12/5/2018

Project #4

Code Analyzer:

Operational Concept Document

CSE 681 – SOFTWARE MODELLING AND ANALYSIS

INSTRUCTOR – Dr. JIM FAWCETT

Anish Nesarkar

SUID - 368122582

Contents

1. EXECUTIVE SUMMARY

The purpose of this project is to build a software system tool for Code Analyzer to perform Code Analysis on a desired C# program. It can also be extended to other similar languages like C++ and Java.

The Code analyzer will be basically used for evaluating package dependencies to determine whether our package has dependencies on other packages and if any other packages depend on us, searching for particular construct in the code maybe to check ownership of code files, construct code metrics to measure the complexity and maintainability of the managed code.

The Code Analyzer is implemented on a remote machine. The users connect to the remote server which holds the Code analysis tool that performs type analysis, dependency analysis and determines strong components between a given set of files.

The intended users and uses for the Code Analyzer would be:

- Parser : The most important user for the Code analyzer would be a parser or a compiler or an interpreter. The stream of Tokens generated by the lexical analyzer is sent to parser for syntax analysis.

- Packages : The Code analyzer is useful for checking package dependencies. i.e. to know whether a class defined in one package is declared or not in other class of different package.

- Developers/Programmers : The Code analyzer is used by Developers or programmers to analyze code written by other developers that they need for their own work.

- Quality Assurance team : The Code analyzer is used by the Quality Assurance team to run analyses on code in a remote repository from clients on their desktops.

Some of the critical issues that we would be facing in this project are given below. The solutions for the critical issues are provided in further sections:

- There are complications in tokenizer when it comes to take into consideration some special tokens, comments, quoted strings. They can become complicated to recognize in a program depending upon its usage.

- The semiExpression is accessed by the parser using the ITokenCollection interface. This helps in avoiding the binding to the concrete details of the Lexer. But, the rules in the parser needs to access almost all the instances in the semi instance functionality.

- The .Net does not provide peeking more than the first character. Thus, deciding the next state becomes very difficult if we need to look next two available characters.

- The programming language used C# does not provide pointers to function.

2. INTRODUCTION

Lexical Analysis means grouping a stream of data like characters or letters into meaningful syntax of a source code. Code analysis determines the validity of syntactic organization of the program.

The Code Analyzer that we are designing for code analysis is taking a stream of token data in a queue and performing a lexical analysis on it. Tokens are numbers, operators, keywords, constants, identifiers, string, special characters. The SemiExpressions are a list of tokens of any language which are syntactically correct. They are generated by appending a list of tokens until a terminating token is reached. We have generated semiExpressions in project 2 for a list of terminating characters.

A Typetable is a table of collection of all the data types used in the program. It may consist of namespaces, using statements, structs, aliases, delegates etc. Dependency analysis is the analysis made on a set of files to determine if the files are dependent on any other file in the set. A strongly connected components are a set of nodes in a graph wherein from every node we can reach to any other node in that component. In this project, the files are considered as nodes. Thus, typetable, dependency analysis and determining strong components is done in the project 3.

A Windows Communication Foundation (**WCF**) is a framework for building service-oriented applications. Using **WCF**, you can send data as asynchronous messages from one service endpoint to another. Windows Presentation Foundation (**WPF**) is a graphical subsystem by Microsoft for rendering user interfaces in Windows-based applications. The WPF would be Client in this project who will request for files that would be used to perform code analysis in the remote server. The next section talks about the organizing principles for this project.

The packages will be explained in detail in the later sections of this document.

2.1 Organizing ideas

The Code analyzer project idea is to have a client server network wherein the server consists of code analysis tool that generates typetable, dependency results and strong components between a set of files which the client requests. A simple tokenizer takes source code files as inputs and outputs characters, numbers, keywords, identifiers etc. in the form of tokens. After successfully implementing tokenizer on C# source code, a Semi-expression package is implemented which 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. Semi-Expressions are determined by special terminating characters: semicolon, open brace, closed brace, and newline when preceded on the same line with 'using'. These tasks are done in project 2.

In project 3, A typetable is generated which stores the type information in each of a collection of C# source files. It does this by building rules to detect type definitions such as classes, structs, enums, namespaces, aliases etc. After generating the type table , dependency analysis is performed for each file in a set of collection of files. A strong component in graph which gives information of all the files which can be reached from any other file in the set by following direct or transitive dependency links. Tarjan's algorithm is used to determine the strong component. All these tasks are performed on remote server.

