DECEMBER 5, 2018

REMOTE PACKAGE DEPENDENCY ANALYSIS (OCD)

PROJECT 4

DEBOPRIYO BHATTACHARYA (SUID: 451438326)
CSE-681: SOFTWARE MODELLING AND ANALYSIS (FALL 2018)

INSTRUCTOR: DR. JIM FAWCETT

Contents

1 Executive Summary

Dependency Analysis of project files is very important; it gives us a clear picture of how the files in our project are dependent on each other. If we do not know the dependencies that our project files have, then we will not be sure which files to update when changes are made to certain files. This can be a fatal error and can result lot of complications and bugs that are almost impossible to get rid of. The lexical scanner's architecture contains following components:

- Source Code Files: The files on which our dependency analyzer will run.
- Tokenizer: Divides the source code into simple tokens by removing unnecessary symbols, spaces, and comments. It uses the process of lexemes to do this task.
- Semi Expression Generator: Now, generated tokens pass to Semi Expression Generator and goes through a process of syntax checking. Here the tokens are grouped together that using a predefined grammar. Each time Semi Expression Generator requests the Tokenizer for a new token, until all tokens are finished.
- Parser: The parser uses the Semi Expression Generator and Tokenizer. It has all the rules defined to determine different types of expressions, it has scope stack to determine scope of objects. It also contains the typetable class that it finally used to display the files, names of different variables, classes, namespaces, delegates, enums, functions, structs and interfaces.
- DependencyAnalyzer: This is used to do a second pass of all the files and then compare the found types with the typetable to determine which files are dependent on which file.
- Graph: This package uses the dependency data from and then implements Tarjan's Algorithm to determine the strong components that are formed using that dependency data. Our target will always be to have as many strong components as packages, which would mean that there are low dependency among the different packages.
- AnalyzerServer: This is the backend of our Analyzer, that receives path to directory or path to multiple files and uses the dependency analyzer to do the analysis and returns the result to the client GUI interface. It also returns paths to different files and tells the Client which directory contains what files.
- AnalyzerClient: This is the gui interface that allows a user to traverse through the

different server and client directories that are allowed by the program and see the different files. The user is able to select the files and then ask the server to analyze dependency of those files, or the user can just select a path and then see the result of the analysis of all the files inside the directories of that path.

- MessagePassingComm: This is the backbone of the Client-Server model which is used to send and receive messages between the AnalyzerClient and AnalyzerServer.

The analyzer requires administrator privileges to work, this can be a critical issue for machines where administrator privilege might not be available. The total number of files if very large can prove to be a challenge, as the entire running time of the analyzer will be very long. If the underlying parser fails to identify some character or tokens, it might lead to problems that have not been handled.

2 Introduction

The purpose of this project is to build a robust dependency analyzer that can be used not only for C# but many other languages by changing the underlying parser. It consists of multiple packages that form a complete tool with a graphical user interface and a backend server.

The client is a package, based on Windows Presentation Foundation (WPF), residing on the local machine. This package provides facilities for connecting a channel to the remote server. This package provides the capability for sending requests messages for each of the functionalities of our dependency analyzer, and for receiving messages with the results, and displaying the resulting information.

The Analyzer Server package residing on a remote machine that exposes an HTTP endpoint for Comm Channel connections. The server uses the dependency analyzer to generate the dependency and then send the results to the client.

The MessagePassingComm package implements asynchronous message passing communication using the Windows Communication Foundation Framework (WCF), which provides a well-engineered set of communication functionalities wrapping sockets and windows IPC.

The dependency analysis package finds, for each file in a specified collection, all other files from the collection on which they depend. File A depends on file B, if and only if, it uses the name of any type defined in file B. It uses the parser to find the typetable of the files. Then uses the defined rules and actions to decide on the dependency.

Finally, we come to the strong component generator. It treats each file as a node on a graph, then it implements edges according to the dependency of the different files. Then we use Tarjan's algorithm to find the strong components.

