Правительство Российской Федерации
Федеральное государственное бюджетное образовательное учреждение
высшего профессионального образования
«Санкт-Петербургский государственный университет»

Кафедра системного программирования

Озерцов Александр Сергеевич

Визуализация кода встроенных языков в Microsoft Visual Studio

Курсовая работа

Научный руководитель:
ст.пр., магистр Григорьев С. В.

Санкт-Петербург
2015

Оглавление

Введение

Встроенные языки

При разработке сложных программ используется более одного языка программирования. В таких ситуациях говорят об основном языке и встроенном языках.

Встроенный язык — язык, команды которого выполняются из базового языка. Команды на таком языке собираются из строковых литералов в процессе работы основной - внешней - программы. Обычно это предметно-ориентированные языки (специализированные под конкретную область применения). Использование встроенного языка увеличивает функциональные возможности языка общего назначения при использовании второго в области специфичной для встроенного языка. Так как код встроенного языка в основном языке представляется компилятору в виде строкового выражения, то до момента исполнения невозможно проверить код на предмет наличия в нем лексических и синтаксических ошибок. Данную проблему разрешает статический анализ встроенных языков. Однако, в сложных программных системах строковые выражения, чаще всего формируются динамически. Примером может служить динамический запрос к базе данных, такой запрос является результатом конкатенации строк, притом конкатенация может происходить в условном операторе или цикле. Подобный код статически не анализируется. Запуск продукта без проверки не рекомендуется, так как ошибка может привести к непредсказуемым результатам. Кроме того, использование встроенных языков усложняет процесс разработки и отладки, так как порождается много различных вариантов строк, каждая из которых потенциально содержит ошибку. В таком случае проводится анализ всего множества выражений, которые могут быть получены.

Пример выполнения кода на JavaScript из кода, написанного на языке Java:

```
1 import javax.script.*;
2 public class InvokeScriptFunction {
3     public static void main(String[] args) {
4         ScriptEngineManager manager = new ScriptEngineManager();
5         ScriptEngine engine = manager.getEngineByName("JavaScript");
6         //JavaScript code in a String
7         String script = "function hello(name) {print('Hello, ' + name);}";
8         // evaluate script
9         engine.eval(script);
10     }
11 }
```

Проблема

В рамках исследовательского проекта YaccConstructor [1], [5] были реализованы алгоритмы лексического и синтаксического анализа динамически формируемых выражений [2]. Лексический анализ осуществляет токенизацию входного графа с фрагментами кода на ребрах; синтаксический анализ строит лес разбора цепочек из множества значений динамически формируемого выражения, а так же сообщает о синтаксически некорректных цепочках. Ввиду того, что строки формируются с помощью строковых операций, конкатенаций в циклах и условных операторах, то возможно большое количество вариантов и становится проблематично ориентироваться в токенизированном графе. Как правило, человек лучше воспринимает и обрабатывает информацию в систематизированном виде, поэтому удобно представлять результаты в графическом виде. Графическое представление позволяет наглядно представить все возможные пути формирования строковых выражений. Наиболее удобным представлением в данной ситуации является граф, так как позволяет выразить то, что выражения формируются с помощью конкатенации, циклов и позволяет переиспользовать фрагменты встроенного выражения.

1. Обзор

Ниже описаны существующие на данный момент решения, позволяющие осуществлять синтаксический и лексический разбор языков и представлять какую-либо информацию в виде графа.

Visual Studio CodeMap Технология Code Map в Visual Studio позволяет графически представить код, вложенный в окне редактора. Особенно эта возможность может пригодиться при изучении старого или незнакомого кода.
С точки зрения визуализации данный инструмент позволяет.

- Строить граф связности файлов проекта.
- Отражать все методы внутри каждой компоненты файла.
- Сворачивать/разворачивать область отображения.

Недостатком данного инструмента являются.

- Закрытый исходный код.
- Отсутствие возможности анализировать строковые выражения.

Ниже представлен результат работы данного инструмента (рис. 1).