In project 4, A WCF communication service is used to establish communication between client and server. A WPF is used as a client which will request a set of files that Code analysis should perform on the server. After performing the operations, the server send back the results of typetable, dependency result and strong component result back to the client.

The entire project is built on Visual Studio 2017 using C# language and .NET framework 4.6.1. The final idea is to implement remote code analysis on the source code.

3. PARTITIONS

The Lexical Analyzer that is being designed is illustrated using Package Diagram, Class Diagram and Activity Diagram.

3.1 Package Diagram

The package diagram of the entire project is as shown below.

The package diagram of Lexical Analyzer illustrates the different packages used mainly the Tokenizer, SemiExpression package and Automatic Test Suit on the server.

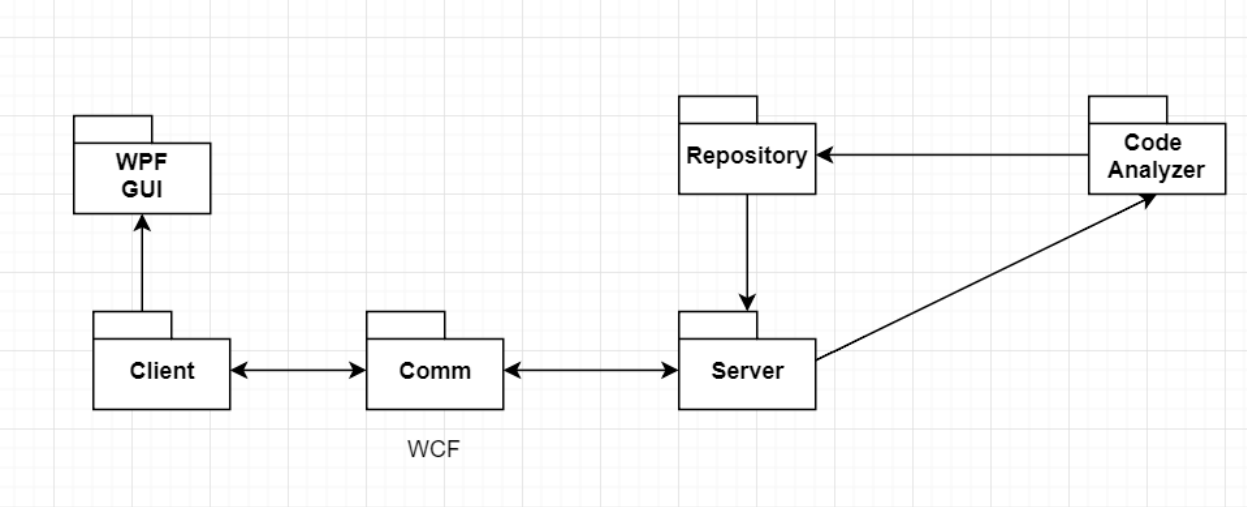


Fig 1 : Package Diagram of Code Analyzer

3.1.1 Client

The Client package consists of WPF GUI which is intended for the user to select a list of files from the server repository and perform Code analysis on them. 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 capabilitiy for sending requests messages for each of the functionalities of Project #3, and for receiving messages with the results, and displaying the resulting information This package communicates with the server using the WCF service. The client sends a request message to the server over the channel to perform code analysis on a set of files and receives an acknowledgement from the server the output result for the request.

3.1.2 Comm

The Comm package consists of message communication service between server and the client. The Comm 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.

3.1.3 Server

This package resides in the remote machine that exposes an HTTP endpoint for Comm Channel connections. The server implements all the functionalities developed in project 3. It accesses the files from the repository and sends it to the client to select. The selected files are then sent to Code Analysis to perform the required operations. i.e. to get typetable, dependency result, strong component result.

3.1.4 Repository

This package stores all the required files for the project. The Server package accesses this package to send the files to the client to select them for analysis. The Code Analysis package accesses this package to the operations on the selected files.

