втором случае, данная сделка не является рекомендованной, поскольку несёт очень большие риски, связанные с мошенничеством. Возможен такой вариант, когда предыдущий владелец создает фальшивую батарею и программирует ее на выдачу нескольких статусов, считанных с оригинальной батареи. Т.е. статусы с фальшивой батареи будут выглядеть так, как будто они подписаны реальной цифровой подписью и выдаются зарегистрированной в блокчейн батереей.

US-[3]USCounter Передача прав на батарею новому владельцу

Данный сценарий используется при доступе к блокчейн во время передачи прав владения. При этом новый владелец может убедиться, в том что передаваемая батарея оригинальная через получение и проверки статуса батареи.

Использование скрипта
`car.py owner <path to battery> <new_owner>`
Обращается к файлу с прошивкой батарей и получает информацию о статусе батарей. Подключается к узлу Ethereum и отправляет запрос к `Battery Management Contract` на смену владельца батареи. В терминал выводится статус выполнения команды.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py owner firmware/00360d2b.py 0x50dfbef376d01e05e134a9b3a322afef50111ec1` `Success`
Комментарий: В блокчейн сеть отправляется транзакция на изменение владельца батареи, транзакция успешно включена в блок, `Battery Management Contract` можно проинспектировать на предмет изменения владельца батареи.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py owner firmware/d819ba81.py 0x0dcd2f752394c41875e259e00bb44fd505297caf` `Failed. Not allowed to change ownership.`
Комментарий: При попытке смены владельца у батареи выдается ошибка, поскольку данная батарея не принадлежит тому, кто пытается сменить владельца. В блокчейн сеть отправляется транзакция на изменение владельца батареи, но не изменяет его состояние.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py owner firmware/d819ba81.py 0x0dcd2f752394c41875e259e00bb44fd505297caf` `Failed. Not allowed to change ownership.`
Комментарий: При попытке смены владельца у батареи может выдается ошибка, поскольку данная батарея не зарегистрирована в блокчейн. В блокчейн сеть отправляется транзакция на изменение владельца батареи, но не изменяет его состояние.

EPIC-[2]EpicCounter Замена батарей при отсутствии подключения к сети блокчейн

US-[3]USCounter Отложенная передача прав владения

Использование скрипта
`car.py offline <path to battery> <new_owner>`
Обращается к файлу с прошивкой батарей и получает информацию о статусе батарей. Подписывает приватным ключем текущего владельца батареи статус (включая цифровую подпись) и адрес нового владельца. В терминал выводится в шестнадцатеричном формате доверенность, включающая статус батареи, адрес нового владельца, подпись текущего владельца.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py offline firmware/00360d2b.py 0x50dfbef376d01e05e134a9b3a322afef50111ec1` `0c164f......8e51bc`
Комментарий: На экран выведена доверенность - строка из шестнадцатеричных символов, которая потом может использоваться для отложенного подтверждения владения.

US-[3]USCounter Регистрация прав владения

Использование скрипта
`car.py register <certificate>`
Подключается к узлу Ethereum и отправляет данные доверенности к `Battery Management Contract` на передачу прав владения адресу, указанному в доверенности. В терминал выводится статус выполнения команды.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py register 0c164f......8e51bc` `Success`
Комментарий: В блокчейн сеть отправляется транзакция на изменение владельца батареи, транзакция успешно включена в блок, `Battery Management Contract` можно проинспектировать на предмет изменения владельца батареи.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py register 0c164f......8e51bc` `Failed. Not allowed to change ownership.`
Комментарий: При попытке смены владельца у батареи может выдается ошибка, поскольку данная батарея не принадлежит тому, кто пытается сменить владельца. В блокчейн сеть отправляется транзакция на изменение владельца батареи, но не изменяет его состояние.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)
`# car.py register 0c164f......8e51bc` `Failed. Not allowed to change ownership.`
Комментарий: При попытке смены владельца у батареи может выдается ошибка, поскольку данная батарея не зарегистрирована в блокчейн. В блокчейн сеть отправляется транзакция на изменение владельца батареи, но не изменяет его состояние.

Оптимизация системы

Поскольку вычислительные мощности, затрачиваемые валидаторами блоков, на включение транзакций блок так или иначе оплачивают пользователи системы, то, очевидно, что решение, предоставляемое командой разработчика ПО должно быть конкурентным не только по функционалу, но и по стоимости пользования.

US-[3]USCounter Оптимизация по использованию вычислительных ресурсов

Чем оптимальнее логика работы контрактов, тем меньше будут платить пользователи за вызов того или иного функционала децентрализованного приложения.

