МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ
ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет информационных технологий
Кафедра компьютерных технологий

Направление подготовки 09.04.01 Информатика и вычислительная техника
Направленность (профиль): Технология разработки программных систем

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА МАГИСТРА

Витченко Владислава Алексеевича

Тема работы:

УНИФИЦИРОВАННАЯ АРХИТЕКТУРА МОДУЛЯ РАСШИРЕНИЯ ЯДРА WEB-IDE ПРОЦЕСС-ОРИЕНТИРОВАННОГО ЯЗЫКА REFLEX

«К защите допущена»
Зав. кафедрой КТ ФИТ НГУ,
д. т. н., доцент,

Зюбин В.Е./……..
(ФИО) / (подпись)
«31» мая 2022г.

Руководитель ВКР
Зав. кафедрой КТ ФИТ НГУ,
д. т. н., доцент

Зюбин В.Е./……..
(ФИО) / (подпись)
«31» мая 2022г.

Соруководитель ВКР
к. т. н, доцент каф. ОИ ФИТ НГУ

Лях Т.В. /……
(ФИ О) / (подпись)
«31» мая 2022г.

Новосибирск, 2022

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«НОВОСИБИРСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»
(НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ, НГУ)

Факультет информационных технологий

Кафедра компьютерных технологий
(название кафедры)

Направление подготовки: 09.04.01 ИНФОРМАТИКА И ВЫЧИСЛИТЕЛЬНАЯ ТЕХНИКА
Направленность (профиль): Технология разработки программных систем

УТВЕРЖДАЮ

Зав. кафедрой Зюбин В.Е.
(фамилия, И., О.)

……
(подпись)
«17» декабря 2020 г.

ЗАДАНИЕ
НА ВЫПУСКНУЮ КВАЛИФИКАЦИОННУЮ РАБОТУ МАГИСТРА

Студент*у(ке)* Витченко Владиславу Алексеевичу, группы 20222
(фамилия, имя, отчество, номер группы)

Тема Унифицированная архитектура модуля расширения ядра Web-IDE процесс-ориентированного языка Reflex
(полное название темы выпускной квалификационной работы магистра)

утверждена распоряжением проректора по учебной работе от «17» декабря 2020 г. №0446
Срок сдачи студентом готовой работы «31» мая 2022 г.
Исходные данные (или цель работы): исследование способов расширения WEB-IDE процесс-ориентированного языка Reflex
Структурные части работы: исследовательская (текст ВКР), программный модуль.

Руководитель ВКР
Зав. кафедрой КТ ФИТ НГУ,
д. т. н., доцент,
Зюбин В.Е./……..
(ФИ О) / (подпись)
«20» декабря 2020 г.

Задание принял к исполнению
Витченко В.А/……..
(ФИО студента) / (подпись)
«20» декабря 2020 г.

Соруководитель ВКР
к. т. н, доцент каф. ОИ ФИТ НГУ

Лях Т.В. /……
(ФИ О) / (подпись)
«20» декабря 2020 г.

ОГЛАВЛЕНИЕ

СПИСОК СОКРАЩЕНИЙ И ОПРЕДЕЛЕНИЙ

АУ — Алгоритм Управления;

КФС — Кибер-физические системы;

IDE — Integrated Development Environment;

ПО — Программное Обеспечение;

API — Application Programming Interface;

DSM — Domain Specific Modules;

AST — Abstract Syntax Tree.

1. ВВЕДЕНИЕ

В современном мире существует потребность в использовании “кибер-физических систем” — информационно-технологических комплексов, подразумевающих воздействие компьютерных программ на реальную физическую среду [1].

Алгоритмы, исполняемые в рамках кибер-физических систем, называют алгоритмами управления. Возможность воздействия программного обеспечения, реализующего алгоритмы управления, на физическую среду накладывает серьезные риски. Для минимизации и устранения подобных рисков к реализации алгоритмов управления применяются особые требования по валидности результатов исполнения.

Для соблюдения данных требований привычных инструментов разработки оказывается недостаточно. Поэтому ведутся исследования новых подходов и разработка специализированных средств, позволяющих упростить соблюдение требований к качеству разработанных решений [2].

Таким образом, появляются специализированные инструменты для реализации алгоритмов управления, среди них присутствуют решения Института автоматики и электрометрии: процесс-ориентированная парадигма программирования, процесс-ориентированный язык Reflex, процесс-ориентированный язык POST, комплекс динамической верификации программ на языке Reflex, интегрированная среда разработки для языка Reflex — RIDE 2.0 [3][4].

RIDE 2.0 — является клиент-серверным WEB-приложением. И на текущий момент предоставляет функциональность редактирования и компиляции программ на языке Reflex. Реализована подсветка синтаксиса и отображение ошибок компиляции непосредственно в самом редакторе, а не в терминале [5].

Так как только редактора кода недостаточно при реализации алгоритмов управления, в среду необходимо интегрировать специализированные средства такие как: инструменты реверсного инжиниринга, отладчики, верификаторы, редакторы диаграмм.

Для предоставления единообразного способа добавления подобных инструментов в среду разработки, необходим описанный механизм разработки и интеграции расширений.

В связи с этим, целью работы стало исследование способов расширения RIDE 2.0 — WEB-IDE процесс-ориентированного языка Reflex.

Были поставлены следующие задачи:

1. Проанализировать специфику RIDE 2.0.
2. Сформулировать требования к механизму расширения.
3. Описать процедуру разработки и интеграции новых расширений в среду разработки.
4. Реализовать программные средства, упрощающие разработку расширения.

Работа разбита на три главы: “Анализ”, “Проектирование”, “Реализация”. В главе “Анализ” приводятся результаты исследования специфики RIDE 2.0 и требования к необходимому механизму расширения. В главе “Проектирование” описывается структура. В главе “Реализация” описаны детали реализации программных средств, упрощающих разработку расширений.

