Санкт-Петербургский государственный университет

Программная инженерия

Кафедра системного программирования

Группа 20Б.11-мм

Локтев Сергей Сергеевич

Плагин для поддержки языка Vyper в IntelliJ Platform

Отчёт по учебной практике

Научный руководитель:
к.ф.-м.н., доцент кафедры системного программирования СПбГУ Березун Д. А.

Консультант:
Мишин Н.М.

Санкт-Петербург
2022

Оглавление

Введение **3**

1. Постановка задачи **5**

2. Обзор **6**

2.1. Существующее решение 6

2.1.1. Поддержка синтаксиса 6

2.1.2. Компиляция 6

2.1.3. Исполнение кода 7

2.1.4. Статический анализ 7

2.2. Плагин для поддержки языка Solidity 7

2.3. Используемые технологии и инструменты 8

3. Описание решения **9**

3.1. Обновление зависимостей 9

3.2. Использование другой библиотеки Docker-клиента 9

3.3. Измнение функциональности, более не поддерживаемой в Intellij Platform SDK 10

3.4. Исправление ошибок в программном описании грамматики языка Vyper 10

3.5. Автодополнение кода 11

4. Результаты **13**

Список литературы **14**

Введение

В современном мире большое распространение получает технология *блокчейн*. Под ней понимают цепочку блоков, каждый из которых хранит в себе хеш-значение предыдущего блока, полученное с помощью криптографиеческой хеш-функции [1], и время, в которое был создан блок [5]. К тому же, в блоках находятся данные, необходимые для конкретной реализации данной технологии. Таким образом, для успешной попытки подмены хранящихся в нем данных, требуется пройти от выбранного для атаки блока по цепочки из хеш-значений других блоков до самого первого, инициализирующего блока, и заменить их все. Устройство предложенной структуры данных делает такой подход крайне ресурсозатратным, ввиду чего блокчейн лёг в основу разработки систем совершения транзакций, названных криптовалютами.

Первой и самой известной[1] криптовалютой стал Bitcoin, разработанный в 2008 году разработчиком под псевдонимом Сатоши Накомото [6]. Данная система позволяла совершать транзакции с "биткоинами", а информация о валидных транзакциях вышеописанным способом сохранялась в блокчейн. Ввиду таких особенностей, как анонимность, безопасность и открытось информации о транзакциях, Bitcoin начал постепенно набирать популярность как децентрализованное платежное средство, а по состоянию на ноябрь 2021 года, его пиковая стоимость составляла 65.300 долларов за единицу криптовалюты[2].

Следствием роста популярности криптовалют стала необходимость в создании технологии, позволяющей эффективно расширять область приложения блокчейна. Это привело к реализации такого инструмента, как *смарт-контракты*. Под смарт-контрактом понимают программу, которая совершает действия, опираясь на некий набор соглашений по взаимодействию, протокол.

В рамках криптовалюты были разработаны языки для написания смарт-контрактов, взаимодействующих с конкретной плафтормой и за-

[1]Статистика: https://www.blockchain.com/ru/charts (дата обращения: 14.12.2021)

[2]Источник: https://www.coindesk.com/price/bitcoin/ (дата обращения: 23.05.2022)

пускающихся на определенной виртуальной машине. Так, например, для криптовалюты Ethereum [3] существуют языки Solidity, Vyper, которые запускаются на Ethereum Virtual Machine (EVM). В рамках данной учебной практики отдельно стоит упомянуть Vyper, который появился после Solidity, но при реализован более безопасным. Так, например, язык Vyper обладает строгой типизацией, встроенными проверками на переполнение и выход за границы массивов, а также позволяет оценить "Gas Limit"[3] контракта. Для удобной разработки на любом языке программирования требуется среда, которая снабжена инструментами, ускоряющими и упрощающими действия разработчика, пишушего код.

Важной задачей является создание и поддержка сред разработки (IDE) для языков смарт-контрактов, так как ввиду их молодости, набор уже имеющихся решений не так велик, как у более старых и популярных языков. К тому же, усовершенствование инструментария позволит привлечь новых разработчиков в среду создания смарт-контрактов под технологию блокчейн, что положительно скажется на качестве появляющихся решений и, соотвественно, на качестве опыта пользования криптовалютами.

