Санкт-Петербургский государственный университет

Кафедра системного программирования

Глазырин Кирилл Максимович

Портирование инструментов автоматизации Visual Studio в Rider

Курсовая работа

Научный руководитель:
к. т. н., доцент Литвинов Ю. В.

Санкт-Петербург
2020

SAINT-PETERSBURG STATE UNIVERSITY

Software engineering

Kirill Glazyrin

Porting Visual Studio automation tools to Rider

Course Work

Scientific supervisor:
C. Sc., Associate Professor Yurii Litvinov

Saint-Petersburg
2020

Оглавление

Введение

При разработке программного обеспечения может возникать потребность многократно выполнять однотипные действия над кодом; автоматизация этого процесса может значительно увеличить производительность труда программиста. Один из способов автоматизировать это — написать программу, которая работает с деревьями разбора файлов проекта.

Visual Studio — среда разработки от компании Microsoft — предоставляет программный интерфейс для этого. Это библиотека под названием „EnvDTE“. Используя её, можно получить доступ к абстракциям, значительно упрощающим работу, например, к информации о структуре проекта и абстрактным синтаксическим деревьям, построенным по файлам с кодом. Важная черта этой библиотеки в том, что она используется при создании расширений к среде разработки, но она позволяет взаимодействовать с моделью кода и подсистемами среды разработки без написания плагинов к ней. Например, в T4[1] существует простой способ получить доступ к объектам EnvDTE и взаимодействовать со средой разработки напрямую из пользовательского кода [12]. Благодаря такому удобству этот механизм используется во многих[2] проектах на платформе .NET.

Но у этой технологии есть важный недостаток: она не универсальна. Код из этой библиотеки может быть исполнен только во время работы Visual Studio, из-за чего эти инструменты невозможно использовать в окружениях, в которых эта среда разработки недоступна. В частности, при работе в Rider — среде разработки, созданной компанией JetBrains.

В связи с этим разработчики проектов, использующих этот инструмент, оказываются привязаны к Visual Studio и не могут использовать Rider. Поэтому в компании JetBrains появилась потребность в решении

[1] T4 — шаблонный язык, используемый в .NET для генерации текста или кода. Он состоит из специфичных директив, частей текста и кода на C#. Этот код исполняется до сборки основного проекта.

[2] Согласно сервису по обмену кодом NuGet, на настоящий момент (2020 год) библиотека EnvDTE скачивается 850 раз в день и используется в таких проектах, как Roslyn, ILSpy, gitextensions. Актуальную информацию об использовании этой сборки можно найти на странице https://www.nuget.org/packages/EnvDTE/

этой проблемы.

Возникла идея сделать альтернативный инструмент, который предоставлял бы те же абстракции работы с кодом, но не опирался бы на Visual Studio.

Rider при работе с кодом использует абстракции, схожие с абстракциями EnvDTE, поэтому ключевой шаг в реализации предлагаемой библиотеки — получение модели кода, идентичной модели кода Visual Studio, на основе данных из Rider.

EnvDTE используется в пользовательском коде, а в среде разработки Rider пользовательский код, который может использовать данную технологию, исполняется в процессе, отдельном от процесса среды разработки. Поэтому предлагаемая библиотека должна загружаться в пользовательский процесс и при помощи некоторых инструментов межпроцессного взаимодействия получать из Rider необходимую информацию.

1. Постановка задачи

Цель данной работы заключается в создании библиотеки, предоставляющей такой же интерфейс проектной и кодовой модели, как EnvDTE, но способной работать в среде разработки Rider.

Для того, чтобы достичь поставленную цель, необходимо решить следующие задачи:

- создание библиотеки, имитирующей внешний программный интерфейс EnvDTE;
- создание плагина к среде разработки Rider, позволяющего получать информацию, необходимую для реализации абстракций, аналогичных абстракциям из EnvDTE;
- организация передачи данных между библиотекой, загруженной в пользовательский процесс, и средой разработки;
- интеграция созданного инструмента в подсистемы среды разработки Rider, нуждающиеся в работе с EnvDTE.

2. Обзор предметной области

2.1. Структура EnvDTE

EnvDTE позволяет взаимодействовать со многими подсистемами IDE (рис. 1), например, отладчиком, процессом сборки решения, графическим интерфейсом и так далее. Такое богатство обусловлено тем, что EnvDTE может быть использована для создания расширений к среде разработки.