2. АНАЛИЗ

1. Eclipse Theia

RIDE 2.0 разработана на базе платформы Eclipse Theia. Theia представляет собой программный комплекс (фреймворк) с открытым исходным кодом, позволяющий реализовать на своей основе различные интегрированные среды разработки. На рисунке 1 представлено изображение графического интерфейса среды разработки mBed, разработанной на базе платформы Theia [6].

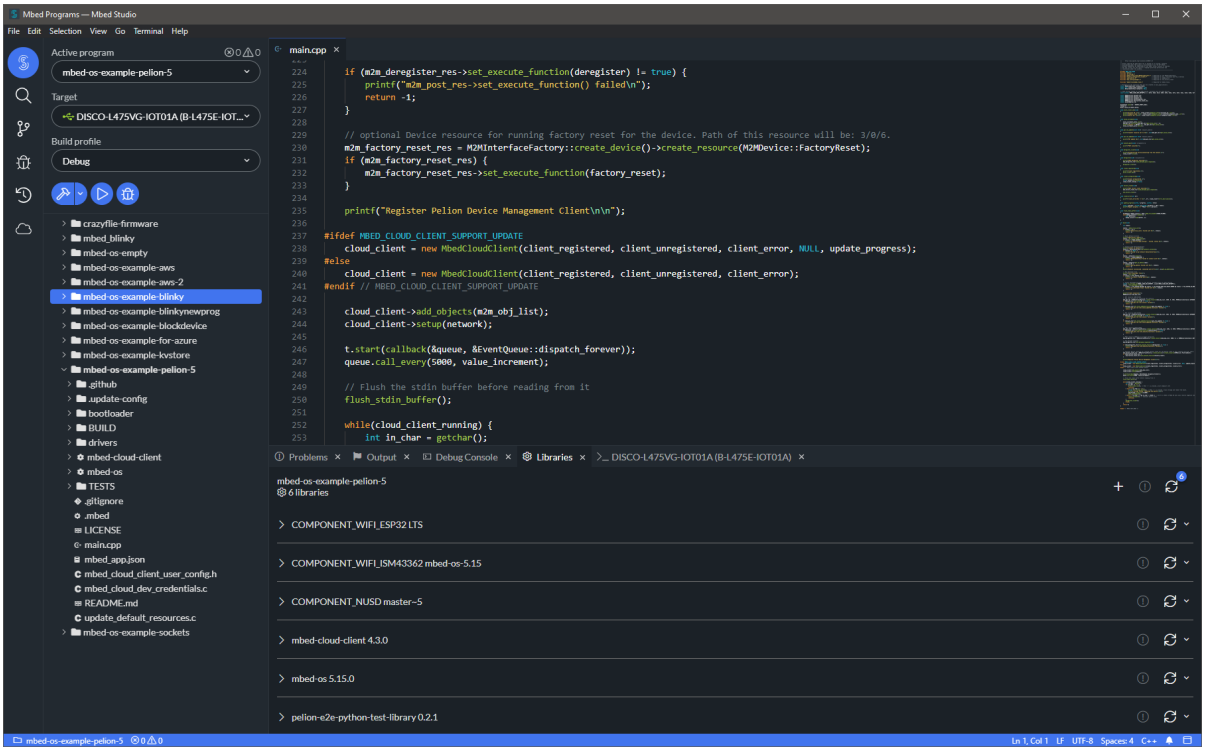

Рисунок 1. mBed — IDE на базе платформы Theia.

Платформа Theia представляет собой набор Node.JS пакетов, собранных в один Node.JS пакет, используя механизм зависимостей. Node.JS — это платформа, позволяющая использовать язык JavaScript как язык общего назначения, а не только как язык для разработки клиентских модулей WEB-приложений. Помимо непосредственно среды исполнения и интеграций с операционными системами, Node.JS предлагает пакетный менеджер — Node Package Manager (NPM) и структуру оформления пакетов, предполагающий

использование специальной конфигурации (package.json). NPM позволяет публиковать пакеты в регистре (npmjs.com), а также выстраивать цепочки зависимостей между пакетами [7].

Package.json представляет собой файл в корневом каталоге пакета, содержащий объект в нотации JSON. Объект представляет собой множество пар ключ-значение. Ключ может представлять собой строку или число, значением могут быть объекты, строки, числа, массивы, булевое значение. Часть пар ключ-значение используется самой системой Node.JS, поэтому их формат формализован. Другие пары можно задавать самостоятельно и использовать в коде, модифицирующем процесс сборки, либо в самом коде программы [8].

Несмотря на то, что Node.JS это платформа для работы с кодом на языке JavaScript, Theia реализована на языке TypeScript, это возможно благодаря тому, что TypeScript изначально предназначен для компиляции исходного кода в JavaScript. Для успешной поддержки реализации пакета на языке TypeScript, необходимо добавить в зависимости пакет “typescript”, содержащий транслятор TypeScript кода в JavaScript. Также необходимо добавить в корень проекта файл “tsconfig.json“, содержащий конфигурацию для транслятора.

В рамках платформы Theia уже реализованы следующие функции, которые могут быть добавлены в среду разработки: настраиваемый виджет редактора кода, виджет терминала, виджет навигации по файловой системе и многие другие. Каждая функция опубликована в отдельном NPM пакете [5].

Создание среды разработки на базе Theia предполагает создание нового Node.JS пакета, который будет использовать в качестве зависимостей пакеты из платформы Eclipse Theia, а также добавлять новую функциональность[9].

Среди пакетов можно выделить обязательный пакет ядра (@theia/core), в котором реализован механизм, связывающий все остальные пакеты. Остальные пакеты из платформы именуются расширениями (theia extension).