3.1.5 Code Analyzer

The Code Analyzer package mainly consists of the Tokenizer and SemiExpressions along with Parser, ITokenCollection, Rules and Actions etc. This package does all the backend core operations in this project. The Server accesses this package to carry out the required functionalities like get type typetable, get dependency result and strong components for the files. The important packages in Code Analyzer package are described below mainly the Tokenizer, SemiExpression, TypeAnalysis, Dependency Analysis and Strong Component.

3.1.6 Tokenizer

The Tokenizer package extracts words, called tokens from a stream of characters. The Token boundaries in the package are White spaces, alphanumeric and punctuator characters, comment and string boundaries, special single and double characters, special loop boundaries (ex. for loop) which require special rules for extraction.

3.1.7 SemiExpression

In SemiExpression package, groups of tokens are collected in the form of 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. The SemiExpressions are determined by special characters: semicolon, open brace, closed brace, and newline when preceded on the same line with ‘using’. It also special function for determining loops (ex. for loop).

3.1.8 Type Analysis

The type analysis package finds all the type information for a particular set of files that are required for dependency analysis. It does this by building rules to detect type definitions - classes, structs, enums, and aliases.

3.1.9 Dependency Analysis

The dependency analysis 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 might do that by calling a method of a type or by inheriting the type. Note that this intentionally does not record dependencies of a file on files outside the file set, e.g., language and platform libraries. The result of the dependency analysis is sent to the Strong Component package to determine the strong components.

3.1.10 Strong Component

A strong component is the largest set of files that are all mutually dependent. That is, all the files which can be reached from any other file in the set by following direct or transitive dependency links. The term 'Strong Component' comes from the theory of directed graphs. There are a number of algorithms for finding strong components in graphs. This package finds all the strong components for the given set of files.

3.2 Class Diagram

The below figure is a client-server class diagram for project 4.

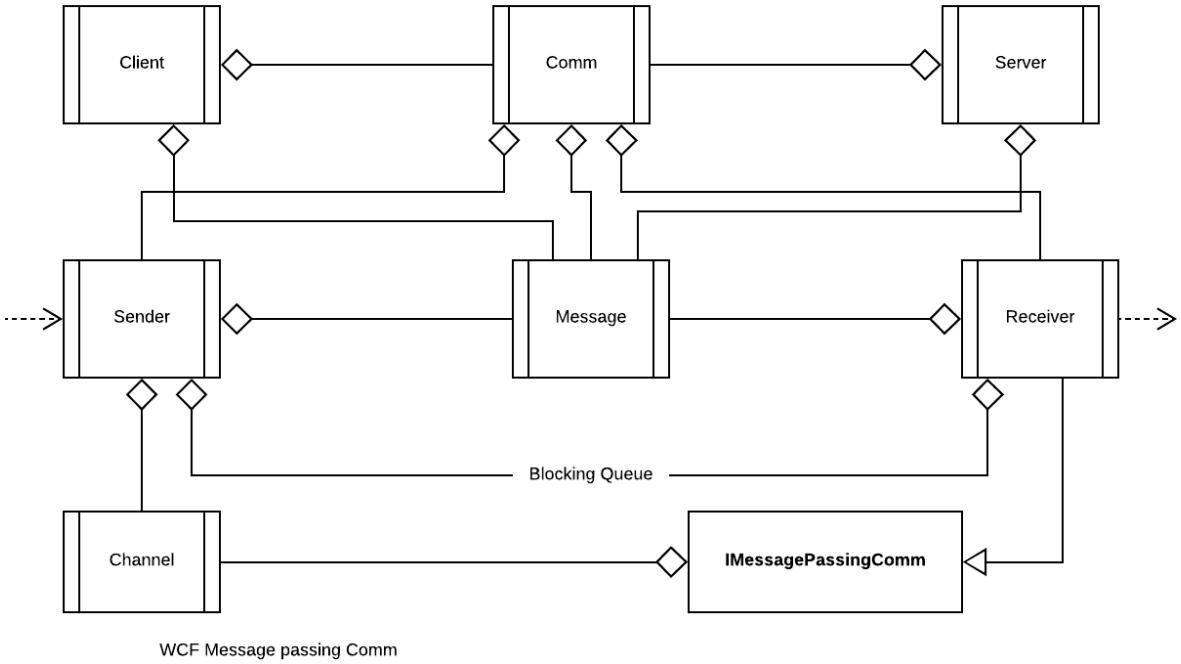


Fig 2 : Client-Server Class Diagram

The above figure shows class diagram for Client-Server Network. A brief description for each of the blocks will be given below.

