САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Математико-механический факультет

Кафедра системного программирования

Мавчун Екатерина Валерьевна

Архитектура для поддержки языковых расширений в абстрактном анализе

Курсовая работа

Научный руководитель:
магистр информационных технологий, ст.преп. Григорьев С. В.

Санкт-Петербург
2014

Оглавление

Введение

В настоящее время широко используется выполнение динамически сформированных символьных строк. Ошибки в таком коде можно обнаружить только в момент выполнения программы. Из подстрок могут собираться, например, запросы на SQL или части HTML страниц. В таком случае SQL (или HTML) является встроенным языком.

Встроенные языки — это языки, выражения на которых собираются из строк во время выполнения основной программы. Наиболее распространённый пример — встроенный в Java, C#, C++ SQL. Предположим, есть приложение на высокоуровневом языке, которое должно работать с базой данных. Во встроенном (статическом) SQL операторы встраиваются непосредственно в исходный текст программы на базовом языке:

```
...
IF @a = @b
    SET @reqField = '#column1'
ELSE
    SET @reqField = '#column2'
SET sqlQuery = 'SELECT ' + @reqField + ' FROM table1 WHERE column3 IS 5'
EXECUTE (sqlQuery)
...
```

В данном примере для переменной sqlQuery статически вычислимы два значения:
1. 'SELECT #column1 FROM table1 WHERE column3 IS 5';
2. 'SELECT #column2 FROM table1 WHERE column3 IS 5';

Динамический SQL подразумевает формирование в процессе выполнения программы кода SQL-операторов и дальнейшее их выполнение. Также существую другие встроенные языки и предметно-ориентированные языки (DSL): JSON, XML, HTML, JavaScript и др.

Работа со встроенными языками включает в себя подсветку синтаксиса, поиск ошибок, трансформацию. Чтобы работать со встроенными языками, нужно уметь их анализировать, для этого существует абстрактный анализ [7] [13] — анализ встроенных языков.

Работу со встроенными языками хочется поддерживать в интегрированной среде разработки (IDE), кроме того, удобно было бы поддерживать разные встроенные языки. Такие возможности среды могут быть полезны пользователям, которые хотят просто использовать новые встроенные языки, или разработчикам, которые хотят создавать новые языковые расширения (под языковыми расширениями подразумеваются модули, позволяющие поддерживать новые встроенные языки).

Существующие инструменты для работы со встроенными языками, например, такие как Alvor [3] [2], позволяют работать со встроенным языком, но только с одним. Можно получить поддержку других встроенных языков, изменяя исходный код, но это довольно трудоёмкая задача. В других существующих инструментах, таких как Java String Analyzer (JSA) [6] [1] или PHP String Analyzer [11] [9], довольно просто можно получить поддержку встроенных языков, но они предназначены для выполнения различных проверок динамически формируемых строк (то есть для статического анализа).

То есть, существующие инструменты в большинстве своём реализуют поддержку какого-то одного конкретного встроенного языка. В эти инструменты можно вручную добавить поддержку других языков, но намного удобнее было бы получить поддержку встроенного языка без изменений в исходном коде.

В рамках исследовательского проекта YaccConstructor [17] ведётся разработка инструментария для абстрактного анализа. Планируется создать плагин для ReSharper [12], позволяющий поддерживать работу с разными встроенными языками в IDE. ReSharper - это плагин к Microsoft Visual Studio [8], который проводит синтаксический анализ кода. Планируется использовать YaccConstructor — инструмент для создания парсеров и обработки грамматик. В данной работе рассмотрена как архитектура для поддержки разных встроенных языков в плагине к ReSharper, так и особенности её реализации.

1. Постановка задачи

На кафедре ведётся разработка инструментария для абстрактного анализа: основная цель — создать плагин к ReSharper, который позволит работать со встроенными языками.

Главной целью данной работы является разработка и реализация инфраструктуры, позволяющей поддерживать различные встроенные языки в плагине к ReSharper после их описания, а именно, после описания грамматики и лексической спецификации.