Пакеты на старте приложения регистрируют свое содержимое в ядре, используя механизм Dependency Injection, а именно его реализацию “inversify.js”. Само содержимое пакетов с точки зрения ядра представляет собой различные классы на языке TypeScript, которые могут коммуницировать с другими классами, как в рамках своего пакета, так и с классами из других пакетов [10].

2. RIDE 2.0

RIDE 2.0 является Node.JS пакетом, который зависит от пакетов Eclipse Theia, и добавляет в платформу поддержку языка Reflex, а также принципом организации встраиваемых модулей — domain specific modules (DSM). Пакет RIDE 2.0 на данный момент не опубликован в регистре NPM. Поэтому невозможно получить доступ к исходным кодам или собранной версии приложения, используя пакетный менеджер. Но доступ можно получить на GitHub странице проекта [5].

Интеграция языка Reflex в среду разработки реализована в двух плоскостях:

1. Интеграция в интерактивный редактор кода. Произведена путем добавления конфигурации для встроенного в Theia редактора кода. И путем добавления модуля, реализующего Language Server Protocol для Reflex программ.
2. Интеграция API доступа к AST Reflex программы внутри серверного модуля. Произведена путем добавления в систему класса, объект которого может предоставить доступ к AST Reflex программы через вызов соответствующего метода.

RIDE 2.0 является клиент-серверным WEB-приложением. Использование среды разработки происходит через браузер на компьютере пользователя. В браузере исполняется код, отвечающий за отображение информации пользователю, получение и передачу информации о пользовательских действиях на сервер. На сервере исполняется бóльшая часть бизнес-логики приложения. Таким образом, каждый пользователь имеет свой экземпляр клиентского приложения, которое запускается при переходе в браузере по адресу, на котором развернута система. Серверная же часть существует в одном экземпляре и существует в независимости от того, сколько пользователей используют систему в данный момент и используют ли вообще.

Такой подход является ключевым, так как позволяет развернуть платформозависимую функциональность единожды — на удаленном сервере. И не накладывать обязанности по локальной разверстке сложных вычислительных систем на плечи пользователей.

С другой стороны, на серверную логику накладываются некоторые ограничения. Один и тот же код будет параллельно использован несколькими пользователями. Если этот код сохраняет какое-либо состояние в промежутке между запросами от пользователей, то состояние становится общим. А общее состояние приводит к необходимости его синхронизации.

Решением данной проблемы может выступать использование принципа “отсутствия состояния клиент-серверных систем”, который заключается в том, что сам сервер никогда не содержит информации о текущем состоянии клиента, любая информация о состоянии клиентской части приложения должна быть передана серверу в самом запросе от клиента. Отсутствие сохраненного состояния о клиенте на сервере локализует состояние до рамок конкретного запроса и убирает необходимость в синхронизации этого состояния [11].

RIDE 2.0, как и платформа Theia, построены согласно упомянутому принципу. При разработке расширений также рекомендуется ему следовать.

3. Механизм расширения Eclipse Theia

Платформа Theia изначально предполагает пакетный подход организации кодовой базы, который уже может быть использован в качестве механизма расширения. Рассмотрим более подробно структуру Theia и реализацию расширений, предполагаемую данным подходом.

Как было упомянуто, ядро пытается связать пакеты, используя механизм Dependency Injection. Пакеты должны регистрировать свое содержимое сами, код регистрации представляет собой вызовы глобальных функций из библиотеки “inversify.js”, которые позволяют сообщить контейнеру зависимостей о том, что нужно добавить новый класс, содержащий реализацию некой функциональности, или добавить новую реализацию уже известного для системы интерфейса.

Например, для расширения, содержащего клиентский код, может понадобиться добавить в контейнер зависимостей новую реализацию элемента меню действий или новую текстовую команду (См. Рис. 2) и так далее.

```
import { HelloWorldCommandContribution, HelloWorldMenuContribution }
from './hello-world-contribution';
import {
    CommandContribution,
    MenuContribution
} from "@theia/core/lib/common";

import { ContainerModule } from "inversify";

export default new ContainerModule(bind => {
    bind(CommandContribution).to(HelloWorldCommandContribution);
    bind(MenuContribution).to(HelloWorldMenuContribution);
});
```

Рисунок 2. Регистрация клиентского модуля расширения, выводящего сообщение по нажатию кнопки.

Для того чтобы код регистрации исполнился, необходимо чтобы в конфигурации package.json объекте было указано поле “theiaExtension”, в котором необходимо указать пути до файлов с регистрацией (См. Рис. 2). В примере файлы регистрации имеют имя “example-module.ts” [9].

```
├── package.json
├── tsconfig.json
└── src
    ├── browser
    │   └── example-module.ts
    └── node
        └── example-module.ts
```

Рисунок 3. Минимальная структура пакета расширения.

На рисунке 3 видно, что файла регистрации два. Это связано с тем, что итоговое приложение разделено на клиентский модуль (browser) и серверный

модуль (backend). Оба файла опциональные, допустимо чтобы расширение имело только клиентскую или только серверную часть, либо обе.

Регистрация модулей происходит в разное время и на разных окружениях. Регистрация серверного модуля будет происходить на старте сервера в окружении сервера, регистрация клиентского — когда пользователь перейдет на страницу приложения в браузере пользователя.

Помимо регистрации самих модулей, классы клиентского модуля должны регистрировать себя в виде элементов графического интерфейса платформы Theia (См. Приложение 1) — всплывающих сообщений, кнопок, элементов меню, элементов панели действий и тп.

Механизм расширения Eclipse Theia ограничен только возможностями Node.js. При необходимости пакеты Theia могут быть заменены полностью или частично путем создания нового пакета на основе исходного кода оригинального.

Интеграция новых расширений в систему предполагает только добавление зависимости в файл package.json в виде строки с указанием названия NPM пакета и версии пакета.