[3]Ссылка на официальный источник: https://ethgas.io/ (дата обращения: 16.12.2021)

1. Постановка задачи

Целью данной учебной практики является доработка существующего плагина для поддержки языка смарт-контрактов Vyper[4] в средах разработки, основанных на Intellij Platform[5]. За основу был взят плагин, созданный в результате работы Тюрина А.В. "Комплексная поддержка синтаксиса языка Vyper в IntelliJ Platform"[6].

Для достижения данной цели были поставлены следующие задачи:

- изучить имеющуюся реализацию в целях восстановления работоспобности решения
- исправить приоритетные ошибки в работе плагина
- расширить функциональность автодополнением кода (*autocomplete*), предоставляемыми Intellij Platform SDK

[4]Документация/Основная страница языка: https://vyper.readthedocs.io/en/stable/ (дата обращения: 16.12.2021)

[5]Основная страница: https://www.jetbrains.com/ru-ru/opensource/idea/ (дата обращения: 16.12.2021)

[6]Ссылка на работу: https://oops.math.spbu.ru/SE/YearlyProjects/spring-2019/371/Tyurin-report.pdf (дата обращения: 14.12.2021)

2. Обзор

В данном разделе представлен обзор существующего решения задачи о поддержки языка Vyper в Intellij Platform. Также обоснован выбор инструментов для выполнения поставленных задач.

2.1. Существующее решение

2.1.1. Поддержка синтаксиса

Основной задачей при создании инструмента работы с языком в среде разработки является поддержка синтаксиса данного языка. Это позволяет реализовать:

- подсветку кода
- выявление грамматических ошибок при написании кода
- навигацию по коду
- автодополнение кода

Для реализации поддеркжи синтаксиса была описана грамматика языка Vyper с помощью инструмента Grammar-Kit[7]. Данный инструмент позволяет описать синтаксис языка в форме Бэкуса-Наура [4], а также сгенерировать парсер и лексер на основе указанной формы.

2.1.2. Компиляция

В рамках имеющегося решения существует возможность компиляции написанного на языке Vyper кода. Это реализовано при помощи отправки кода в образ компилятора[8] Vyper, запускаемого в Docker-контейнере [2]. В качестве результата возвращается байт-код в текстовом виде, который разработчик может самостоятельно запустить.

[7]Репозиторий инструмента: https://github.com/JetBrains/Grammar-Kit (дата обращения: 16.12.2021)
[8]Образ: https://hub.docker.com/r/murmulla/vyper_and_vyper_run (дата обращения: 16.12.2021)

2.1.3. Исполнение кода

Помимо компиляции, упомянутый образ также способен исполнять переданный ему код. Имеющееся решение позволяет запускать контракты напрямую из среды разработки и отображать результаты работы контракта в соотвествующем окне IDE.

2.1.4. Статический анализ

Инструменты статического анализа кода позволяют выявлять ошибки, нахождение которых выходит за рамки работы синтаксического анализатора. В рамках имеющегося решения существует поддержка статического анализатора SmartCheck[9], который создан для работы с EVM. С помощью соответствующего Docker-образа[10] имеется возможность запустить анализатор из контекстного меню IDE и получить результат анализа в текстовом виде, который также отображается в окне среды разработки.

2.2. Плагин для поддержки языка Solidity

Язык **Solidity**[11] является предшественником языка Vyper и служит для создания смарт-контрактов на базе платформы Ethereum. В рамках Intellij Platform существует плагин для поддержки языка Solidity[12], обладающий схожей функциональностью с объектом данной учебной практики. Помимо этого, плагин языка Solidity имеет возможность загружать написанные смарт-контракты напрямую в блокчейн Ethereum, а также создавать прототип смарт-контракта (*file template*). Исходный код плагина использовался для нахождения решений определенных задач, а также для выявления возможных нововведений и последующих улучшений плагина языка Vyper.

[9]Репозиторий инструмента: https://github.com/smartdec/smartcheck (дата обращения: 16.12.2021)
[10]Образ: https://hub.docker.com/r/murmulla/smartcheck (дата обращения: 16.12.2021)
[11]Основная страница языка: https://docs.soliditylang.org/en/v0.8.14/ (дата обращения: 19.05.2022)
[12]https://plugins.jetbrains.com/plugin/9475-solidity (дата обращения 19.05.2022)

2.3. Используемые технологии и инструменты

Kotlin был выбран в качестве языка программирования ввиду того, что имеющееся решение было написано на нём.