Но в проектах, опирающихся на эту библиотеку, направленных на автоматизацию работы с кодом, эти абстракции не требуются. Для них достаточно обращаться к абстракциям проектной модели. Они включают в себя такие интерфейсы, как *Solution*, *Project*, *CodeElement* и так далее. EnvDTE также позволяет взаимодействовать и с деревьями разбора файлов. Доступны такие интерфейсы, как *CodeType*, *CodeNamespace*, *CodeFunction* и так далее. Примечательно, что данная библиотека раскрывает не только синтаксическую структуру кода, но и, частично, его семантику. Так, *CodeClass* позволяет получить список классов и интерфейсов, от которых он наследуется, а *CodeFunction* – её тип.

2.2. Версии EnvDTE

В целях сохранения обратной совместимости разработчики исходной библиотеки не меняли существующие интерфесы EnvDTE, а создавали новые сборки. Поэтому в настоящее время одновременно существуют пять различных версий библиотеки; предлагаемая альтернатива должна позволять работать через любую версию интерфейсов.

2.3. Существующие решения

Потребность в использовании EnvDTE вне Visual Studio существует уже давно, и найдены следующие пути её решения:

- находить при помощи методов межпроцессного взаимодействия

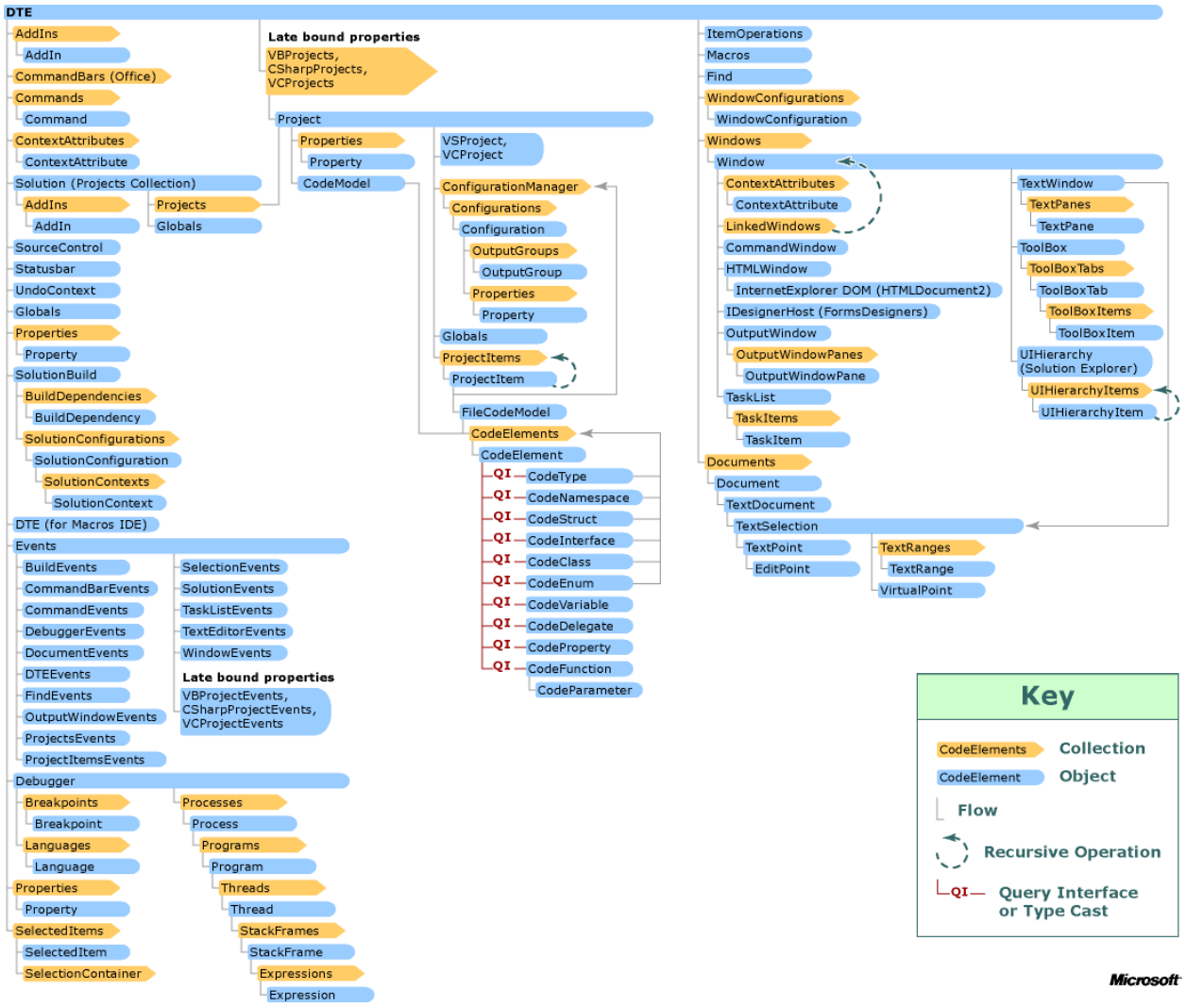

Рис. 1: Ключевые классы EnvDTE [11].

экземпляр Visual Studio, уже исполняющийся на машине, и получать инструменты автоматизации из него [13];

- запускать новый экземпляр Visual Studio без пользовательского интерфейса и получать инструменты автоматизации из него [14].