Такая гибкость накладывает ограничение в виде высокого порога входа для разработчиков расширений. Для успешной разработки расширения, следуя подходу Theia, необходимо понимать внутреннее устройство платформы Theia: принцип регистрации модулей, принцип регистрации элементов пользовательского интерфейса. Учитывая то, что платформа Theia находится на стадии бета-тестирования, сообщество и база знаний платформы немногочисленные, высокий порог входа может стать критичным отталкивающим фактором для новых разработчиков расширений RIDE 2.0.

4. Механизм расширения RIDE 2.0

В рамках реализации RIDE 2.0 разработчиками был добавлен механизм, позволяющий расширять функциональность RIDE 2.0 с помощью так называемых domain specific modules (DSM). Также описана процедура интеграции DSM в систему. Так как данный механизм также призван обеспечить расширяемость системы, необходимо проанализировать его.

Механизм представляет собой расширение Theia с названием “dsm-wrapper”. Пакет содержит browser и node модули, для клиентской и серверной части соответственно [5].

Серверный модуль помимо кода серверной части содержит директорию для файлов конфигурации DSM. Для каждого DSM в данной директории необходимо добавить файл. При старте приложения серверный модуль читает файлы из директории и на основе данных оттуда производится подключение DSM.

Расширения при описанном подходе должны являться Spring приложениями на Java, реализующим один HTTP метод с произвольным набором параметров. Приложение должно быть развернуто на том же сервере, что и RIDE 2.0. В файле конфигурации DSM должна храниться информация о порте, на котором Spring приложение развернуто. Серверный модуль на старте подключается к указанному порту [5].

Клиентский модуль “dsm-wrapper” на основе конфигурационных файлов создает для каждого расширения пункт контекстного меню второго уровня, который становится доступен при выборе файла Reflex программы в виджете файловой системы. После нажатия на пункт контекстного меню клиентский модуль открывает форму ввода, в которой пользователь может задать параметры для метода Spring приложения. Возможные поля формы и их типы должны быть перечислены в конфигурационном файле DSM. После

подтверждения формы dsm-wrapper запрашивает из “AST-Service” модуля AST выбранного Reflex файла. Далее из AST и результата формы формируется HTTP запрос в Spring приложение и выводится полученный результат в виде текста на экран.

Интеграция нового расширения при таком подходе происходит посредством добавления конфигурационного файла в каталог в пакете “dsm-wrapper” и разверткой приложения на сервере по порту из конфигурационного файла.

Метод интеграции, очевидно, более трудоемкий по сравнению с подходом интеграции расширений Theia. Необходимо каким-то образом получить собранный jar файл Spring приложения для его развертки, либо собирать его непосредственно перед интеграцией руками администратора сервера.

Также помимо более трудоемкой интеграции данный подход обеспечивает немногочисленные возможности по настраиванию пользовательского интерфейса расширения — только возможность выбрать набор полей формы и их тип. Таких возможностей будет недостаточно при реализации графических редакторов с нетривиальной логикой пользовательского взаимодействия.

При этом, данный механизм полностью абстрагирует разработчика расширения от внутреннего устройства Theia — необходимо только соблюсти контракт взаимодействия по HTTP, оформить конфигурационный файл согласно формату. Такой подход сильно минимизирует порог входа для разработчиков.

5. Требования к механизму расширения

Таким образом, RIDE 2.0 уже содержит два механизма расширения. Один предоставляет гибкость результата за счет тесной интеграцией

расширения с платформой Theia, другой наоборот — предоставляет легкость разработки расширения, жертвуя гибкостью. Необходимо предоставить некую “золотую середину” — решение, обеспечивающее больше гибкости получаемого расширения, при этом не заставляя разработчиков расширения изучать внутреннее устройство платформы Theia.

Помимо описанных особенностей, оба механизма расширения предполагают наращивание функциональности посредством добавления нового кода в систему, а не модификации старого — это преимущество необходимо также сохранить. Так как такой подход позволяет обеспечить бóльшую устойчивость системы к изменениям.

Принимая во внимание проанализированные существующие решения, были сформулированы следующие требования к разрабатываемому механизму расширения:

1. Механизм расширения должен предоставлять возможность создания расширений с собственным графическим интерфейсом.
2. Механизм расширения должен абстрагировать разработчиков от внутреннего устройства Theia, а именно от механизма регистрации модулей, механизма регистрации элементов пользовательского интерфейса.
3. Механизм расширения должен предоставлять процесс интеграции новых расширений путем только модификации конфигурационных файлов.
4. Механизм расширений должен предполагать расширение системы посредством добавление новых пакетов/модулей, а не модификации существующих.

3. ПРОЕКТИРОВАНИЕ

1. Автогенерация базового репозитория

Так как существует механизм, позволяющий обеспечить максимальную широту возможностей при разработке расширения, логично взять данный подход как основу нового механизма расширения. Возможно сжать гибкий и абстрактный контракт (механизм расширения Theia), жертвуя какими-либо возможностями, но невозможно расширить и так узкий и конкретный контракт (Механизм расширения RIDE 2.0 DSM).

Необходимо абстрагировать гибкий подход от внутренних механизмов Theia. Для этого код интеграции с механизмами Theia должен быть изолирован от остального кода расширения. Код интеграции с Theia должен быть реализован в рамках разработки механизма расширения. Остальной код расширения, изолированный от интеграции — будет разрабатываться в процессе разработки конкретных расширений. Изоляцию необходимо производить для каждого из модулей — клиентского и серверного.

Код интеграции должен каким-то образом попадать в итоговое расширение. Для таких задач используют различные подходы — выделение кода в подключаемую библиотеку или автогенерация кода.

Было предложено использовать автогенерацию базового кода расширения при инициализации репозитория расширения. В данной задаче автогенерация является более предпочтительной, так как такой подход позволяет предоставить возможность при необходимости или желании разработчика расширения редактировать код интеграции. В то время как при использовании подхода библиотеки, внесение изменений в реализацию библиотеки под конкретную задачу будет более затруднительным процессом.