3.2.1 Client

The Client Class in the above class diagram is the GUI WPF. It aggregates the message class and the Comm class.

3.2.2 Comm

The Comm class forms a communication channel between the client and receiver. It aggregates the sender, receiver and message classes.

3.2.3 Server

The server class performs code analysis and sends back result to client. It aggregates Comm class and message classes as shown in the figure.

3.2.4 Message

The message class holds the attributes that are sent over the channel like the command, arguments, ToAddress, FromAddress etc.

3.2.5 Sender

The sender class is contained in Comm class. It aggregates message class, channel andblocking queue.

3.2.6 Receiver

The receiver class is contained in Comm class. It aggregates the message class, blocking queue and also inherits the interface IMessagePassing Comm.

3.2.7 Channel

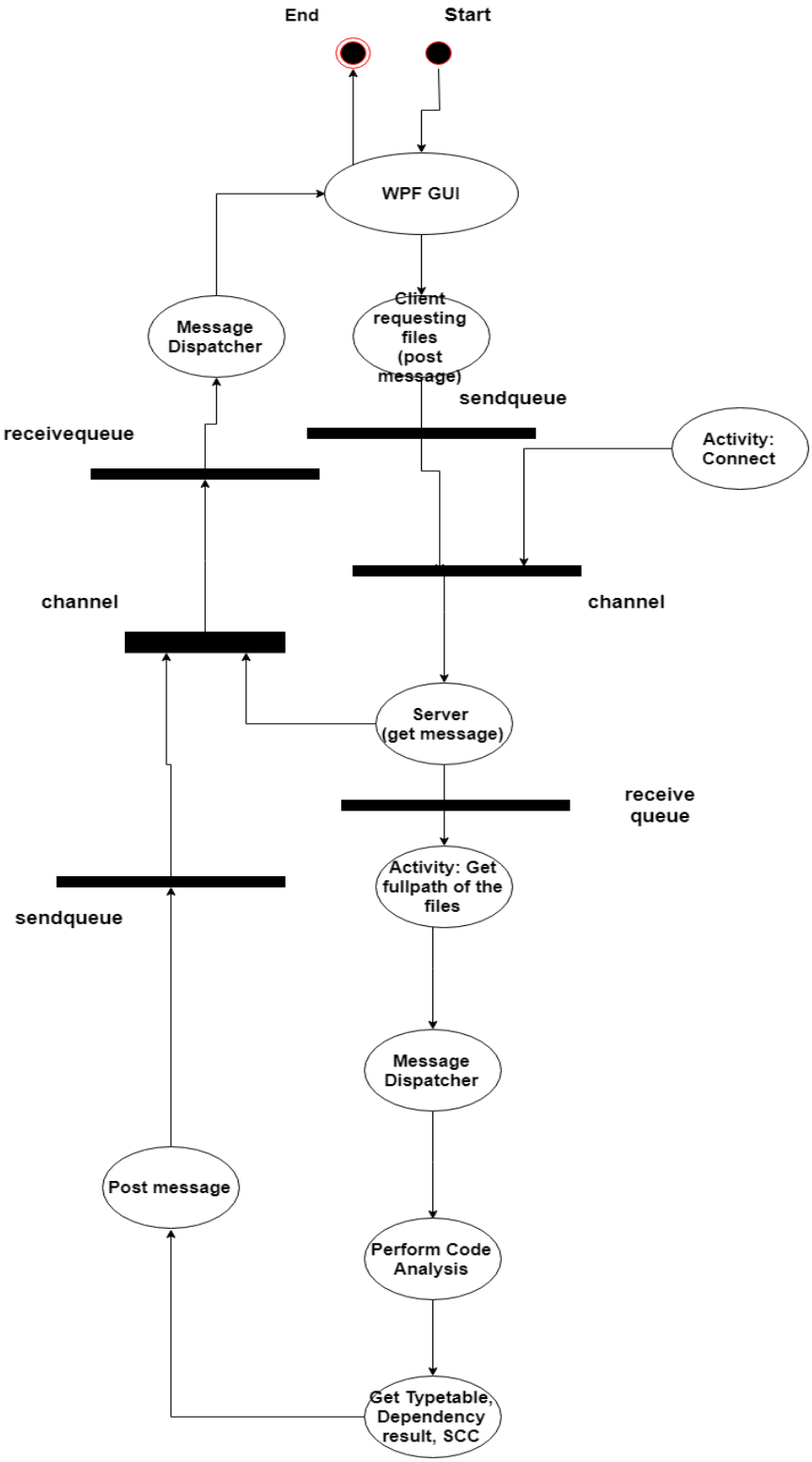
The channel class acts as a communication between the sender and receiver. It is aggregated by Sender and the interface ImessagePassingComm.