Поэтому необходимо реализовывать методы контрактов таким образом, чтобы потреблять минимально возможное количества газа.

Для проверки решения данной части задания, будет проверяться, что методы, указанные ниже в таблице потребляют не больше указанного количества газа. При этом каждый метод засчитывается отдельно, т.е. допускается частичное выполнения задания. Метод будет проверяться только если успешно проверено задание (или

конкретная часть его), указанное в последнем столбце.

AC	method	gas, max	depends on
AC-[3]USCounter-01	`Management Contract:registerVendor`	—	
AC-[3]USCounter-02	`Battery Management:createBattery`	—	
AC-[3]USCounter-03	`Battery Management:transfer(address, bytes20)`	—	
AC-[3]USCounter-04	`Battery Management:transfer(bytes32, bytes32, bytes32)`	—	
AC-[3]USCounter-05	`Battery Management:delegatedApprove`	—	
AC-[3]USCounter-07	`Deal:agreeToDeal`	—	
AC-[3]USCounter-08	`Deal:confirmDeal`	—	
AC-[3]USCounter-09	`Deal:cancelDeal`	—	
AC-[3]USCounter-10	`Deal:releaseTokensByCar`	—	
AC-[3]USCounter-11	`Deal:releaseTokensByServiceCenter`	—	

US-[3]USCounter Оптимизация комиссии за валидацию транзакций

Скорость валидации транзакций и включения в блок влияет на удобство пользования системой (время присутствия электромобиля на сервисном центре для замены батареи должно быть сильно меньше, чем при зарядке этой батареи). Высокое вознаграждение валидатору транзакции, естественно ускорит включению ее в блок. Но с другой стороны, высокая стоимость оттолкнет пользователей системы. Следовательно, нужно предлагать пользователю использовать такой размер вознаграждения, который был бы оптимальным по скорости проведения транзакций и не приводил бы к явной переплате.

Для получения оптимальной стоимости газа предлагается использовать оракул `https://github.com/poanetwork/gasprice` доступный по адресу `https://gasprice.poa.network/`.

Для проверки решения данной части задания, будет проверяться, для транзакций, изменяющих состояние блокчейн, скрипты `setup.py`, `vendor.py`, `scenter.py` и `car.py` выставляет такую стоимость газа, которая по мнению оракула приводит к вероятности в 90 процентов включения транзакции в следующий блок. Задание считается выполненным, если каждый скрипт выставляет стоимость газа согласно требованиям выше.

Анализ использования системы

Поскольку одним из инструментов привлечения новых участников системы является демонстрация ее популярности, то вместе с децентрализованным приложением должен предоставляться инструмент сбора статистики.

Имя скрипта
`stat.py [command]`

US-[3]USCounter Сбор статистики

Использование скрипта

`stat.py [--tokens]`

Подключается к узлу Ethereum и, по адресу `Management Contract` из `database.json` определяет первый блок, в котором была регистрация контракта, после чего блокчейн исследуется на предмет нужных данных. В терминал выводится статистика по использованию системы.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

```
# stat.py
Registered vendors: aaa
Registered batteries: bbb
Vendor and batteries income: ccc ether
Registered deals: ddd
Batteries transfers: eee
Canceled deals: fff
```

Комментарий: Собрана статистика по использованию системы, где

- `Registered vendors` - количество зарегистрированных вендоров;
- `Registered batteries` - общее количество зарегистрированных батарей;
- `Vendor registration income` - средства полученные от регистрации вендоров и принятые в качестве дополнительной оплаты за регистрацию батарей;
- `Registered deals` - общее количество зарегистрированных сделок;
- `Batteries transfers` - общее количество операций по изменению владельца батареи;
- `Canceled deals` - общее количество сделок, которые были отменены и не привели к оплате за заменую батарей.

Пример вызова команды (AC-[3]USCounter-[2]ACCounter)

```
# stat.py --tokens
Registered vendors: aaa
Registered batteries: bbb
Vendor and batteries income: ccc ether
Registered deals: ddd
Batteries transfers: eee
Canceled deals: fff
Token flows:
* XXX tokens used in deals
* YYY tokens paid for service
* ZZZ tokens locked on deal contracts
```

Комментарий: Собрана статистика по использованию системы, где

- `tokens used in deals` - количество расчетных токенов, которые были пересланы на счета сделок;
- `tokens paid for service` - количество расчетных токенов, переведенных за совершенные замены батарей;
- `tokens locked on deal contracts` - количество расчетных токенов, заблокированных в текущий момент на незавершенных сделках.