Operational Concept Document(OCD)

for

A Design Of Remote code Analyzer

Author's Name: Ren Jie (SUID: 21037-2590)

CSE681 software modeling and analysis

Instructor:Dr. jim Fawcett

Date:12/04/2018

Table of Content

1. Executive Summary

In computer science, Code Analyzer is one of the useful tools to parse the code and files with in a project or solution. Especially when the project is large and contains thousands lines of code. This Remote Code Analyzer will be able to analyze all the .cs files under a certain directory and find their dependency and all the Strong component. A well designed code analyzer can not only analyze the Csharp files correctly but can also easily expand its function to parse different form of language like java and C++ with reasonable amount of changes.

The potential users of this code analyzer include the computer science students, professors, graders, developers, data scientists and companies. Understanding users' need and build a user friendly interface is a critical to design a good lexical scanner.

This code analyzer will have two major parts which are the Client part and the server part. They work collaboratively to allow user to execute code analysis on the remote server and get the result of the dependency and strong components among the target files. Each of Client part and the server part is consist of several functional modules to meet the requirement of this project and work as expected. The detail of their partition and activity will be discussed later in this OCD .

The critical issues exist in each level of this project. From the top level, the critical issues include the communication protocol between client and server, message dispatching, communication exception handling and security issue. about the detail of the implementation of Code analyzing, The critical issues includes special cases like identify two-character operator tokens in tokenizer, terminal conditions in semiExpression collector and etc. it also includes some details about the how to enter the next state of the tokenizer since we already peeked the characters in the terminal judgement procedure. Those critical issues will be discussed later in this OCD.

2. Introduction

2.1 Concept and function

On the Client side, the client part of this project should provide a user-friendly GUI to send request and display reply. It should have a navigator for user to navigator the remote directories in the server, ways for user to send the analysis request and a well-formed GUI to display the analysis result to the user.

On the Server side, the server should first have a filemanager to navigate the directories as the user requested. But also mostly importantly is to have a module to execute the code analysis order. This include the actions to scan files and extract char collections as Tokens, collecting the tokens into semiExpressions, find out the user defined type, analyze the dependency between files and find out all the strong components base on their dependencies.

The whole project should satisfice the following requirement

- The Server packages shall evaluate all the dependencies between files in a specified file set, based on received request messages.
- The Server packages shall find all strong components, if any, in the file collection, based on the dependency analysis, cited above.Provide an interface for other functions to call.
- The Client packages shall display requested results in a well formated GUI display.
- Shall include an automated unit test suite that demonstrates you meet all of the functional requirements, stated above

3. Users and uses

The code analyzer is an essential and fundamental part in many areas and applications. Therefore, with appropriate adjustment for actual cases, it can be expanded to meet the needs of different uses and users.

3.1 students

Students can use this code analyzer to check their homework themselves before they hand their work to their professors. Especially when their homework is not really code and cannot be checked by the IDE(for example: presudo code, math equations, etc). They only need to make some adjustment to the semiExpression.
They can also find the flaws in their project structure by finding the strong components and file dependency and make sure which files are worth to pay extra attention.

3.2 professors and graders

Professors and graders can use this code analyzer to analyzer their student's homework and help them to grade, it will save them a lot of time and effort.

3.3 software developers

This application is mostly designed for the software developers. They can make use of it in different ways. They can not only use the lexical scanner as a development tool to help them analyzing the code they write in their developing process, but also as part of the applications they are going to implement. Possible use cases as part of the application includes compiler, parsing web information at backend, xml parser, search engine for their application database and any other application which needs to parse a stream of Strings.

3.4 companies

Creating an effective evaluating system is critical to maintain a progressive company culture. Companies may develop applications base on this code analyzer to supervise their employee's contribution and performance. Employers may also use any other technology of information collection and analyze to manage their stuffs.

Companies and also use this code analyzer to manager their project better. It will help them to determinate which files are basic ones because a lot of files depend on them so that they can test those important files more seriously.