3.2.8 IMessagePassingComm

The IMessagePassingComm acts as an interface which aggregates the channel class.

3.3 Activity Diagram

The below figure shows complete activity diagram for remote Code Analyzer. It represents the flow from one activity to other activity. It describes the dynamic aspects of the system.



3.3.1 Application Activities

The project starts from the activity WPF GUI. The client sends a request message to server by enqueuing the message in a blocking queue. The message is sent to the server only when both the client and server are connected to the channel.

The Server then extracts the received message from its receive queue by calling getmessage() function. Based on the command received from the client in the message, a message dispatcher is called having a lambda function for the corresponding command. The next activity then performs code analysis to get either typetable, dependency result or scc result based on the command request of the client. The server post backs the reply message to the client by enqueuing it in the send queue. The server then posts back the reply message through the channel.

The message sent to the client through the channel is enqueued in the receive queue of the client. The message dispatcher of the client is invoked based on the command received by the server. The GUI displays the output from the message dispatcher which can be typetable, dependency output or strongly connected component output. The Activity ends when the client clicks the close button on the GUI.

4. USES

- The Lexical analyzer is used by the parser to break data into smaller elements.
- Manual code reviews are time-consuming, error-prone and costly, the need for automated solutions has become evident. Therefore, static source code analysis using Lexical Analyzer are implemented to address the problem of Vulnerable Web applications.
- The Lexical Analyzers can also be used to find a keyword in a document or determine the count for the keyword in that document and group it accordingly in a database. For example, related articles, books, content writings etc.
- The Code analyzer is useful for checking package dependencies. i.e. to know whether a class defined in one package is declared or not in other class of different package.
- The Code analyzer is used by the Quality Assurance team to run analyses on code in a remote repository from clients on their desktops.

5. CRITICAL ISSUES

5.1 Special Cases in Tokenizer

There are complications in tokenizer when it comes to take into consideration some special tokens, comments, quoted strings. They can become complicated to recognize in a program depending upon its usage.

Solution : Different states are used for the tokens. Thus for the major special cases, a new state can be implemented.

5.2 Peek function in .Net

The .Net does not provide peeking more than the first character. Thus, deciding the next state becomes very difficult if we need to look next two available characters.

Solution : A character queue can be created in a TokenSource class that uses .Net stream. Multiple characters can be added to that queue without removing the characters from the TokenSource instance. The characters are needed are enqueued in a queue so that they remain at the source. If any character is required, we dequeue the queue first, then pull the character from the stream.

5.3 Parser needs to access all Semi instance functionality

The semiExpression is accessed by the parser using the ITokenCollection interface. This helps in avoiding the binding to the concrete details of the Lexer. But, the rules in the parser needs to access almost all the instances in the semi instance functionality.

Solution : The ITokenCollection can have most of the public interfaces of the Semi. Thus, the rules and actions of the parser can manipulate token collections more effectively.

5.4 Security and Vulnerability

This issue is caused when a programmer does any mistake while writing the program for a software and this might make the source file of the software vulnerable.

Solution : Static code analysis of the lexical analyzer can be used to solve this issue scans through a file looking for syntactic matches based on several simple “rules” that might indicate possible security vulnerabilities.

6. REFERENCES

https://ecs.syr.edu/faculty/fawcett/handouts/CSE681/Lectures/cse681codeL1.htm

https://ecs.syr.edu/faculty/fawcett/handouts/CSE681/Lectures/StudyGuideOCD.htm

https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/

https://www.tutorialspoint.com/compiler_design/compiler_design_lexical_analysis.htm

https://courses.cs.washington.edu/courses/cse484/14au/reading/bsi5-static.pdf