Таким образом, автогенерация позволит автоматически добавить в расширение необходимый для интеграции с платформой Theia код, при этом

код интеграции не будет спрятан в библиотеке — будет доступен для анализа и модификации.

2. Клиентская часть расширения

Необходимо предоставить возможность создания расширений с собственным графическим интерфейсом. Но на данный момент добавление собственного графического интерфейса сопровождается реализацией интеграции с Theia. Одним из элементов графического представления Theia является так называемый Widget. Этот элемент представляет собой область на экране, содержимое которого может быть программно определено. В приложении могут существовать несколько Widget разных размеров и положений. На рисунке 4 изображен интерфейс среды разработки на базе Theia. Поля редактора кода, навигатор файловой системы, визуализация диаграмм реализованы на основе Widget [12].

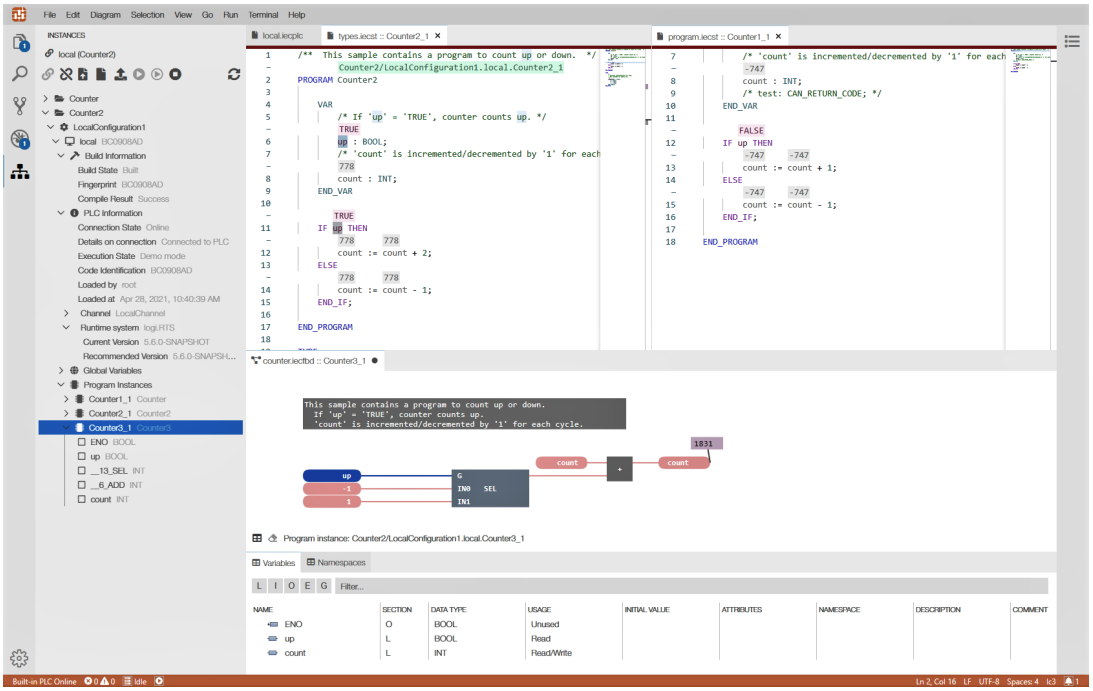

Рисунок 4. Элементы интерфейса, реализованные с помощью механизма Widget.

Другие стандартные элементы графического представления Theia не могут предоставить необходимой гибкости, потому что не позволяют внедрить произвольные графические элементы внутри себя. К таким механизмам относятся Button, Message, Command, MenuItem. Однако, с помощью некоторых из них возможно реализовать механизм скрытия и появления Widget.

Таким образом, механизм Widget позволяет реализовать сложные графические элементы. Widget представляет собой описание протокола взаимодействия элемента графического представления с Theia. Одной из реализаций протокола является ReactWidget, позволяющий использовать библиотеку React.js при разработке графического представления.

Было предложено использовать ReactWidget как основу для клиентского модуля расширений, так как реализация графического представления с использованием React.js является на данный момент наиболее популярным подходом разработки клиентских приложений. React.js как технология старше Theia, а также имеет большее сообщество, а следовательно количество уже решенных и описанных задач и проблем [13]. Ряд существующих в Theia модулей также реализованы с использованием ReactWidget. Помимо самого Widget, необходим механизм вызова новой области на экран. В Theia этого можно добиться через регистрацию обработчиков на элементы меню инструментов или через регистрацию новых текстовых команд.

Таким образом, каждое расширение будет добавлять в систему новый ReactWidget, элемент в меню и текстовую команду, которые служат за открытие области, реализованной с помощью ReactWidget. Интеграция ReactWidget с платформой Theia будет сгенерирована, разработчику расширения будет необходимо реализовать сам графический интерфейс расширения, используя React.js, а также добавить логику взаимодействия с серверным модулем.

3. Серверная часть расширения

С точки зрения серверного модуля интеграция с платформой Theia представляет собой добавление новых классов языка TypeScript в Dependency Injection контейнер. Было предложено реализовать автогенерацию добавления одного класса для каждого расширения. Этого будет достаточно, так как классы можно расширить произвольным количеством методов. Разработчику расширения необходимо будет определить набор методов и реализовать их.

Несмотря на то, что расширение регистрирует один класс в контейнере зависимостей, внутри самого расширения разработчик сможет разделить логику на необходимое ему количество классов/функций/процедур, но для остальных расширений и клиентского модуля серверный модуль конкретного расширения будет представлять собой один объект с заданным интерфейсом.