Tokenizer package is used to detect lexemes. A lexeme is a sequence of characters that matches the pattern defined in the different state pattern classes. Tokens are lexemes that are mapped into a token-name and attribute value. That is, the program understands which category of token a lexeme belongs to. These tokens are bounded by whitespace characters. They may be alphanumeric characters, punctuator characters, strings, single character, multiline comments or single line comment. The tokenizer has a function to look ahead to the next character in the stream so that it can easily decide where one token ends and the next one begins. This helps the tokenizer to effectively partition the input stream.

The SemiExpression package will receive these tokens. SemiExpression groups tokens into sets, each of which contain all the information needed to analyze some grammatical construct without containing extra tokens that have to be saved for subsequent analyses. SemiExpressions are determined by special terminating characters: semicolon, open brace, closed brace, colon when preceded on the same line with 'using' and some other conditions.

3 Uses and Users

A typical application of remote code analysis is for Code Repositories. For that, Quality Assurance staff will run analyses on code in a remote repository from clients on their desktops. Also, developers will analyze code, written by other developers, that they need for their own work.

But, a dependency analyzer can also be used by individual developers to check dependency of the code in a project, which gives a clear idea about the project itself and also how to proceed on making changes. Because if a one file is updated, then all the files that depend on it need to be updated as well.

The dependency analyzer can also be used to make sure that the development of the code is going in the right direction. The packages should always have minimal dependency. The strong components generated from the dependency data tell us which files depend on which files and gives an overall picture about the entire project. Developers always try to minimize dependency and hence maximize the number of components.

4 Partitions

4.1 Packages

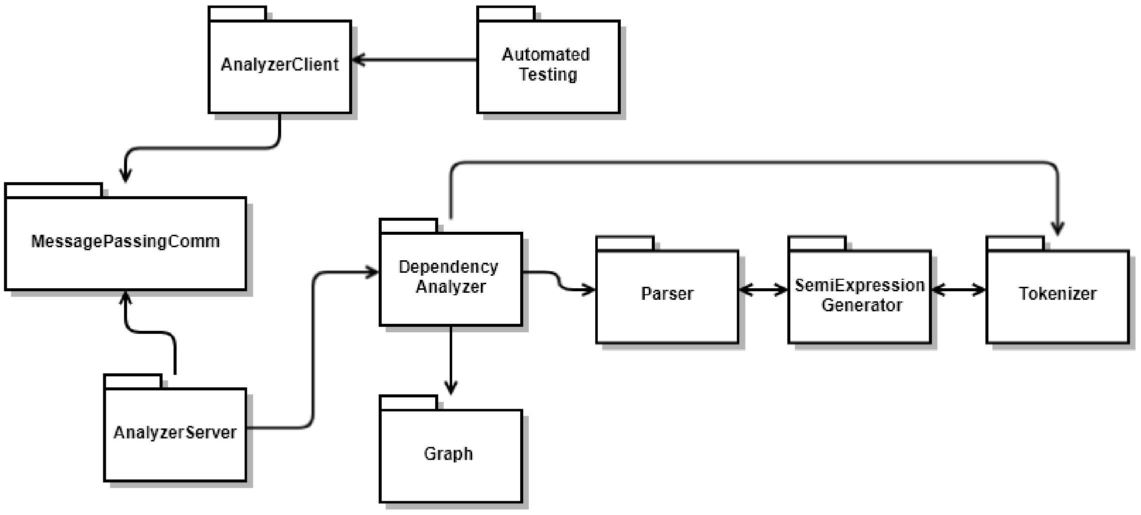

Figure 1: Package Diagram of Remote Package Dependency Analysis Tool

The project is broadly divided into four parts broadly. The GUI Client package which the user uses to select files and directories and see the result of the analysis. The Analyzer Server program that receives the names of the files or the path that the user chose for analysis. The MessagePassingComm for communication between the client and server. Finally, the actual dependency analyzer part which is used to do the analysis. This final part contains the parser, semi expression generator, tokenizer and also the graph which is used to determine the strong components depending on the dependency.

We can just change the parser and we will be able to implement this tool for different programming languages.