Оба эти варианта, однако, не универсальны. Первый вариант подразумевает необходимость вручную открывать среду разработки, что может быть нежелательно. Второй вариант не оптимален с точки зрения производительности: даже для небольших проектов инициализация процесса Visual Studio может занять некоторое время. Результаты измерения времени инициализации этого процесса представлены в таблице 1.

И они оба не применимы в окружениях, в которых запуск Visual Studio невозможен. Например, на машинах, на которых не установлена эта IDE, и на всех операционных системах, кроме Windows. Ограничения на платформу вызваны тем, что библиотека EnvDTE специфична для операционной системы Windows и не может быть использована, например, в Visual Studio for Mac.

Наконец, можно удалить из проекта все завязки на EnvDTE и использовать другие инструменты для автоматизации работы с кодом. Это можно сделать, например, на основе инструмента Scripty [5]. Но для крупных проектов эта работа может оказаться трудозатратной и чреватой ошибками, поэтому этот вариант тоже применим не всегда.

Операция	E	σ
Запуск процесса	2.020 сек	0.0075 сек
Запуск процесса и открытие решения	5.229 сек	0.0759 сек

Таблица 1: Результаты измерения времени инициализации Visual Studio при помощи методов межпроцессного взаимодействия [14]. E — математическое ожидание. σ — стандартное отклонение. Решение, открытие которого измерялось, содержало 500 строк кода на C# и не находилось в системе контроля версий. Никакие плагины к среде разработки установлены не были. Измерения проводились на персональном компьютере с процессором i7-8750H CPU 2.20GHz (Coffee Lake) при помощи инструмента BenchmarkDotNet [1].