Intellij Platform является основой для большого числа популярных сред разработки (более десяти миллионов пользователей по данным JetBrains на 2020 год[13]). Данная платформа предоставляет SDK для расширения возможностей IDE в виде плагинов. В частности, упомянутая SDK предоставляет API для реализации комплексной поддержки языка программирования.

Docker[14] — технология, созданная для автоматизации развертывания приложений, используется для упрощения реализации компиляции, исполнения и статического анализа кода в рамках среды разработки. Достаточно программно запустить нужный Docker-образ и отправить ему требуемые данные.

[13]По данным https://www.jetbrains.com/ru-ru/lp/annualreport-2020/ (дата обращения: 16.12.2021)

[14]Ссылка на официальную страницу инструмента: https://www.docker.com/ (дата обращения: 16.12.2021)

3. Описание решения

В данном разделе представлено описание восстановления работоспособности и расширение функциональности плагина для поддержки языка Vyper в рамках Intellij Platform. Код доступен в репозитории[15].

3.1. Обновление зависимостей

Ввиду того, что имеющееся решение было представлено в 2019 году, многие зависимоти, которые требуются для сборки проекта системой Gradle[16], получили множество обновлений. Таким образом, все зависимости были переведены на самые новые версии при наличии таковых. Отдельно стоит отметить переход с Java 8 на Java 11, так как этого требует современная версия Intellij Platform SDK. К тому же, исходя из сообщений пользователей[17] плагина, оставленных в репозитории, имеющееся решение не запускалось на более новых версиях IDE. В следствие этого была изменена версия среды разработки на более новую, под которую собирается плагин. Также была указана минимальная версия Intellij IDE, с которой работает плагин. Это сделано для корректного отображения ошибки совместимости у пользователей более старых версий среды разработки.

В процессе обновления зависимостей была выявлена устаревшая библиотека Docker-клиента, которую требовалось заменить.

3.2. Использование другой библиотеки Docker-клиента

Библиотека[18], которая использовалась для программной работы с Docker-клиентом в рамках имеющегося решения, более не поддерживается (об этом сказано на её странице на сервисе GitHub). Вследствие

[15]https://github.com/NikitaMishin/vyper-plugin (дата обращения: 14.12.2021)

[16]Основная страница инструмента: https://gradle.org/ (дата обращения: 16.12.2021)

[17]Ссылка на обсуждение: https://github.com/NikitaMishin/vyper-plugin/issues/4 (дата обращения: 14.12.2021)

[18]Репозиторий: https://github.com/spotify/docker-client (дата обращения: 16.12.2021)

этого было решено использовать другую библиотеку[19], которая более распространена, исходя из количества отметок "Избранное" у репозитория, а также до сих пор поддерживается. Для совершения перехода потребовалось полностью переписать код, отвечающий за работу с Docker-клиентом, ввиду отличий в предоставляемых API у двух упомянутых библиотек. Тем не менее, требуемая функциональность была сохранена в полной мере.

3.3. Измнение функциональности, более не поддерживаемой в Intellij Platform SDK

Как уже было отмечено, имеющееся решение было создано в 2019 году. За прошедшее время Intellij Platform SDK перетерпело множество обновлений. В частности, в одной из версий некоторые элементы API были помечены как более не поддерживамые. Эти элементы будут полностью убраны в одном из будущих обновлений SDK. Ввиду этого, такие элементы API, как:

- графические оповещения IDE
- действие IDE по созданию файла с кодом поддерживаемого языка
- интерактивные графические элементы для запуска кода

Были заменены на поддерживаемые аналоги. С учетом этого адаптирован код, использующий данные элементы API.