4.2 Classes

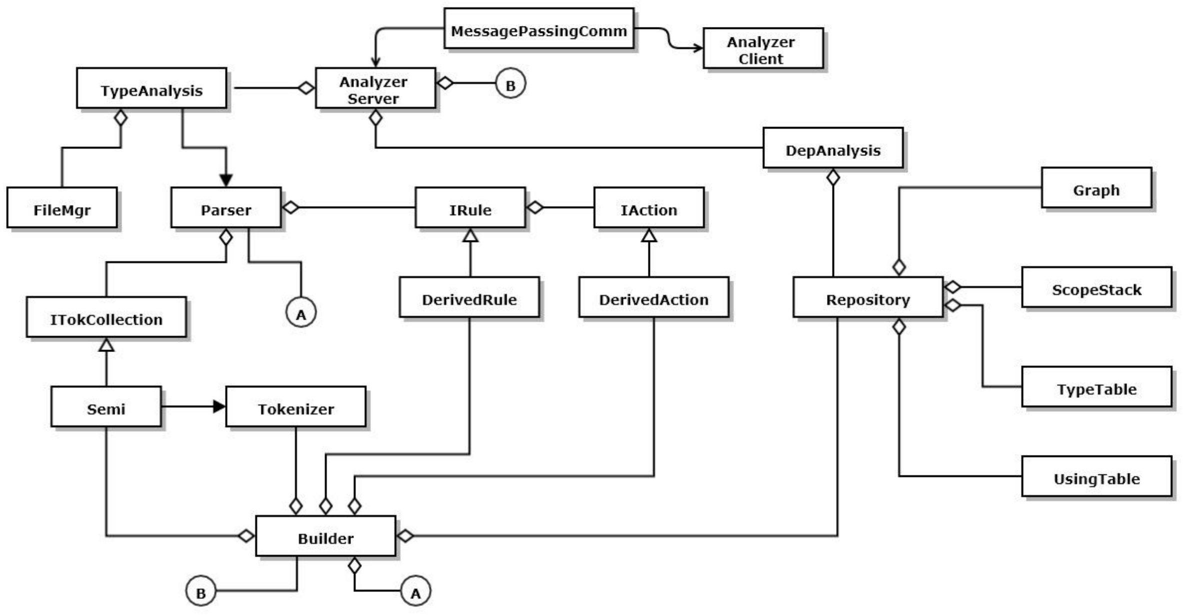

Figure 2: Class Diagram of Remote Package Dependency Analysis Tool

In this class diagram we try to describe all the classes that are present in the Dependency Analyzer. First we have the client gui from which the files to be selected for the analysis are determined and the final outcome is also displayed, in separate tables. Using the MessagePassingComm we send the path to the server program. The server program determines the full path of the files and then uses the dependency analyzer to find the dependency among the chosen files. It does so using the Parser. The Parser implements ITokenCollection where it gets the semi expressions from the tokens.

SemiExpression will call the getTok() function to get tokens generated so far from the tokenizer. Then it will use specific rules like when a group of tokens end with a semi-colon when they together form a single expression. But there can be a few other cases to check in this case, for example in a for loop, there can be multiple semi colons inside the parenthesis, hence we need to put a precedence case of detecting parenthesis tokens which when closes will form an entire expression. Other rules in the SemiExpression class can be sets starting with keywords like “using” and “namespace”. In such cases the rest of the line would be a single expression.

The SemiExpression class implements the ITokenCollection interface. We can design multiple SemiExpression classes for different kinds of programming languages. The tokenizer can remain the same. The ITokenCollection outlines the different methods and situations that we need to handle while accepting tokens and generating expressions using the grammar rules that apply for that specific programming language. In this case we are concentrating specifically on C#. But we can design SemiExpression generators for different programming languages using the same tokenizer package.

The dependency analyzer is described as Builder in this diagram. Once it has got all the semi expressions it uses the rules defined in rules and action to find the different types of variables, classes, namespaces, delegates, enums, functions, structs and interfaces.

The dependency analyzer finally determines the strong components using the graph package before sending all the result details back to the client.

5 Application Activities

A typical application of remote code analysis is for Code Repositories. For that, Quality Assurance staff will run analyses on code in a remote repository from clients on their desktops. Also, developers will analyze code, written by other developers, that they need for their own work.

