Custom Roslyn Tool for Real-Time Static Code Analysis

MASTER'S THESIS

Zuzana Dankovčíková

Brno, Spring 2017

MASARYK UNIVERSITY
FACULTY OF INFORMATICS

Custom Roslyn Tool for Real-Time Static Code Analysis

MASTER'S THESIS

Zuzana Dankovčíková

Brno, Spring 2017

This is where a copy of the official signed thesis assignment and a copy of the Statement of an Author is located in the printed version of the document.

Declaration

Hereby I declare that this paper is my original authorial work, which I have worked out on my own. All sources, references, and literature used or excerpted during elaboration of this work are properly cited and listed in complete reference to the due source.

Zuzana Dankovčíková

Advisor: Bruno Rossi PhD

Acknowledgement

TODO: This is the acknowledgement…

Abstract

TODO: This is the abstract ...

Keywords

Contents

List of Tables

List of Figures

1 Introduction

TODO...

Ideas:

What is code quality, why is it important, tool that support it.. compilers, diversion ... aaaand here comes Roslyn which provides compiler as a platform.

In the .NET world, the compiler used to be a black box that given the file paths to the source text, produced an executable (see Figure 3.1). In order to do that, compiler has to collect large amount of information about the code it is processing. This knowledge, however, was unavailable to anyone but the compiler itself and it was immediately forgotten once the translated output was produced [**roslyn-overview-github**].

Why is this an issue when for decades this black-boxes served us well? Programmers are increasingly becoming reliant upon the powerful integrated development environments (IDEs). Features like IntelliSense, intelligent rename, refactoring or "Find all references" are key to developers' productivity; and even more so in an enterprise-size systems.

This gave a rise to number of tools that analyze the code for common issues and are able to suggest a refactoring. The problem is that that such tool needs to parse the code first in order to be able to understand and analyze it. As a result companies need to invest fair amount of resources to duplicate the logic that the .NET compiler already possesses. Not only is it possible that the compiler and the tool may disagree on some specific piece of code, but with every new version of C# the tool needs to be updated to handle new language features[**dot-net-development-using-the-compiler-api**].

With roslyn.. etc. etc. .. API for analysis.. use in companies for custom analyzers... etc. etc.... https://github.com/dotnet/roslyn/wiki/Roslyn Overview – motivation Make sure to stress out that ".NET Compiler Platform" and "Roslyn" names will be used interchangably as it is in Roslyn Succinctly on page 11.

2 Code Quality and Static Code Analysis

TODO

3 .NET Compiler Platform

As per [**dragon-book**], compiler is a program that can read a program in a *source* language and translate it into a semantically equivalent program in *target* language while reporting any errors detected in the translation process.

In the .NET world, the compiler used to be a black box that given the file paths to the source text, produced an executable (see Figure 3.1). This perception was changed in 2015 when Microsoft introduced the .NET Compiler Platform (commonly referred to as "Project Roslyn").
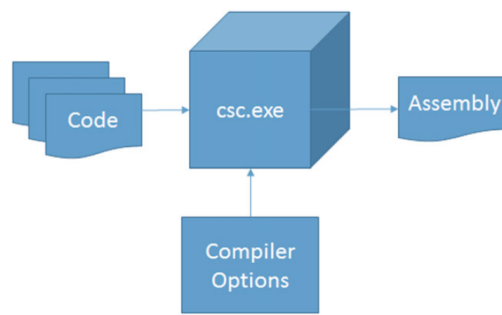

Figure 3.1: Compiler as a black box [**dot-net-development-using-the-compiler-api**]

Not only have been compilers for both VisualBasic and C# rewritten into entirely managed code, they also expose the internals of the compiler pipeline via a public .NET API [1]. This makes them a platform (also known as *compiler-as-as-service*) with rich code analysis APIs that can be leveraged by developers to perform analysis, code generation or dynamic compilation in their own programs [**roslyn-succinctly**]. Those can be later easily integrated into VisualStudio all without the hard work of duplicating compilers' parsing logic.

This chapter will take a look at how the Roslyn API layers are structured, how the original source code is represented by the compiler and how developers can build tools upon the compiler's API. Lastly it will provide a short overview and evaluation of other tools that

---

1. Application Programming Interface

were used before Roslyn or have emerged thanks to .NET Compiler Platform.

Note that although Roslyn provides equivalent APIs for both VisualBasic and C#, this thesis will only focus on the C# part, since only that one is relevant for the practical part of the thesis.

3.1 The Compiler Pipeline

Roslyn compilers expose an API layer that mirrors the traditional compiler pipeline (see 3.2). Instead of a single process of generating the target program, each compilation step is treated as a separate component [**roslyn-overview**]:

- **Parse phase** consists of *lexical analysis* (*scanner*) and *syntactic analysis* (*parser*). First, the lexical analyzer processes the stream of characters from the source program and groups them into a meaningful sequences called *lexemes*. Those are subsequently processed by the *syntax analyzer* that creates a tree-like structure of tokens based on the language grammar [**dragon-book**].

- **Symbols and metadata phsase** where named symbols are generated based on the declarations from the source and imported metadata.

- **Bind phase** in which the identifiers from the source code are matched to their respective symbols.

- **Emit phase** where all the gathered information is used to emmit an assembly.

The .NET Compiler Platform creates an object model containing information obtained by each phase and exposes it through the API in form of .NET objects. These object are also used internally by VisualStudio [2] to support basic IDE functionality. For instance **syntax tree**, that is the result of the parse phase, is used to support formatting and colorizing the code in the editor. The result of the second phase – **hierarchical symbol table**, is the basis for "Object browser"

_____

2. The new generation of VisualStuio leveraging from the Roslyn compiler are called vNext and first one was VS 2015.

Figure 3.2: Compiler pipeline [**roslyn-overview**]

and "Navigate to" functionality. Binding phase is represented as an **object model that exposes the result of the semantic analysis** and is utilized in "Find all references" or "Go to definition". Finally, the Emit phase produces the Intermediate Language (IL) byte codes and is also used for "Edit and Continue" feature [**roslyn-overview**].