Для этого необходимо:

- разработать и реализовать архитектуру для поддержки новых встроенных языков в плагине к ReSharper
- реализовать парсер JSON [5] (для создания модуля, позволяющего поддерживать встроенный JSON в плагине к ReSharper)
- провести апробацию на примере интеграции в плагин существующего парсера T-SQL и реализованного парсера JSON

2. Обзор существующих решений

При разработке различных программ часто возникает необходимость использования нескольких языков программирования. В этом случае лучше иметь возможность узнавать об ошибках до выполнения программы. То есть необходим инструмент для абстрактного анализа, который позволяет поддерживать различные встроенные языки.
Рассмотрим инструменты, позволяющие поддерживать встроенные языки:

- Alvor — плагин для Eclipse для статической проверки встроенного в Java SQL. Alvor может быть использован как для однократного анализа всей программы, так и для инкрементального анализатора, работающего в процессе написания кода. Найденные в коде SQL-запросы проверяются на соответствие SQL-грамматике. Так же SQL-запросы могут проверяться исполнением в указанной тестовой базе.
  - Достоинства
    * Поддерживает абстрактный анализ встроенного в Java SQL
  - Недостатки
    * Поддержка других встроенных языков возможно путём изменения исходного кода
- PHP String Analyzer (PHPSA) — статический анализатор динамических выражений, порождаемых программами на PHP. PHP String Analyzer аппроксимирует значения таких строк некоторой контекстно-свободной грамматикой.
  - Достоинства
    * Достаточно легко можно получить поддержку нового встроенного языка
  - Недостатки
    * Поддерживает только поиск ошибок
- Java String Analyzer (JSA) — инструмент для анализа формирования строк и строковых операций в программах на Java. Для каждого строкового выражения Java String Analyzer строит конечный автомат, представляющий приближённое значение всех значений этого выражения, которые могут быть получены во время выполнения.
  - Достоинства
    * Достаточно легко можно получить поддержку нового встроенного языка

- Недостатки
    * Решает другие задачи (только поиск ошибок)

Таким образом, существующие инструменты для работы со встроенными языками либо реализуют поддержку какого-то одного конкретного языка, либо не поддерживают требуемую функциональность. Ни один из существующих инструментов не позволяет по описанной грамматике и лексической спецификации получить поддержку встроенного языка.

3. Реализация

Разрабатываемый плагин к ReSharper позволяет поддерживать разные встроенные языки в интегрированную среду разработки (IDE). Для реализации плагина используется функциональный язык программирования — F# [14] [4].

Предположим, у пользователя есть программа на C# и он хочет в своей программе использовать встроенный T-SQL. Для пользователя наиболее удобным способом было бы подключить DLL (Dinamic Link Library — динамически подгружаемая библиотека) к IDE в виде плагина, которая бы уже имела возможность работать со встроенным T-SQL. Такми образом, любой проект, открытый в IDE будет иметь поддержку встроенного языка.

Рис. 1: Пример использования встроенного T-SQL пользователем

В приведённом выше примере предполагается, что пользователю была предоставлена библиотека T-SQL.dll, которая содержит всю необходимую функциональность для работы со встроенным T-SQL. Рассмотрим другой пример: разработчик хочет создать языковое расширение для T-SQL, то есть создать T-SQL.dll. Для этого разработчику необходимо описать язык T-SQL, а именно: описать грамматику и лексическую спецификацию T-SQL. Грамматика описывается в файле TSQL.yrd на языке описания трансляций Yard [15] [18] [16]. Лексическая спецификация описывается в файле Lexer.fsl. В том случае, если у разработчика нет парсера для T-SQL, то он создаётся с помощью YaccConstructor. Далее создаётся динамически подключаемая библиотека T-SQL.dll, загрузив её, пользователь получит поддержку встроенного T-SQL в IDE. Последним шагом является разметка кода: IDE должна понимать где и какой встроенный язык пользователь хочет использовать.