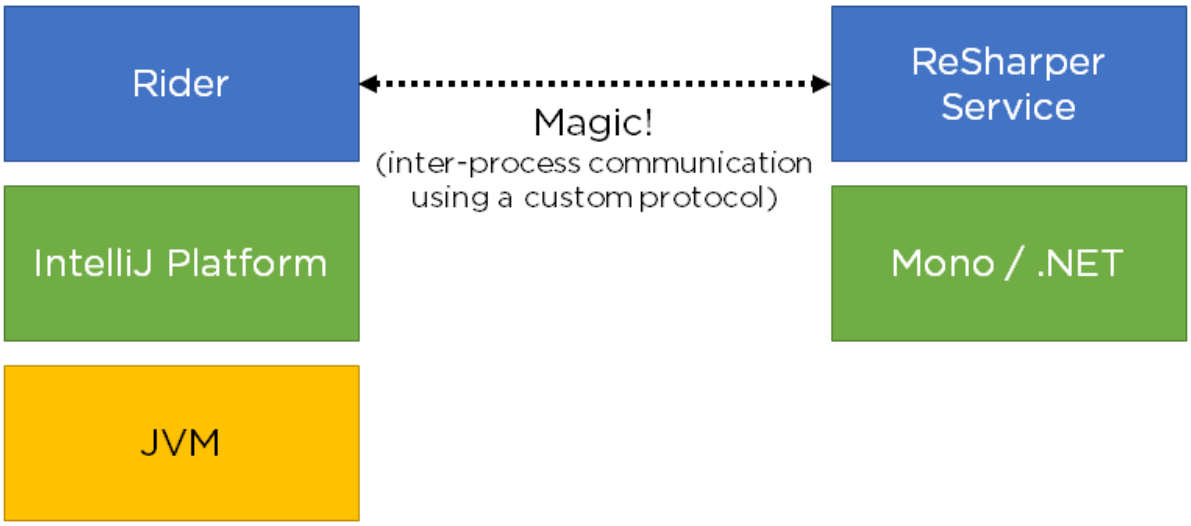

Рис. 2: Архитектура среды разработки Rider [3].

2.4. Rider

2.4.1. Архитектура

Rider [10] — кроссплатформенная среда разработки для .NET от компании JetBrains. Она состоит из двух основных процессов операционной системы: фронтенда и бэкенда. Данная схема изображена на рис. 2.

Фронтенд — видимая часть, пользовательский интерфейс — создан на основе платформы IntelliJ, благодаря чему внешний вид среды разработки имеет много общего с внешним видом многих других продуктов компании JetBrains. Эта часть исполняется на виртуальной машине Java.

Бэкенд — процесс без пользовательского интерфейса, в котором сосредоточена основная логика по работе с кодом — основан на другом продукте компании JetBrains — ReSharper [4]. ReSharper — плагин к среде разработки Visual Studio [9]; для использования его в качестве бэкенда для среды разработки Rider ReSharper был модифицирован – код этого проекта был отвязан от программных интерфейсов Visual Studio. Эта часть исполняется на виртуальной машине .NET. Поскольку эти две части используют различные виртуальные машины, они были выделены в отдельные процессы.

2.4.2. Протокол межпроцессного взаимодействия

Взаимодействие между двумя процессами, составляющими Rider — ключевой момент в работе среды разработки, поэтому для упрощения этой задачи существует библиотека RD [8]. Аббревиатура RD расшифровывается как „Reactive Distributed“, что точно отражает её суть: она предоставляет реактивные сущности (то есть объекты, позволяющие реагировать на изменения, подписываясь на события) и способна работать в распределённых системах. Другое важное свойство этого механизма заключается в том, что он позволяет иметь изменяемое состояние, разделяемое между процессами, и гарантировать его консистентность.

Этот же механизм можно использовать и для того, чтобы обеспечить обмен информацией между процессом, в который загружена предлагаемая библиотека, и средой разработки.

2.4.3. Проектная модель

Модель проекта в Rider имеет много общего с моделью в EnvDTE. Как и Visual Studio, Rider оперирует такими понятиями, как решение, документ, тип, а так же их составляющими: проект, каретка, метод и так далее. Близость этих моделей значительно упрощает задачу данной работы.

2.4.4. Подсистемы, нуждающиеся в EnvDTE

Особенность EnvDTE в том, что её можно использовать не только при написании плагинов к Visual Studio. Например, технология T4 позволяет обращаться к EnvDTE из пользовательского кода. В среде разработки Rider уже есть подсистема, реализующая поддержку T4. Эта система корректно работает в тех случаях, когда пользовательский код не использует EnvDTE, но выдаёт сообщение об ошибке в случае использования этой технологии. Поэтому её применимость несколько ограничена.

3. Предлагаемое решение

В ходе данной работы была реализована часть публичных интерфейсов EnvDTE. Эта реализация была названа JetBrains.EnvDTE. Исходный код был выложен на сервис GitHub по адресу `https://github.com/kirillgla/JetBrains.EnvDTE`. Данное решение поддерживает ключевые функции по работе с моделью проекта и абстрактными синтаксическими деревьями.

3.1. Архитектура решения

Предлагаемое решение (рис. 3) состоит из трёх основных частей:

- сборки, содержащие интерфейсы EnvDTE;
- EnvDTE.Client – сборка, содержащая реализацию этих интерфейсов и логику по отправке запросов;
- EnvDTE.Host – сборка, содержащая логику по обработке запросов.