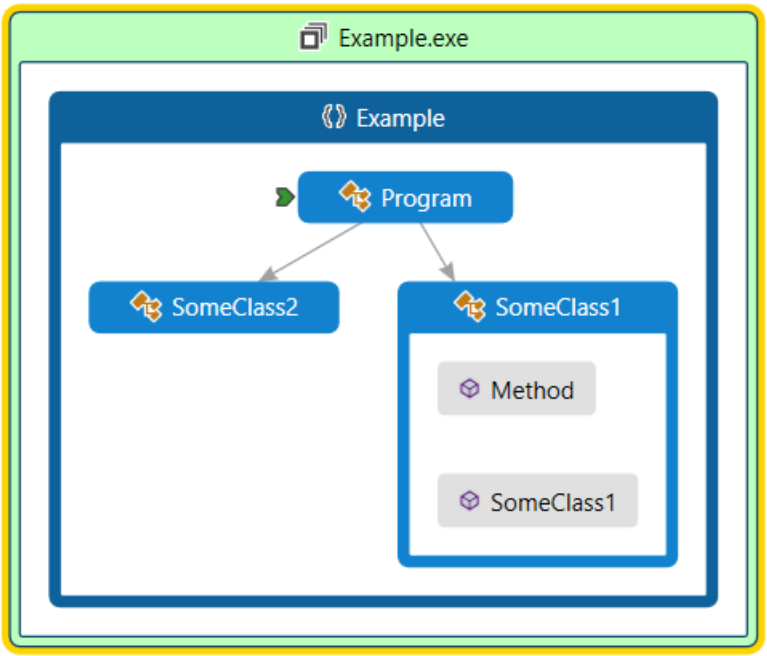

Рис. 1: Графическое представление кода в CodeMap

Python Tutor Данный веб продукт имеет возможность анализировать учебные программы, состоящие из одного файла, на языках Python, Java, JavaScript и наглядно изображать их выполнение. Предназначен для визуализации выполнения программ, написанных в собственной онлайн среде разработки.
С точки зрения визуализации данный инструмент позволяет.

- Отобразить пошаговое исполнение каждого метода, путем визуализации пошаговой работы интерпретатора.

Недостатком данного инструмента являются:

- Невозможность интеграции в среду разработки.
- Обязательное подключение к интернету.
- Невозможность других режимов синтаксического разбора, отличных от пошагового.

Ниже представлен результат работы данного инструмента (рис. 2).

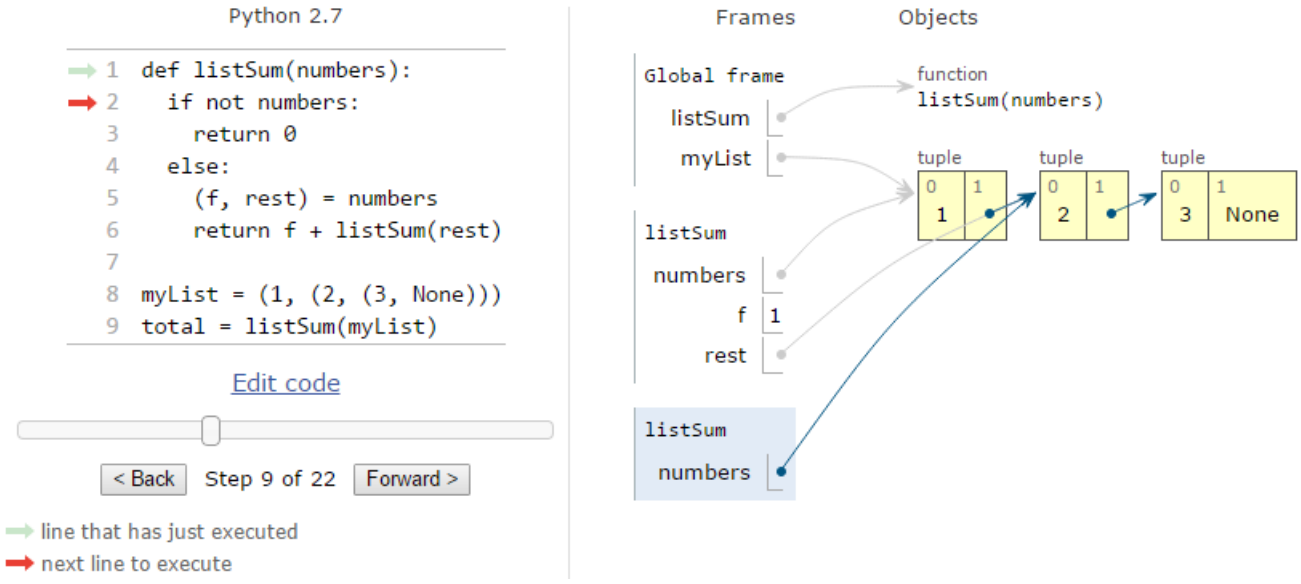

Рис. 2: Пример работы PythonTutor

Code Visual Editor Code Visual Editor - программа, способная анализировать исходный код и отображать его в виде блок-схемы. Программа анализирует языки программирования C, C++, Pascal, Perl, PHP, PL/SQL, POSIX shell, Power Script, T-SQL, Visual Basic/VBA, VB.NET, VBScript, and Qbasic/Basic.
Достоинством продукта является.