4. Структура расширения

На рисунке 5 представлена схема сгенерированного базового репозитория расширения и связи расширения с Theia. Расширение разделено на три подмодуля, для каждого из которого будет представлен отдельный каталог в репозитории.

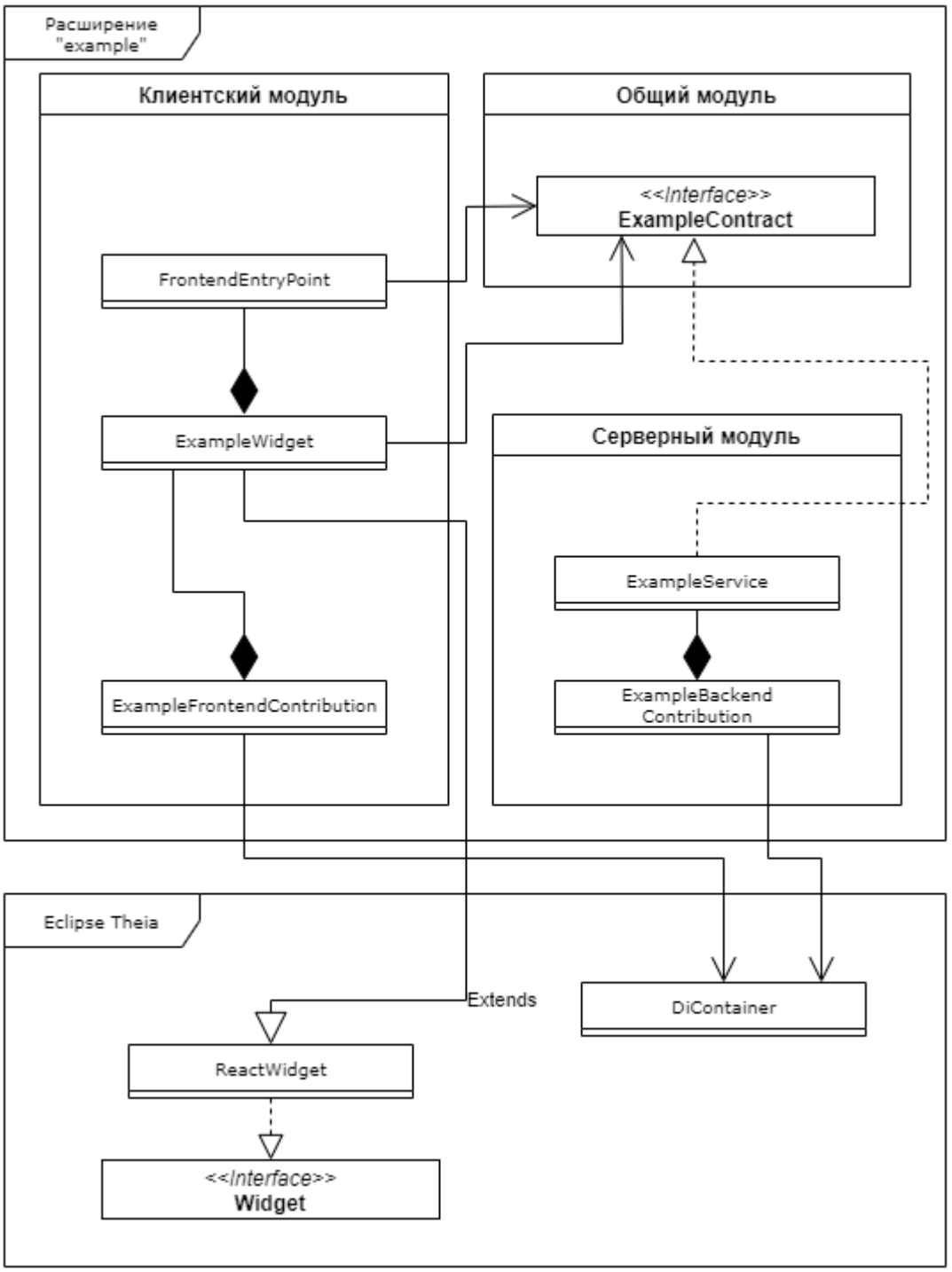

Рисунок 5. Структура расширения и зависимости окружения Theia

Общий модуль содержит один TypeScript интерфейс — контракт клиент-серверного взаимодействия. Клиентский модуль использует этот интерфейс для взаимодействия с серверным модулем, а серверный модуль должен реализовывать данный интерфейс.

Серверный модуль содержит ExampleService — класс, реализующий контракт взаимодействия клиента и сервера. А также ExampleBackendContribution, в котором содержится логика регистрации ExampleService в платформе Theia.

Клиентский модуль содержит три класса. ExampleFrontendContribution реализует регистрацию Widget, элементов меню и команд в приложение. FrontendEntryPoint это точка входа для React.js кода разработчика расширения, модифицируя этот компонент, разработчик может задать специфичную логику клиентского приложения, также во FrontendEntryPoint доступна ссылка на реализацию серверной части для сообщения с сервером. ExampleWidget это наследник ReactWidget, цель которого предоставить интерфейс взаимодействия согласно протоколу Widget, а также, используя Dependency Injection механизм, получить ссылку на реализацию серверного контракта и передать ее в FrontendEntryPoint.

5. Выбор технологии для реализации автогенерации

Для реализации автогенерации существует множество инструментов, наиболее популярные — jHipster и Yeoman. Оба инструмента доступны в виде Node.JS пакета, имеют открытый исходный код. Было решено использовать Yeoman, так как он более легковесен, потому что нацелен на реализацию одной задачи, а не на реализацию множества задач, как jHipster [14].

4. РЕАЛИЗАЦИЯ

1. Реализация механизма автогенерации

Для автогенерации за основу была взята библиотека Yeoman. Yeoman предоставляет инструментарий по работе с файловой системой, взаимодействием с пользователем через терминал через вывод сообщений и обработку пользовательского ввода.