Сборки с интерфейсами EnvDTE и их реализациями рассчитаны на загрузку в процесс с пользовательским кодом. Сборка с логикой по обработке запросов – на загрузку в процесс бэкенда среды разработки Rider. Предлагаемое решение дублирует версии существующих библиотек и содержит проекты с интерфейсами каждой из версий.

3.1.1. Передача данных между процессами

Существуют разные механизмы, которыми можно воспользоваться для того, чтобы передавать данные между процессом бэкенда Rider и процессом, нуждающемся в EnvDTE. Например, Google gRPC [6] и Apache Thrift [2] являются возможными вариантами. Была выбрана библиотека RD [8]. Её достаточно для того, чтобы создавать соединение между процессом бэкенда Rider и пользовательским кодом; весомым аргументом в её пользу является то, что этот механизм уже используется

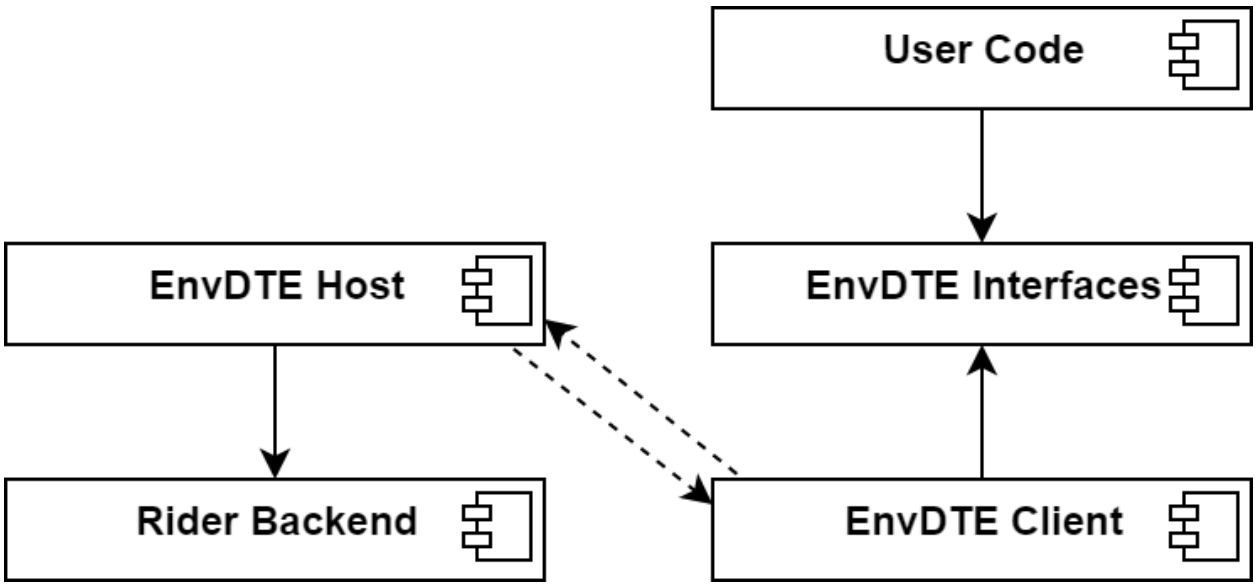

Рис. 3: Архитектура предлагаемой библиотеки.

в Rider. Библиотека RD подразумевает использование rdgen – инструмента для генерации фрагментов кода – классов, которые будут общими для сторон, осуществляющих обмен данными [7]. В предлагаемой библиотеке при помощи него создаются классы модели кода. Например, классы, представляющие проекты и файлы, и узлы абстрактных синтаксических деревьев. Библиотека RD загружается в пользовательский процесс вместе с JetBrains.EnvDTE и передаёт запросы и ответы по каналу связи, работающему на основе сокетов.

Задача передачи информации об абстрактных синтаксических деревьях была разделена на конвертацию конкретного синтаксического дерева в абстрактное и передачу его в другой процесс, поэтому узлы промежуточного дерева дублируют узлы абстрактного синтаксического дерева, строящегося на стороне EnvDTE.