3.2 The .NET Compiler Platform's Architecture

The Roslyn's architecture consists of two main layers (Compiler and Workspaces) and one secondary layer (Features), as seen on Figure 3.3.
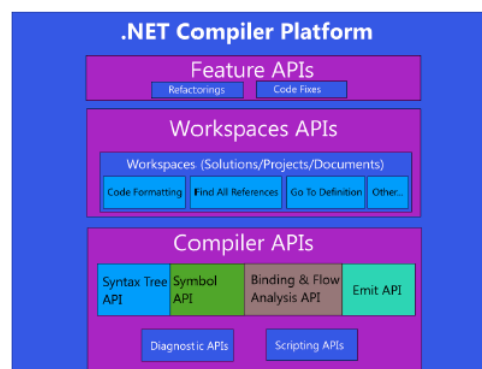

Figure 3.3: Compiler pipeline [**roslyn-succincly**]

One of the key concepts of .NET Compiler Platform is immutability. The compiler exposes hundreds of types that represent all information about source code from `Project` and `Document` to `SyntaxTrees` with almost all of those types being immutable. This means, that once

created, the object cannot change. In order to alter it in any way, new instance must be created, either manually, or from an existing instance by applying one of many `With...()` methods that the API provides.

The Immutability enables the compiler to perform parallel work without need to create duplicate objects or apply any locks on them. The concept is useful for the command line compiler but it is considered extremely important for IDEs where it enables one document to be analyzed by multiple analyzers in parallel.

3.2.1 The Compiler APIs

As discussed in the previous section, the Compiler APIs offer an object model representing the results of syntactic and semantic analysis produced by the respective phases of the compiler pipeline. Moreover, it also includes an immutable snapshot of a single compiler invocation, along with assembly references, compiler options, and source files. This layer is completely agnostic of any Visual Studio components and as such can be used in stand-alone applications as well. There are two separate, though very similar, APIs for Visual Basic and C#, each providing functionality tailored for specific language nuances.

Diagnostic APIs

Apart from parsing code and producing an assembly, the compiler is also capable of raising diagnostics, covering everything from syntax to semantics, and report them as errors, warnings or information messages [**roslyn-succinctly**]. This is achieved through the compilers' Diagnostics APIs that allow developers to effectively plug-in to compiler pipeline, analyze the source code using the exposed object models, and surface custom diagnostics along with those defined by compiler itself. These APIs are integrated to both MSBuild [3] and Visual Studio [4] providing seamless developer experience. The practical part of this thesis hugely relies on Diagnostic APIs to provide custom diagnostics, the details will be discussed in the 9.

---

3. Does this needs an explanation??
4. Visual Studio 2015 and newer.

6

Scripting APIs

As a part of the compiler layer, Microsoft team has introduced new Scripting APIs that can be used for executing code snippets. This APIs were not shipped with .NET Compiler Platform 1.0 and are part of v2.0.0 Release Candidate 3[5].

3.2.2 Workspaces APIs

Workspace represents a collections of solutions, projects and documents. It is a starting point for performing code analysis and refactorings over entire solutions. It provides a single object model containing information about the projects in a solution and their respective documents; exposes all configuration options, assembly and inter-project dependencies, and provides access to syntax trees and semantic models.

Although it is possible to use the Workspace outside of any host environment, the most common use case is an IDE providing an instance of Workspace that corresponds to the open solution. Since the instances of Solution are immutable, the host environment must react to every event (such as user key stroke) with an update of the `CurrentSolution` property of the Workspace. This cycle is illustrated in Figure 3.4.

3.2.3 Feature APIs

This layer relies on both Compiler and Workspaces layers and is designed to provide API for offering code fixes and refactorings. This layer was also utilized while working on the practical part of this thesis.

---

5. As per https://github.com/dotnet/roslyn/wiki/Scripting-API-Samples [26-02-2017].
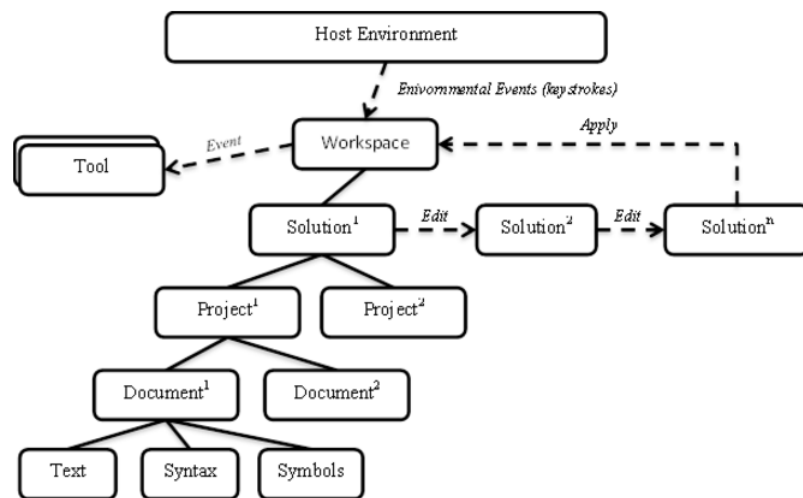
Figure 3.4: Compiler pipeline [**roslyn-overview**]

3.3 Syntax

3.4 Semantics

3.5 Analysers and Code Refactorings

3.6 Other Tools

Current situation & motivation

4 Custom Roslyn Analyzers

A Upgrades and Versioning

TODO...