- Возможность анализировать выделенную часть кода.

Недостатком данной программы являются.

- Отсутствие интеграции в среду разработки.
- Закрытый исходный код.
- Для каждого запуска требуется указывать параметры работы систем;
- Отсутствие возможности анализировать строковые выражения.

Ниже представлен результат работы данного инструмента (рис. 3).

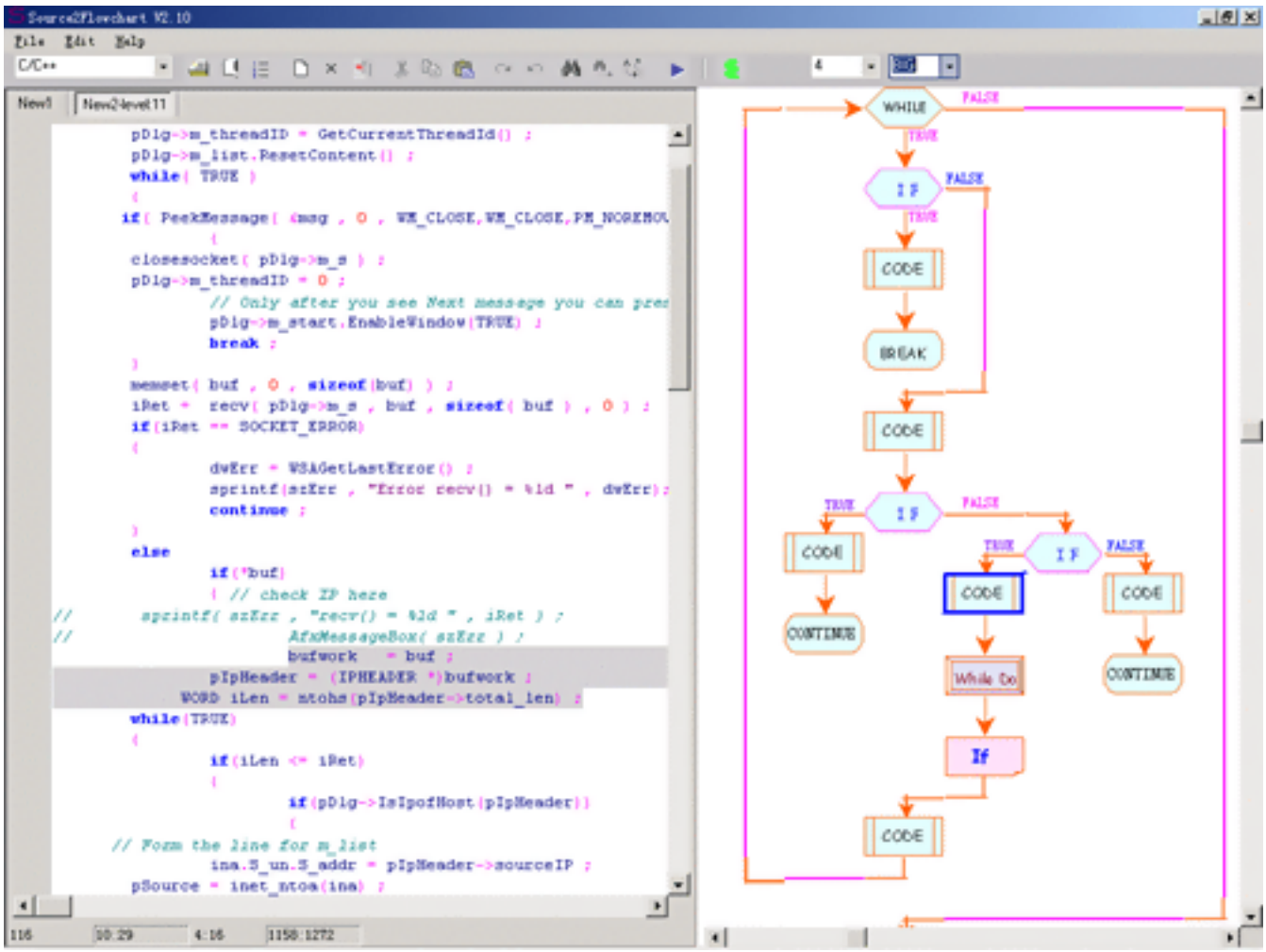

Рис. 3: Пример работы Code Visual Editor

Code to FlowChart Code to FlowChart - продукт, генерирующий код в блок схему. Это помогает пользователям понять сложную структуру программы с помощью визуальных диаграмм. Code to FlowChart поддерживает языки программирования C, C++, VC++ and Pascal/Delphi.
Достоинством продукта является.