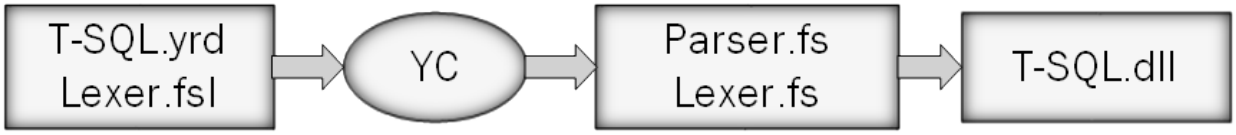

Рис. 2: Пример использования встроенного T-SQL пользователем-разработчиком

Таким образом, чтобы обеспечить поддержку разных встроенных языков в среде разработки нужно:

- предоставить механизм для простой реализации поддержки нового встроенного языка и интеграции его в систему
- дать пользователю возможность указывать в коде строки, соответствующие конкретным встроенным языкам

То есть необходимо уметь динамически загружать модули, которые реализуют поддержку конкретных встроенных языков, узнавать, то место в коде, в котором пользователь хочет получить поддержку встроенного языка, и какой именно язык он хочет поддерживать.

Возможность среды поддерживать новые встроенные языки может быть полезна либо пользователям, которые хотят использовать встроенные языки в своих приложениях, либо пользователям-разработчикам, которым требуется разработать новое языковое расширение с минимальными усилиями, то есть, описав грамматику и лексическую спецификацию языка.

3.1. Архитектура

Зависимость функциональных модулей выглядит следующим образом:

- ReSharper является плагином к Microsoft Visual Studio
- Plugin — это разрабатываемый плагин к ReSharper
- TSQL, JSON — модули, реализующие поддержку встроенных языков. Каждый из таких модулей реализует общий интерфейс, позволяющий унифицировано подключать поддержку нового языка
- Common содержит интерфейс, который реализует парсер, и атрибут InjectedLanguage с полем languageName, позволяющий определять, какой язык пользователь хочет поддерживать в своём приложении. Например, если код помечен атрибутом [InjectedLanguage (“SQL”)], то соответствующие строковые выражения являются кодом на языке SQL.
- Core обеспечивает связь плагина с модулями, реализующими встроенные языки

3.2. Динамическая загрузка модулей

Чтобы предоставить механизм для простой реализации поддержки нового встроенного языка и интеграции его в систему, необходима динамическая загрузка модулей, реализующих поддержку встроенных языков, которая осуществляется с помощью средства для создания расширяемых приложений Mono.Addins [10].