3.1.2. Передача абстрактных синтаксических деревьев

В Rider есть инфраструктура для видонезависимого анализа файлов. Она позволяет построить конкретное синтаксическое дерево, которое называется PSI – Program Structure Interface. При необходимости требуемые узлы преобразуются в промежуточное представление, которое затем передается в пользовательский процесс и преобразуется в

узлы абстрактного синтаксического дерева EnvDTE.

Класс промежуточного представления узлов дерева, используемый обеими сторонами, генерируется при сборке предлагаемой библиотеки. Он хранит идентификатор типа узла и идентификатор узла. Идентификатор типа позволяет процессу-получателю определить, к какому типу относится полученный узел, какой интерфейс он предоставляет, и соответственно, какой узел абстрактного синтаксического дерева необходимо создать для него. Идентификатор узла используется при последующих обращениях к процессу Rider: при обработке запроса этот идентификатор позволяет однозначно определить, какой узел послужил основой для узла AST.

Наличие промежуточного представления позволяет ослабить связность кода: ни одна из сторон не имеет информации о том, какое представление дерева используется на другой стороне. Так, на стороне Rider невозможно использовать PSI при обращении к абстрактным синтаксическим деревьям классов из библиотек, и приходится использовать другое представление данных, которое строится на основе метаданных из библиотек при помощи механизмов, уже реализованных в Rider, и такая замена происходит прозрачно для стороны, использующей EnvDTE. Это также позволяет не загружать в пользовательский процесс сборки, содержащие определение PSI, что уменьшает риск коллизий. Другим ключевым моментом является ленивость преобразований и неотделимость передачи дерева от его построения. Благодаря этому обе стороны не только никогда не строят деревья, используемые другой стороной, но и никогда не строят промежуточные деревья; передаются только отдельные узлы.

3.1.3. Делегирование в Rider

Получение информации о семантике кода тоже делается при помощи запроса из пользовательского процесса в процесс Rider. Каждый узел абстрактного синтаксического дерева хранит узел промежуточного представления, по которому он был построен. По узлу промежуточного представления всегда можно однозначно определить, какой узел

PSI послужил его основой. После выбора узла PSI вся логика делегируется инфраструктуре среды разработки, в которой реализованы аналогичные операции. Таким образом, среда разработки выступает в роли бэкенда для пользовательского кода: все нетривиальные операции производятся в ней, а в пользовательский процесс загружается только легковесный посредник.

3.2. Интеграция в Rider

Была реализована интеграция библиотеки в уже существующую подсистему поддержки языка T4. Исходный код этой подсистемы с поддержкой EnvDTE выложен на сервис GitHub по адресу https://github.com/JetBrains/ForTea/tree/net202-envdte. Все сборки из JetBrains.EnvDTE сделаны частью плагина к среде разработки и могут быть загружены в её рантайм. Это сделано для того, чтобы иметь возможность гарантировать, что на машине конечного пользователя эти сборки будут доступны, и код, опирающийся на них, можно будет исполнить.

3.2.1. Редактирование кода

При редактировании в T4-файлах распознаются типы из этих сборок и присутствует ряд интеллектуальных функций редактора для символов, специфичных для EnvDTE, например, автодополнение, подсветка синтаксиса, показ документации и подсказки типов. (рис. 4). Это реализовано при помощи стандартного для Rider механизма: метаданные из созданных в ходе данной работы сборок загружаются в процесс среды разработки и разбираются при помощи инструментов, уже использующихся в среде разработки; интеллектуальные механизмы предоставляются на основе этих метаданных. Благодаря этому функции редактора всегда соответствуют наиболее актуальной версии библиотеки.

Рис. 4: Пример интеллектуальных функций редактора для символов EnvDTE в файле на языке T4.

3.2.2. Кодогенерация, компиляция, исполнение

Для того, чтобы исполнить пользовательский код на языке T4, среда разработки транслирует его в C#, затем компилирует и исполняет получившийся код. Кодогенерация была изменена так, чтобы к пользовательскому коду добавлялись строчки, при необходимости инициализирующие все структуры, специфичные для данной реализации библиотеки, и устанавливали соединение с процессом бэкенда Rider. При компиляции сгенерированного кода на C# пути до сборок интерфейсов EnvDTE, являющихся частью плагина, добавляющего поддержку языка T4, передаются компилятору в числе остальных библиотек, необходимых для компиляции этого кода. При кодогенерации в код также добавляются строки, позволяющие виртуальной машине .NET найти и загрузить сборки из плагина в тот момент, когда пользовательский код впервые обратится к типам из них. При исполнении T4-файла процесс бэкенда Rider открывает порт, к которому пользовательский код может подключиться и отправить EnvDTE-специфичные запросы. Адрес этого порта затем передаётся дочернему процессу с пользовательским кодом в качестве переменной среды.