- Подсветка выделенного кода в построенной блок-схеме.
- Цветовое выделение различных языковых конструкций.

Недостатком данной программы являются.

- Отсутствие интеграции со средой разработки.
- Закрытый исходный код.
- Отсутствие возможности анализировать строковые выражения.

Ниже представлен результат работы данного инструмента (рис. 4).

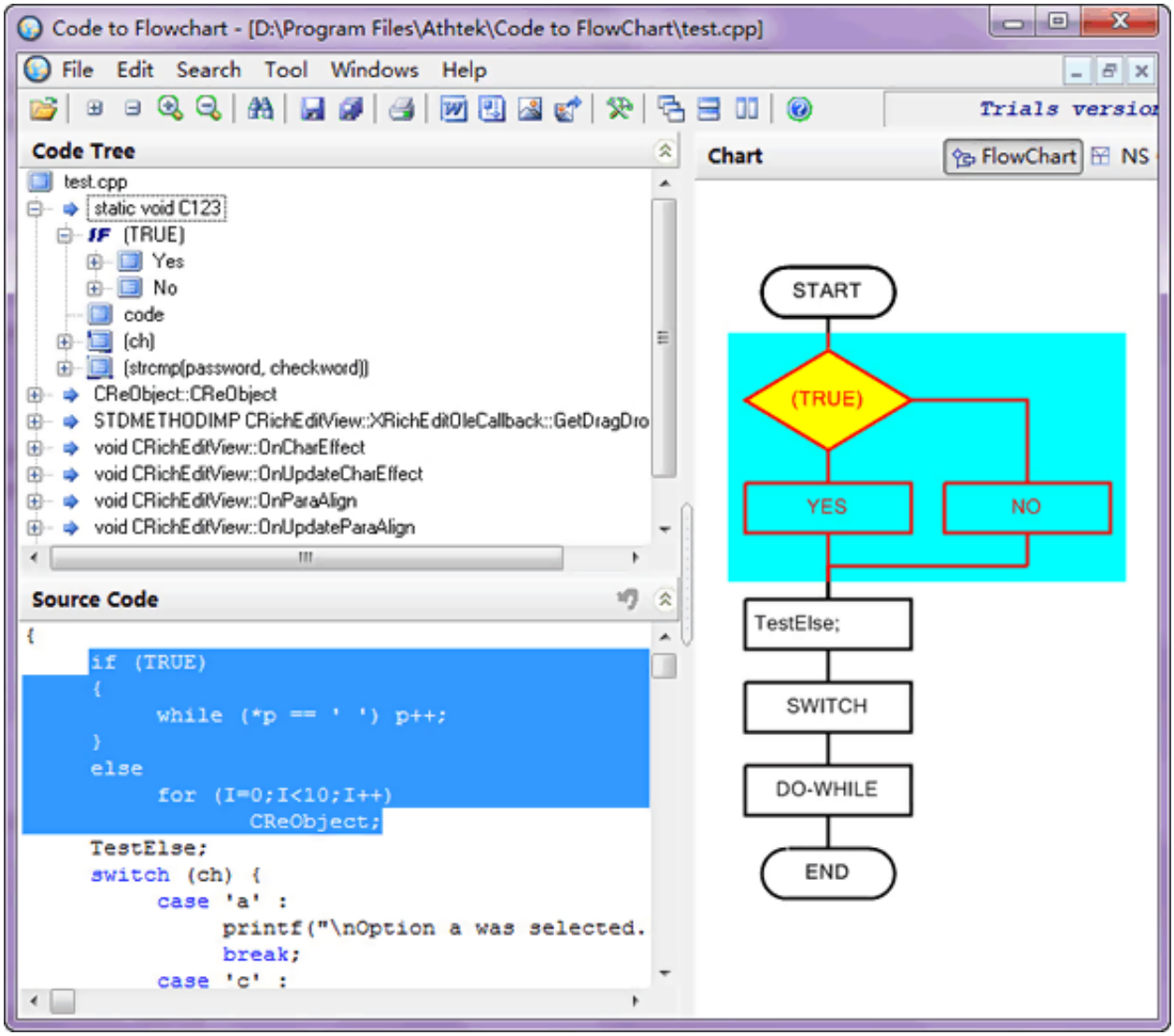

Рис. 4: Пример работы CodeToFlowChart

2. Постановка задачи

Целью данной работы является разработка компоненты, предназначенной для графического представления данных в среде разработки Microsoft Visual Studio с последующей интеграцией в проект YaccConstructor. Данный модуль позволяет визуализировать структуру динамически формируемого выражения, что позволяет упростить поиск ошибок и поддержку кода. Для достижения поставленной цели были сформулированы следующие задачи.