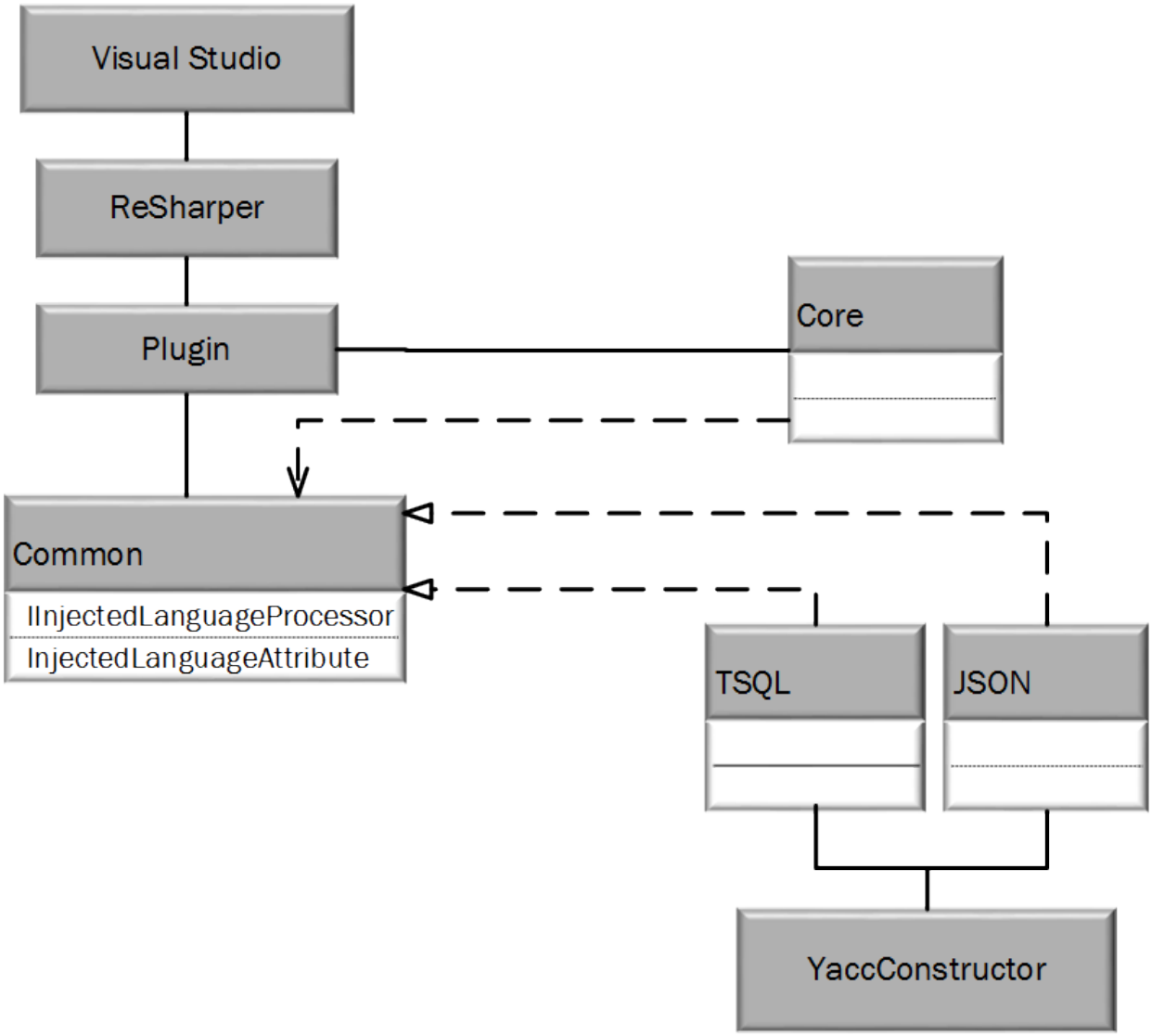

Рис. 3: Зависимость модулей

3.3. Разметка атрибутом

Чтобы дать пользователю возможность указывать в коде программы строки, соответствующие конкретным встроенным языкам, создан атрибут InjectedLanguage с полем languageName типа String (languageName - название языка). То есть, если код помечен атрибутом [InjectedLanguage (“SQL”)], то в данном коде необходимо поддерживать язык SQL. По значению поля languageName происходит поиск языкового расширения, соответствующего языку, название которого пользователь указал в атрибуте.

Пример использования атрибута InjectedLanguage:

...
var key = “Name”;
var value1 = “Ivan”;

```
var value2 = “Vasily”;
var obj = “ ” + key + “ : ”;
if (condition)
{
    obj = obj + value1;
}
else
{
    obj = obj + value2;
}
obj = obj + “ ”;
[InjectedLanguage (“JSON”)]
PrintJS(obj)
...
```

3.4. Интерфейс

Пользователю, разрабатывающему языковые расширения, необходимо описать грамматику и лексическую спецификацию языка, поддержку которого он хочет получить в программе. После описания языка с помощью YaccConstructor создаётся парсер и лексер. Чтобы однотипно подгружать модули на стороне YaccConstructor, парсер должен реализовывать следующий интерфейс:

- название языка, хранящееся в поле типа string — languageName
- метод, преобразующий токен в число — TokenToNumber
- метод, преобразующий число в строку — NumToString
- метод, “упаковывающий” токен — TokenData
- метод, запускающий лексический анализ — Tokenize
- метод, запускающий синтаксический анализ — Parse

Если пользователь хочет поддерживать в своей программе новые встроенные языки, ему необходимо выполнить следующие действия:

- установить плагин для ReSharper
- установить языковое расширение
- разметить код атрибутами