Генератор репозитория представляет собой отдельный Node.JS пакет, точкой входа в который является анонимный JavaScript класс. Анонимный класс наследует класс Generator из библиотеки Yeoman.

Анонимный класс содержит набор методов, каждый из этих методов при запуске будет выполнен в порядке объявления. В методах сгруппированы по смыслу команды, выполняющие генерацию. Для доступа к интерфейсу файловой системы и взаимодействием с пользователем используются методы Generator (См Рис. 6).

```
prompting() {
  this.log(
      yosay(`Welcome to the breathtaking
${chalk.red('generator-reflex-ide-extension')} generator!`)
    );
    const prompts = [
      {
        type: 'input',
        name: 'name',
        message: `Enter extension's name ${chalk.red('KEBAB CASE
ONLY')} (files and class names will be based on it):`,
        default: true
      }
    ];
    return this.prompt(prompts).then(props => this.props = props);
  }
```

Рисунок 6. Приветствие пользователя и получение имени расширения

Основной этап генерации — это копирование шаблонов интеграции с платформой Theia. Все шаблоны хранятся внутри генератора в отдельном каталоге.

Так как у каждого расширения свое уникальное название, генератор на старте предлагает пользователю ввести желаемое имя итогового расширения. Генератор производит предобработку шаблонов с проставлением введенного имени в необходимых названиях.

Помимо генерации файлов генератор вносит изменения в список зависимостей RIDE 2.0, чтобы указать, что необходимо включить в сборку новое расширение. Эти изменения в список зависимостей будут иметь силу только локально у пользователя, и не будут влиять на основной репозиторий RIDE 2.0 (См Рис 7). Так как новая зависимость в момент локальной разработки еще не опубликована в регистре NPM, необходимо добавить Workspace зависимость, это сообщит Node.JS, что зависимость необходимо искать локально.

```
addWorkspace() {
  const path = this.destinationPath('package.json');
  const original = this.fs.readJSON(path);
  original.workspaces.push(this.props.name);
  this.fs.write(path, JSON.stringify(original));
}

addDependencies() {
  const addDependency = (path) => {
    const original = this.fs.readJSON(path);
    original.dependencies[this.props.name] = "0.0.0";
    this.fs.write(path, JSON.stringify(original));
  }
  addDependency(this.destinationPath('browser-app/package.json'));
  addDependency(this.destinationPath('electron-app/package.json'));
}
```

Рисунок 7. Добавление зависимостей в RIDE 2.0

В результате применения генератора в локальном репозитории RIDE 2.0, будет создано расширение в виде каталога, содержащего все необходимые файлы для интеграции с платформой Theia. При запуске приложения будет доступен новый пункт меню, соответствующий созданному расширению. При выборе указанного пункта открываться дополнительная область интерфейса с текстом и кнопкой. По нажатию на кнопку происходит запрос на сервер, после чего полученный ответ отображается в области (см. Рис. 8).

```
export const BackendResponseWidget: FC<BackendResponseWidgetProps> =
({backend}) => {
    const [text, setText] = useState('none text yet');
    const retrieveText = useCallback(
        () => backend.getTextToDisplay()
            .then(response => setText(response.message)),
        [backend]
    )

    return (
        <div>
            <button onClick={retrieveText}>click here to retrieve new
message from backend</button>
            <div style={{marginTop: '10px'}}>
                {text}
            </div>
        </div>
    )
}
```

Рисунок 8. Реализация запрашивания и вывода сообщения с сервера

Генератор был опубликован в регистре NPM и находится в общем доступе.

2. Описание процесса разработки расширения

Был предложен следующий процесс разработки расширения:

1. Разработчику расширения необходимо установить GIT, а также Node.JS версии 10.*.
2. Разработчику расширения необходимо клонировать репозиторий RIDE 2.0 для локальной разработки и тестирования. Команда:

```
git clone
https://github.com/TatianaLiakh/RIDE-2.0.git
```

3. Разработчику необходимо установить исполняемые файлы инструмента Yeoman, используя пакетный менеджер NPM, поставляемый вместе с Node.JS. Команда:

```
npm install -g yo
```

4. Разработчику необходимо установить генератор репозитория, также используя NPM. Команда:

```
npm install -g generator-reflex-ide-extension
```

5. Разработчику необходимо запустить генератор в директории репозитория RIDE 2.0 и указать имя желаемого расширения. Так как имена пакетов NPM должны быть уникальными следует убедиться, что пакета с таким именем еще не существует в реестре, а также использовать постфикс “reflex-ide-extension” для единообразия [15]. Команда:

```
yo reflex-ide-extension
```

6. Разработчику необходимо реализовать необходимую функциональность, добавляя реализацию в FrontendEntryPoint и

ExampleService. При разработке следует соблюдать принцип отсутствия состояния при разработке серверного модуля.

7. Разработчику необходимо опубликовать расширение. Процедура публикации стандартна для NPM и приведена в ReadMe файле репозитория генератора [16]. Также в ReadMe можно найти наиболее актуальную информацию об изменениях в процедуре разработки и интеграции.

На этом разработка первой версии расширения завершается.

3. Описание процесса интеграции расширения

Новые расширения необходимо интегрировать в среду разработки администратором сервера. Процедура интеграции повторяет процедуру интеграции при гибком подходе интеграции расширений Theia. Для интеграции необходимо добавить зависимость в файл package.json проекта RIDE 2.0, после чего перезапустить процесс развертывания с новыми исходными файлами конфигурации.

При развертывании Node.js загрузит все новые зависимости из регистра NPM.

.

5. ЗАКЛЮЧЕНИЕ

Было проведено исследование способа расширения RIDE 2.0, был предложен механизм расширения, предполагающий использование созданного генератора репозитория на базе библиотеки Yeoman, была описана процедура разработки и интеграции расширений.