3.3. Ограничения

В предложенной реализации поддержаны основные функции, связанные с проектной моделью: с файлами и проектами; и основные функции, связанные с абстрактными синтаксическими деревьями. Все остальные функции в библиотеке не имеют содержательную реализацию и бросают исключение при вызове. Кроме того, созданная реализация поддерживает только функции чтения: любая модификация абстрактного синтаксического дерева или проектной модели вызовет исключение.

Заключение

В рамках данной работы были выполнены следующие задачи:

- была создана библиотека, имитирующая внешний программный интерфейс EnvDTE; в ней были реализованы основные функции, связанные с проектной моделью, и основные функции, связанные с абстрактными синтаксическими деревьями.
- был создан плагин к среде разработки Rider, позволяющий получать информацию, необходимую для реализации абстракций, аналогичных абстракциям из EnvDTE;
- была организована передача данных между библиотекой, загруженной в пользовательский процесс, и средой разработки;
- был написан код, добавляющий поддержку EnvDTE в одну из подсистем Rider при помощи созданной библиотеки.

Список литературы

[1] Akinshin Andrey. BenchmarkDotNet // article. — 2019. — URL: https://benchmarkdotnet.org/articles/overview.html (дата обращения: 13.10.2019).

[2] Apache. Apache Thrift // Web page. — 2017. — URL: https://thrift.apache.org/ (дата обращения: 27.11.2019).

[3] Balliauw Maarten. Rider front end plugin development // blog. — 2017. — URL: https://blog.jetbrains.com/dotnet/2017/02/07/rider-front-end-plugin-development/ (дата обращения: 28.10.2019).

[4] Ellis Matt. Project Rider — C# IDE // JetBrains .NET Tools Blog. — 2016. — URL: https://blog.jetbrains.com/dotnet/2016/01/13/project-rider-a-csharp-ide/ (дата обращения: 28.10.2019).

[5] Glick Dave. Announcing Scripty // blog. — 2016. — URL: https://daveaglick.com/posts/announcing-scripty (дата обращения: 13.10.2019).

[6] Google. Google gRPC // Web page. — 2019. — URL: https://www.grpc.io (дата обращения: 27.11.2019).

[7] JetBrains. Protocol extension // ReSharper DevGuide. — 2018. — URL: https://www.jetbrains.com/help/resharper/sdk/Products/Rider.html#protocol-extension.

[8] JetBrains. RD // GitHub. — 2019. — URL: https://github.com/JetBrains/rd (дата обращения: 27.10.2019).

[9] JetBrains. ReSharper // overview. — 2019. — URL: https://www.jetbrains.com/resharper/ (дата обращения: 28.10.2019).

[10] JetBrains. Rider // overview. — 2019. — URL: https://www.jetbrains.com/rider/ (дата обращения: 28.10.2019).

[11] Microsoft. Automation Model Overview // MSDN. — 2016. — URL: https://docs.microsoft.com/en-us/visualstudio/extensibility/internals/automation-model-overview (дата обращения: 06.10.2019).

[12] Microsoft. Getting data from Visual Studion // MSDN. — 2016. — URL: https://docs.microsoft.com/en-us/visualstudio/modeling/design-time-code-generation-by-using-t4-text-templates?view=vs-2019#getting-data-from-visual-studio (дата обращения: 23.11.2019).

[13] Osenkov Kiril. How to get DTE from Visual Studio process ID? // blog. — 2011. — URL: https://blogs.msdn.microsoft.com/kirillosenkov/2011/08/10/how-to-get-dte-from-visual-studio-process-id/ (дата обращения: 13.10.2019).

[14] Warren Genevieve. Launch Visual Studio using DTE // MSDN. — 2019. — URL: https://docs.microsoft.com/en-us/visualstudio/extensibility/launch-visual-studio-dte (дата обращения: 13.10.2019).