- Изучить библиотеку для визуализации графов GraphX [4].
- Изучить плагин для Visual Studio, проводящий статический анализ кода Resharper.
- Создать графическую компоненту, встроенную в среду разработки Microsoft Visual Studio.
- Добавить компоненту в YC.ReSharper.AbstractAnalysis plugin [3] с возможностью отрисовки графов в среде Microsoft Visual Studio.
- Добавить в YC.ReSharper.AbstractAnalysis plugin возможность запускать визуализатор непосредственно в среде разработки Microsoft Visual Studio.
- Добавить в YC.ReSharper.AbstractAnalysis plugin возможность просмотра требуемой информации в графическом виде, путем вывода графа.
- Ввиду того, что в коде может формироваться несколько строковых выражений, следует добавить в YC.ReSharper.AbstractAnalysis plugin возможность переключаться между графами различных формируемых выражений.

Ниже приведен пример кода и ожидаемого графа:

```
1 private void Go(int a, int b, int c)
2 {
3     String s = ”select ”;
4     if (a > b) {
5         s += ”x from y”;
6     }
7     else {
8         if (c > b) {
9             s += ”y from x”;
10        }
11        else {
12            s += ”y from z”;
13        }
```

```
14      }
15      Program.ExecuteImmediate(s);
16  }
```

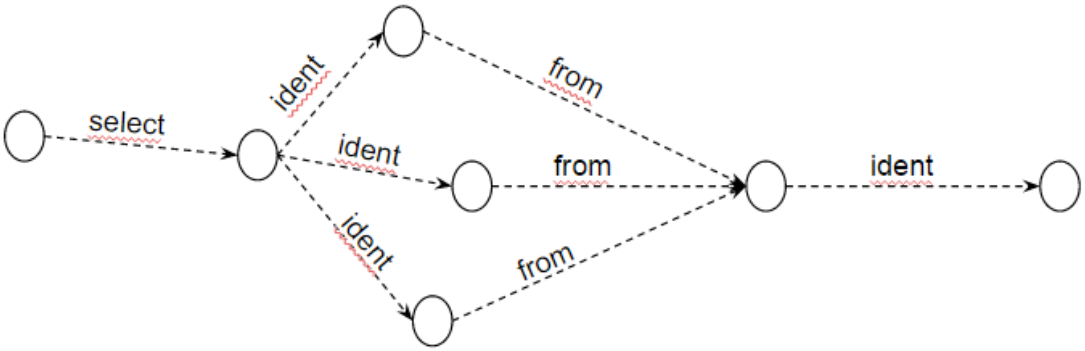

Рис. 5: Ожидаемый граф

3. Основная часть

В данном разделе будут рассмотрены подходы, примененные в разработке, и решения принятые в ходе работ.

На данном этапе компоненту можно разбить на три основные части, а именно:

- Подсистема создания диалогового окна. Данная подсистема отвечает за создание диалогового окна, встроенного в среду разработки Microsoft Visual Studio и логику взаимодействия с ним. Она основана на библиотеке ReSharper.SDK. Эта библиотека позволяет создавать собственные плагины для среды разработки Microsoft Visual Studio с сохранением дизайна данной среды разработки.

- Подсистема внутреннего представления графов. Пакет предназначен для удобного хранения всех составляющих компонент графов внутри плагина. Подсистема основана на библиотеках GraphX.Net и QuickGraph, имеющих соответственные классы для описания вершин, ребер и логической компоненты графов.

- Подсистема вывода графов в диалоговое окно. В её ответственность входит преобразование внутреннего представления графа в графическое с выводом в диалоговое окно. Данная подсистема основана на библиотеке GraphX.Net. Библиотека содержит набор алгоритмов визуализации графов, позволяет гибко конфигурировать устройство и внешний вид графа.

Для упрощения понимания взаимодействия компонент ниже изображена архитектура компоненты:

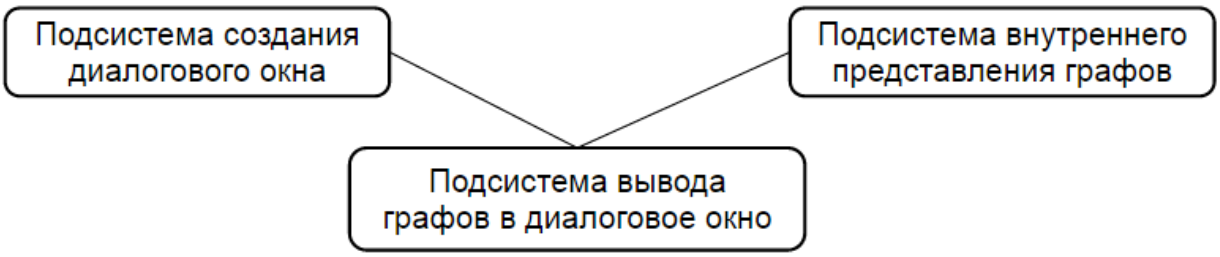

Рис. 6: Общая архитектура компоненты

В процессе создания компоненты была обнаружена проблема хранения графов. Изначально планировалось хранить графы внутри диалогового окна, но было обнаружено, что данная подсистема начинает работу после открытия окна, однако граф может быть сгенерирован до этого, в результате часть графов может быть утеряна. Было решено завести отдельное хранилище для графов, чтобы компонента, независимо от времени запуска уже могла предоставить разобранные выражения в виде визуализированных графов и никакие из них не были потеряны. Для отображения нескольких графов в главном окне компоненты был использован класс TabControl, который содержит в себе класс Tab из библиотеки System.Windows.Forms.Control. Такая организация позволяет переключаться между графами, а при появлении нового добавить его. Класс Tab хранит внутри себя граф и выводит его графическое представление. Также граф реагирует на действия пользователя над графом, а именно при наведении указателя типа “мышь” на одно из ребер оно выделяется цветом.

Ввиду того, что конечного пользователя интересует не только граф выражения, но и возможность наблюдать в графе этапы генерации строкового выражения, для работы с графом из библиотеки GraphX.Net были использованы и доработаны классы Edge, Vertex, LogicCore, GraphArea, Graph для хранения любой текстовой информации о выражении на ребрах и вершинах графа и возможности визуализировать его. Для представления графов в проекте YaccConstructor используется класс из библиотеки QuickGraph, а GraphX.Net использует другой класс библиотеки QuickGraph, поэтому для представления графов был реализован конвертер графов.

Главное окно компоненты реализовано с помощью библиотеки Rasharper.SDK. Эта библиотека позволяет создать элемент, полностью интегрированный в среду разработки Microsoft Visual Studio. Для реализации главного окна были использованы классы Descriptor, Registrar и реализован интерфейс IActionHandler. Класс Descriptor служит для описания нового объекта среды разработки. Реализация интерфейса отвечает за создание событий, когда пользователей вызывает компоненту, а класс WindowRegistrar обрабатывает эти события.

Для более удобного представления системы ниже представлена диаграмма классов:

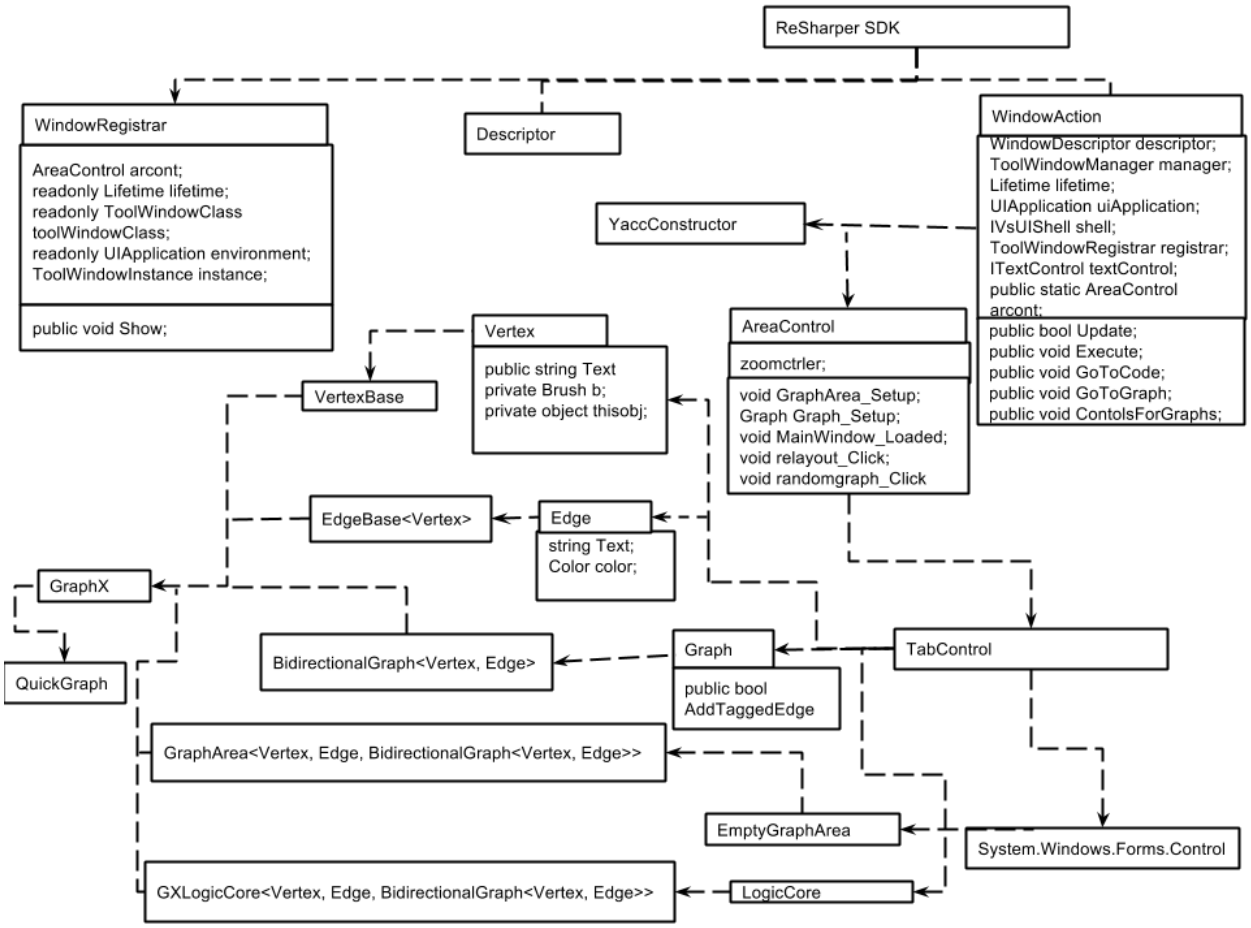

Рис. 7: Диаграмма классов

По окончанию работы была проведена апробация плагина. Пример исходного кода и результаты можно увидеть ниже:

```
1 private void Go(int a, int b, int c)
2 {
3     Program.ExecuteImmediate(”select x from select x from select x from select
4                 x from select x from select x from y”);
5     Program.Eval(”2 * (3 + 5)”);
6     String s = ”select ”;
7     if (a > b) {
8         s += ”x from y”;
9     }
10     else {
11         if (c > b) {
12             s += ”y from x”;
13         }
14         else {
15             s += ”y from z”;
```

```
16          }
17      }
18      Program.ExecuteImmediate(s);
19      String stmt = ”drop procedure”;
20      if (c == 1) {
21          Program.ExecuteImmediate(stmt + ”prc2”);
22      }
23      else {
24          Program.ExecuteImmediate(stmt + ”prc1”);
25      }
26  }
```