За основу механизма расширения был взят гибкий подход расширения родительской платформы Theia, удовлетворяющий большей части требований к механизму расширения. Соответствия остальным требованиям удалось добиться благодаря автогенерации репозитория.

Генератор репозитория публично доступен и может быть установлен с помощью NPM.

Разработанное решение унифицирует процесс разработки и интеграции расширений для RIDE 2.0 и уменьшает порог входа при разработке расширений с собственным графическим интерфейсом.

На данный момент проводится исследование на тему расширения возможностей механизма, позволяющих встраивать модули, написанные на других языках программирования.

Также планируется исследовать расширения, которые будут написаны с использованием предложенного механизма, чтобы выявить наиболее часто встречающиеся шаблоны и вынести их в отдельную конфигурацию генератора.

Помимо этого для добавляемых расширений планируется продумать и описать процедуру проверки на отсутствие вредоносного кода при интеграции в систему.

Работа была представлена на Международной научной студенческой конференции 2022, а также была отмечена дипломом первой степени на XXIV

Всероссийской студенческой научно-практической конференции Нижневартовского государственного университета в секции очных докладов.

Выпускная квалификационная работа выполнена мной самостоятельно и с соблюдением правил профессиональной этики. Все использованные в работе материалы и заимствованные принципиальные положения (концепции) из опубликованной научной литературы и других источников имеют ссылки на них. Я несу ответственность за приведенные данные и сделанные выводы.

Я ознакомлен с программой государственной итоговой аттестации, согласно которой обнаружение плагиата, фальсификации данных и ложного цитирования является основанием для не допуска к защите выпускной квалификационной работы и выставления оценки «неудовлетворительно».

Витченко Владислав Алексеевич _____

Подпись студента

«20» мая 2022 г

6. СПИСОК ЛИТЕРАТУРЫ

1. Зюбин В. Е. Процесс-ориентированное программирование: Учеб. пособие // Новосибирск. Новосиб. гос. ун-т. 2011. 194 с.
2. Зюбин В. Е. " Си с процессами"-язык программирования логических контроллеров //Мехатроника, автоматизация, управление. – 2006. – №. 12. – С. 31-35.
3. Горнев, И. А. Создание среды разработки для языка Reflex: диплом. работа / И. А. Горнев. – Новосибирск : Новосиб. гос. ун-т., 2019.
4. Зюбин В. Е. Использование виртуальных объектов для обучения программированию информационно-управляющих систем //Информационные технологии. – 2009. – №. 6. – С. 79-82.
5. Марченко, К. В. Архитектура системы для групповой WEB-разработки программ на процесс-ориентированном языке Reflex: диплом. работа / К. В. Марченко. – Новосибирск : Новосиб. гос. ун-т., 20.
6. Eclipse Theia Overview. [Электронный ресурс] // Eclipse Theia. URL: https://theia-ide.org/#features (дата обращения: 05.05.2022).
7. Node.js official docs. [Электронный ресурс] // Node. URL: https://nodejs.org/en/about/ (дата обращения: 05.05.2022).
8. Package.json. [Электронный ресурс] // NPM JS. URL: https://docs.npmjs.com/cli/v7/configuring-npm/package-json (дата обращения: 05.05.2022).
9. Authoring Theia Extensions. [Электронный ресурс] // Eclipse Theia. URL: https://theia-ide.org/docs/authoring_extensions/ (дата обращения: 05.05.2022).
10. Build your own ID [Электронный ресурс] // Eclipse Theia. URL: https://theia-ide.org/docs/composing_applications/ (дата обращения: 05.05.2022).

11. Khare R., Taylor R. N. Extending the representational state transfer (rest) architectural style for decentralized systems //Proceedings. 26th International Conference on Software Engineering. – IEEE, 2004. – С. 428-437.
12. Widgets [Электронный ресурс] // Eclipse Theia. URL: https://theia-ide.org/docs/widgets/ (дата обращения: 05.05.2022).
13. Front-end frameworks and libraries [Электронный ресурс] // State of JS 2021. URL: https://2021.stateofjs.com/en-US/libraries/front-end-frameworks/ (дата обращения: 05.05.2022).
14. JHipster VS Yeoman [Электронный ресурс] // StackShare. URL: https://stackshare.io/stackups/jhipster-vs-yeoman (дата обращения: 05.05.2022).
15. NPM About [Электронный ресурс] // NPM JS. URL: https://docs.npmjs.com/about-npm (дата обращения: 05.05.2022).
16. Generator Reflex IDE Extension [Электронный ресурс] // GitHub. URL: https://github.com/Oladik123/generator-reflex-ide-extension (дата обращения: 05.05.2022).

ПРИЛОЖЕНИЕ 1

```
import { injectable, inject } from "inversify";
import { CommandContribution, CommandRegistry, MenuContribution,
MenuModelRegistry, MessageService } from "@theia/core/lib/common";
import { CommonMenus } from "@theia/core/lib/browser";

export const HelloWorldCommand = {
    id: 'HelloWorld.command',
    label: "Shows a message"
};

@injectable()
export class HelloWorldCommandContribution implements CommandContribution {

    constructor(
                @inject(MessageService) private readonly messageService:
MessageService,
    ) { }

    registerCommands(registry: CommandRegistry): void {
        registry.registerCommand(HelloWorldCommand, {
            execute: () => this.messageService.info('Hello World!')
        });
    }
}

@injectable()
export class HelloWorldMenuContribution implements MenuContribution {

    registerMenus(menus: MenuModelRegistry): void {
        menus.registerMenuAction(CommonMenus.EDIT_FIND, {
            commandId: HelloWorldCommand.id,
            label: 'Say Hello'
        });
    }
}
```

Рисунок 4. Регистрация привязки элементов интерфейса Theia для расширения, выводящего сообщение по нажатию кнопки.