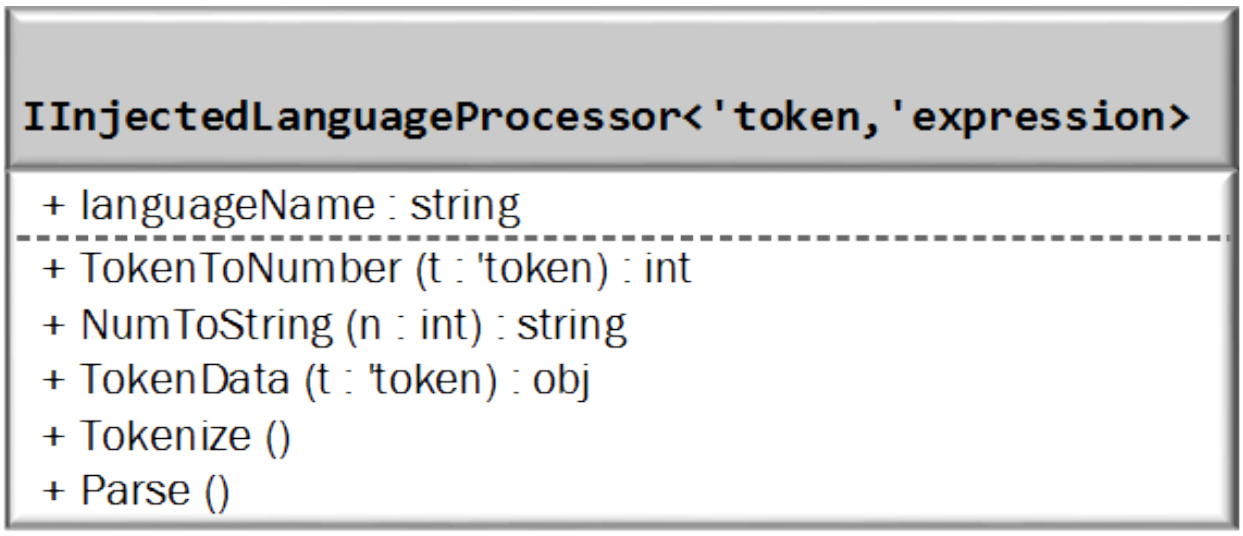

Рис. 4: Интерфейс, который должны реализовывать парсеры

После выполнения вышеперечисленных инструкций пользователь получит поддержку встроенного языка в IDE.

Если пользователь-разработчик хочет создать новое языковое расширение, то ему необходимо:

- установить плагин для ReSharper
- описать грамматику языка (yrd-файл)
- описать лексическую спецификацию языка (fsl-файл)
- сгенерировать лексер и парсер по описанным выше файлам

В результате проделанных действий разработчик получит модуль, позволяющий поддерживать новый встроенный язык.

4. Апробация

Для апробации разработанной архитектуры был реализован парсер JSON и интегрирован в плагин для ReSharper.

4.1. Парсер JSON

JSON (JavaScript Object Notation) — это текстовый формат обмена данными. JSON может быть использован, например, для сериализации сложных структур.
JSON представляет собой одну из двух структур:

- Набор пар “ключ : значение”. В различных языках программирования набор пар “ключ : значение” реализовано как объект, запись, структура, хэш-таблица или ассоциативный массив. Ключ описывается строкой, значение — это любая форма
- Упорядоченный набор значений. В большинстве языках программирования упорядоченный набор значений реализован как массив, вектор или список

В качестве значений в JSON используются структуры:

- Объект — это заключённое в фигурные скобки “ ”, неупорядоченное множество пар “ключ : значение”. Пары отделяются друг от друга запятыми
- Массив — это заключённое в квадратные скобки “[ ]” множество значений. Значения разделяются запятыми
- Значение — это строка, число, объект, массив или литерал: true, false или null. То есть, структуры могут быть вложены друг в друга
- Строка — это заключённое в двойные кавычки, упорядоченное множество из символов Unicode

В файле Lexer.fsl, по которому будет сгенерирован лексер, с помощью регулярных выражений описаны все значения, которые могут встретиться в JSON, такие как символ, строка, целое число, экспонента, шестнадцатиричные числа и т.д. В файле JSON.yrd описан вид структур:

...
value: string | number | object | array | ’true’ | ’false’ | ’null’
array: ’[’ list<value ’,’> ’]’
pair: string ’:’ value
...