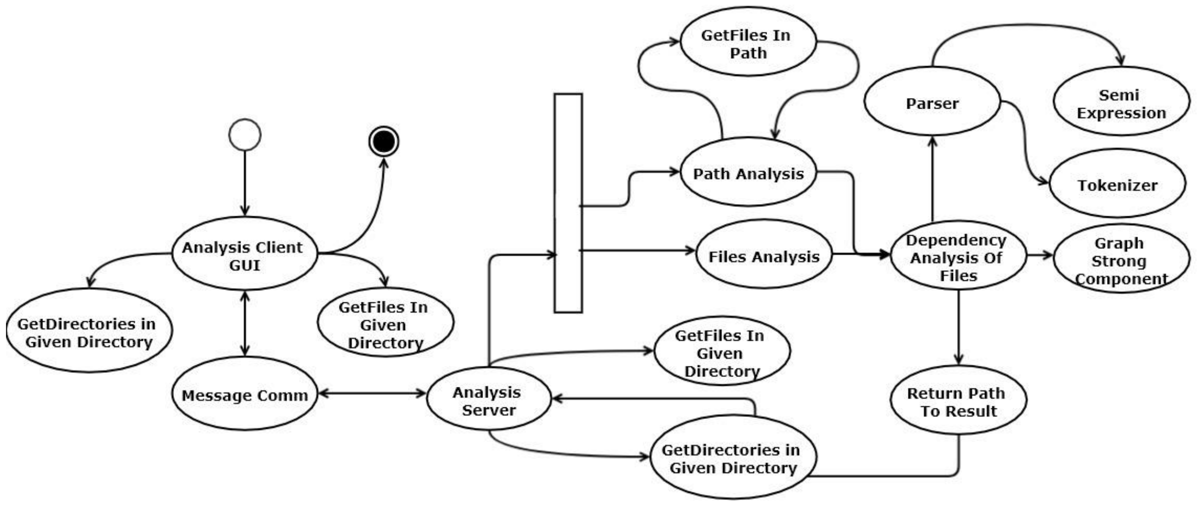

Figure 3: Remote Package Dependency Analysis Tool Activity Diagram

The GUI Client package which the user uses to select files and directories and see the result of the analysis. The Analyzer Server program that receives the names of the files or the path that the user chose for analysis. The MessagePassingComm for communication between the client and server. Finally, the actual dependency analyzer part which is used to do the analysis. This final part contains the parser, semi expression generator, tokenizer and also the graph which is used to determine the strong components depending on the dependency.

6 Critical Issues

6.1 Efficiency and Scalability

One of the most important issues that might arise is, how many files the server can analyze and from how many connections can it analyze files. Thanks to the multi-threaded nature of the analysis our Dependency Analysis Sever is able to handle many requests from multiple clients and do dependency analysis in parallel.

This project is very much scalable and can be adapted to work with different programming languages by changing the Rules and Actions defined in the parser. The highly modular nature of the project makes it a trivial task.

6.2 Portability

Portability is also an important criterion to consider. Since projects can be worked on by many developers, working at different stages of the software development process on different systems. It is important that the project is easily portable from one platform to another. Visual Studio immensely helps in this aspect. The developer can forget worrying about the platform and concentrate on the software architecture, features, functions.

6.3 Ease of Use

Ease of use of the project code is also an important aspect. It might not be intuitive for the user to know how the GUI functions. And might not be able to effectively use the program. Hence a hint was added right above the files selected box which displays which files have been selected, to tell the user how to select and deselect files and folders.

7 References

1) https://ecs.syr.edu/faculty/fawcett/handouts/CSE681/code/Project1HelpF2018/

2) https://ecs.syr.edu/faculty/fawcett/handouts/CSE681/code/Project2HelpF2018/

3) https://ecs.syr.edu/faculty/fawcett/handouts/CSE681/code/Project3HelpF2018/

4) https://ecs.syr.edu/faculty/fawcett/handouts/CSE681/code/Projec41HelpF2018/

5) https://ecs.syr.edu/faculty/fawcett/handouts/CSE681/midterm/MTF18/MT2F18-InstrSol.pdf

6) https://docs.microsoft.com/en-us/dotnet/csharp/

7) https://stackoverflow.com