3.4. Исправление ошибок в программном описании грамматики языка Vyper

В процессе работы над плагином были выявлены ошибки в работе парсера — они были связаны с частично некорректным описанием грамматики языка с помощью инструмента Grammar-Kit. К тому же,

[19]Репозиторий: https://github.com/docker-java/docker-java (дата обращения: 16.12.2021)

ввиду прошедшего времени с создания изначальной реализации плагина, язык Vyper претерпел изменения, которые требовалось учесть. Так, язык пополнился "декораторами", которые, например, влияют на видимость функций. Найденные ошибки, а также недостающие элементы языка из более новых версий были добавлены в описание грамматики и протестированы с помощью примеров из официального репозитория GitHub языка Vyper[20].

3.5. Автодополнение кода

Полезным инструментом при написании кода является автодополнение, позволяющее разработчику не дописывать до конца ключевые слова или известные переменные, функции и тому подобное. В рамках платформы Intellij существует два типа дополения: статическое и динамическое. Суть обоих типов заключается в создании списка подходящих элементов, которые будут предложены разработчику в качестве конечного вида "слова", которое он пишет в данный момент.

Статическое дополнение применяется, когда множество возможных элементов не зависит от содержания программы. С помощью такого типа можно дополнять ключевые слова языка, например "def" (объявление функции) и "struct" (объявление пользовательских структур). Тем не менее, в список предложений не будут входить все ключевые слова языка. Для определения нужного подмножества используется не только синтаксический анализ (совпадает ли имеющийся элемент с каким-либо префиксом элемента множества), но и расположение элемента, который пишет разработчик, в дереве разбора. Так, ключевое слово "def" может находиться только на верхнем уровне. Intellij Platform предоставляет возможность создавать "паттерны" — достаточно описать с помощью встроенных отношений (родитель, сын, "лист", внутри файла и тому подобное) последовательность элементов дерева разбора, при котором будет предложено определенное подмножество ключевых слов.

[20]https://github.com/vyperlang/vyper/tree/48e326f0685723e24c9d8daa8d13a4011099483b/examples (дата обращения: 18.05.2022)

С помощью интерфейса *Completion Contributor*[21] реализовано статическое дополнение. Для его использования достаточно указать паттерн и список элементов, которые описаны выше.

Динамическое дополнение, напротив, применяется когда множество возможных элементов зависит от содержания программы. В данную категорию входят имена переменных, функций, пользовательстких структур и тому подобное. В момент поиска подходящих элементов для подстановки запускается механизм обхода дерева разбора, путь которого полностью указывается разработчиком плагина, который позволяет найти объекты, удовлетворяющие требуемым условиям. После этого объекты собираются в множество предложений пользователю для подстановки. Данная функциональность реализована с помощью интерфейса *PsiPolyVariantReferenceBase*, одна из функций которого отвечает за поиск нужного элемента в дереве разбора.

[21] Документация: https://plugins.jetbrains.com/docs/intellij/completion-contributor.htmldefine-a-completion-contributor (дата обращения: 23.05.2022)

4. Результаты

В ходе выполнения данной работы были достигнуты следующие результаты:

- изучена имеющаяся реализация плагина для поддержки языка Vyper под Intellij Platform, а также похожий плагин для языка Solidity
- восстановлена работоспособность плагина
- добавлена поддержка автодополнения кода

Код доступен в репозитории[22], размещённом на сервисе GitHub.

[22]https://github.com/NikitaMishin/vyper-plugin (дата обращения: 14.12.2021)

Список литературы

[1] Al-Kuwari Saif, Davenport James H., and Bradford Russell J. Cryptographic Hash Functions: Recent Design Trends and Security Notions. — Cryptology ePrint Archive, Report 2011/565. — 2011. — https://ia.cr/2011/565.

[2] Anderson Charles. Docker [software engineering] // Ieee Software. — 2015. — Vol. 32, no. 3. — P. 102–c3.

[3] Buterin Vitalik et al. Ethereum white paper: a next generation smart contract & decentralized application platform // First version. — 2014. — Vol. 53.

[4] Deremer Franklin L. Generating parsers for BNF grammars // Proceedings of the May 14-16, 1969, spring joint computer conference. — 1969. — P. 793–799.

[5] Haber Stuart and Stornetta W Scott. How to time-stamp a digital document // Conference on the Theory and Application of Cryptography / Springer. — 1990. — P. 437–455.

[6] Nakamoto Satoshi. Bitcoin: A peer-to-peer electronic cash system // Decentralized Business Review. — 2008. — P. 21260.