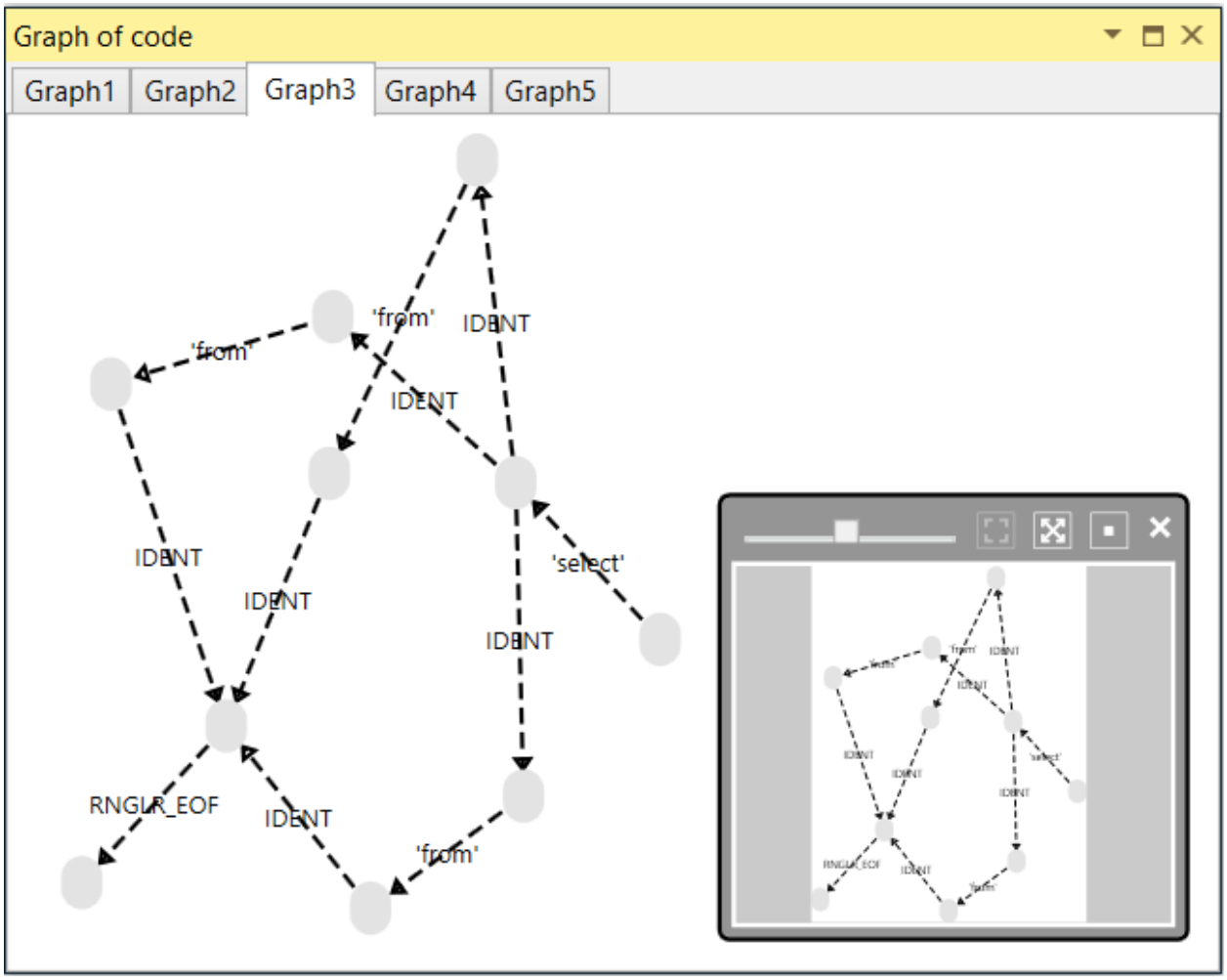

Рис. 8: Полученный граф

Заключение

Результаты

В результате работы была доработана библиотека GraphX для хранения требуемой информации на ребрах и вершинах графа, перехвата действий пользователя над графом. Была доработана компонента для передачи графа из анализатора в визуализатор. Доработан YC.ReSharper.AbstractAnalysis plugin для хранения требуемых графов, полученных в результате исполнения нескольких строковых выражений независимо от запуска визуализатора. Получившаяся система в результате работы позволяет следующее:

- Запуск визуализатора непосредственно в среде разработки Microsoft Visual Studio.
- Визуализация графов, полученных в результате работы алгоритмов лексического и синтаксического анализа динамически формируемых выражений.
- Возможность переключения между графами разных формируемых выражений.
- Возможность выделить рассматриваемое ребро графа.
- Возможность масштабирования графов.

Исходный код проекта YaccConstructor можно найти на сайте
https://github.com/YaccConstructor/YaccConstructor

Автор принимал участие под учетной записью m13oas.

Дальнейшее развитие

Дальнейшим развитием компоненты является поддержка динамического изменения в окне количества графов. Также развитие взаимодействия с пользователем, а именно возможность навигации из графа в код и обратно из кода в граф. Не менее важно улучшение графического представления, путем добавления выделения цветом ошибочных путей, улучшение алгоритмов автоматической раскладки и выбор алгоритма раскладки графа непосредственно в компоненте. Добавление возможности сохранения графа в файл.

Список литературы

[1] Github IO. Проект YaccConstructor // Основная страница проекта. — 2012. — URL: http://yaccconstructor.github.io/YaccConstructor/index.html.

[2] Grigorev Semen Kirilenko Iakov. GLR-based abstract parsing. Proceedings of the 9th Central & Eastern European Software Engineering Conference in Russia. — 2013. — ACM : http://dl.acm.org/citation.cfm?id=2556616.

[3] Mavchun Semen Grigorev Ekaterina Verbitskaia Andrei Ivanov Marina Polubelova Ekaterina. String-embedded language support in integrated development environment // Proceedings of the 10th Central and Eastern European Software Engineering Conference in Russia. — 2014. — URL: http://dl.acm.org/citation.cfm?id=2687233.2687247.

[4] Panthernet. Проект GraphX // Основная страница проекта. — 2011. — URL: http://ntv.spbstu.ru/telecom/article/T3.174.2013_11/.

[5] Кириленко Я.А. Григорьев С.В. Авдюхин Д.А. Разработка синтаксических анализаторов в проектах по автоматизированному реинжинирингу информационных системг // Конференция Microsoft. — 2013. — URL: http://yaccconstructor.github.io/YaccConstructor/index.html.