Далее с помощью YaccConstructor генерируются лексер и парсер для JSON. Таким образом мы получили необходимую функциональность для поддержки встроенного JSON.

Таким образом, реализован унифицированный механизм для простой поддержки нового встроенного языка и интеграции его в систему. Так как механизм уже реализован, то динамическая загрузка модулей для поддержки JSON и T-SQL является технической задачей, которая будет проделана в ближайшее время. Реализованный механизм был протестирован на примере проекта на C, в который был встроен JSON и T-SQL, и тем самым, была получена поддержка новых встроенных языков в среде разработки.

Заключение

Результаты:

- разработана архитектура для поддержки новых встроенных языков в плагине к ReSharper
- реализован парсер JSON
- проведена апробация на примере интеграции в плагин существующего парсера T-SQL и реализованного парсера JSON

Реализованный парсер JSON выложен на GitHub, учётная запись emavchun:
`https://github.com/emavchun/YC.Abstract.JSON.git`
Разрабатываемый плагин для ReSharper, учётная запись emavchun@gmail.com:
`https://code.google.com/p/recursive-ascent/`

В составе исследовательской группы была написана статья “Инструментальная поддержка встроенных языков в интегрированных средах разработки”, в которой в том числе были представлены результаты данной работы. Статья принята на семинар ”Наукоемкое программное обеспечение (НПО)”, который пройдет в рамках девятой международной Ершовской конференции по информатике (PSI’14).
Дальнейшее развитие:

- оформить реализованную архитектуру в виде плагина для возможности дальнейшего удобного использования
- завершить апробацию на примере интеграции TSQL в плагин

Список литературы

[1] A. Christensen A. Møller, Schwartzbach M. Precise analysis of string expressions. — June 2004.

[2] Aivar Annamaa Andrey Breslav Jevgeni Kabanov, Vene Varmo. An interactive tool for analyzing embedded sql queries. — С. 131–138. — Vol. 6461 of Lecture Notes in Computer Science.

[3] Alvor. — URL: http://code.google.com/p/alvor/.

[4] F# Language Specification. — URL: http://fsharp.org/specs/language-spec/.

[5] JSON. — URL: http://www.json.org/.

[6] Java String Analyzer. — URL: http://www.brics.dk/JSA/.

[7] Kyung-Goo Doh Hyunha Kim David A. Schmidt. Abstract LR-parsing. — 2011. — С. 90–109.

[8] Microsoft Visual Studio. — URL: http://www.visualstudio.com.

[9] Minamide Y. Static Approximation of Dynamically Generated Web Pages. — 2005. — С. 432–441.

[10] Mono.Addins. — URL: http://www.mono-project.com/Mono.Addins.

[11] PHP String Analyzer. — URL: http://www.score.is.tsukuba.ac.jp/~minamide/phpsa/.

[12] ReSharper. — URL: http://www.jetbrains.com/resharper/.

[13] Semen Grigorev Iakov Kirilenko. GLR-based abstract parsing. — 2013.

[14] Syme Don, Granicz Adam, Cisternino Antonio. Expert F#. — Apress, 2007. — ISBN: 978-1-4302-0285-1. — URL: http://dx.doi.org/10.1007/978-1-4302-0285-1_19.

[15] Yard. — URL: https://code.google.com/p/recursive-ascent/wiki/YARD_frontend.

[16] Алексеевич Авдюхин Дмитрий. Язык описания трансляции для средств реинжиниринга информационных систем. — 2013. — Дипломная работа. URL: http://se.math.spbu.ru/SE/diploma/2013.

[17] Кириленко Я.А. Григорьев С. В. Авдюхин Д. А. Разработка синтаксических анализаторов в проектах по автоматизированному реинжинирингу информационных систем. — 2013. — С. 94–98. — URL: http://elibrary.ru/contents.asp?issueid=1127792&selid=19142331.

[18] Сергеевич Чемоданов Илья. Генератор синтаксических анализаторов для решения задач автоматизированного реинжиниринга программ. — 2007. — Дипломная работа. URL: https://code.google.com/p/recursive-ascent/downloads/detail?name=IlyaChemodanov_Yard.pdf#makechanges.