4. Application activity

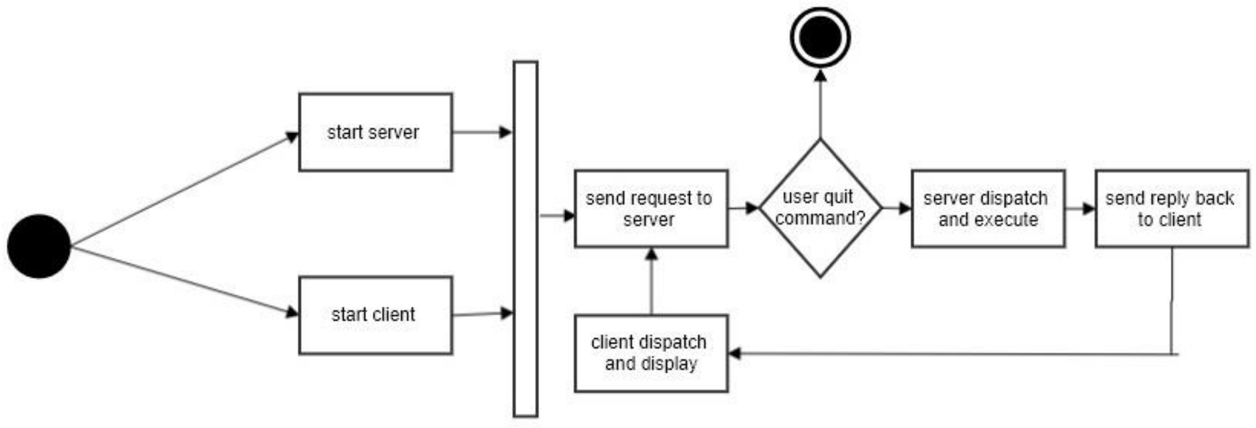
start server
start client
send request to server
user quit command?
server dispatch and execute
send reply back to client
client dispatch and display

Figure 1 activity diagram

The Figure 1 above shows the activity of the lexical scanner when a new client command is executed. In each client and server, the message is dispatch with a dispatch which is constantly trying to dequeue from a blocking queue.

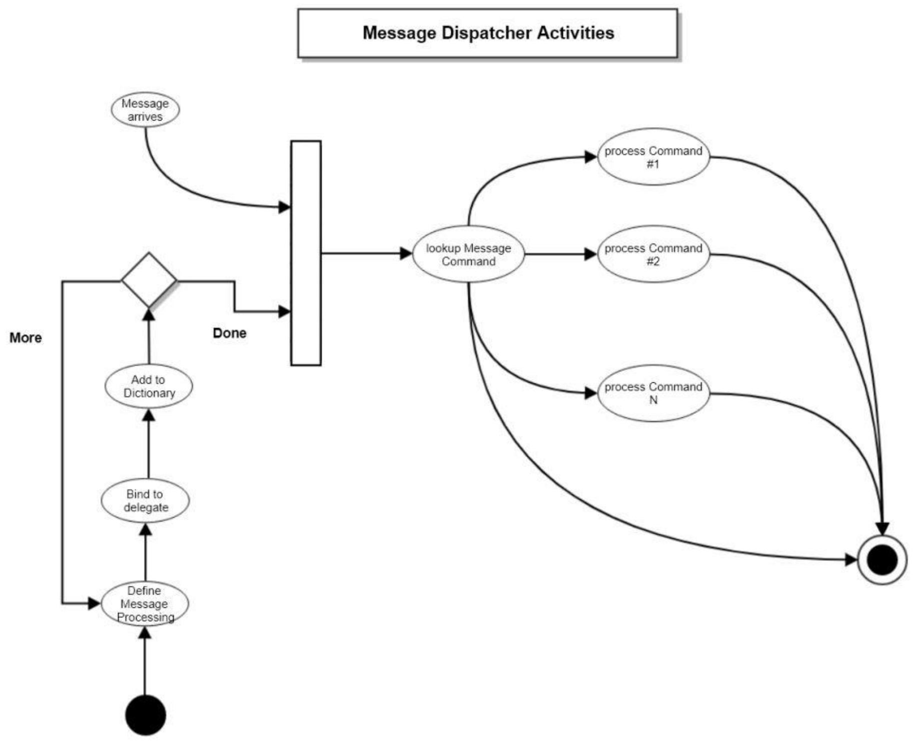

Figure 2 message Dispatcher Activities

The blocking queue in the dispatch makes sure that the thread is running smoothly and wont throws a queue exception.

5. Code Analysis Module Partition

5.1 Overview

The figure3 above shows the package partition of the code Analysis module in the Server.

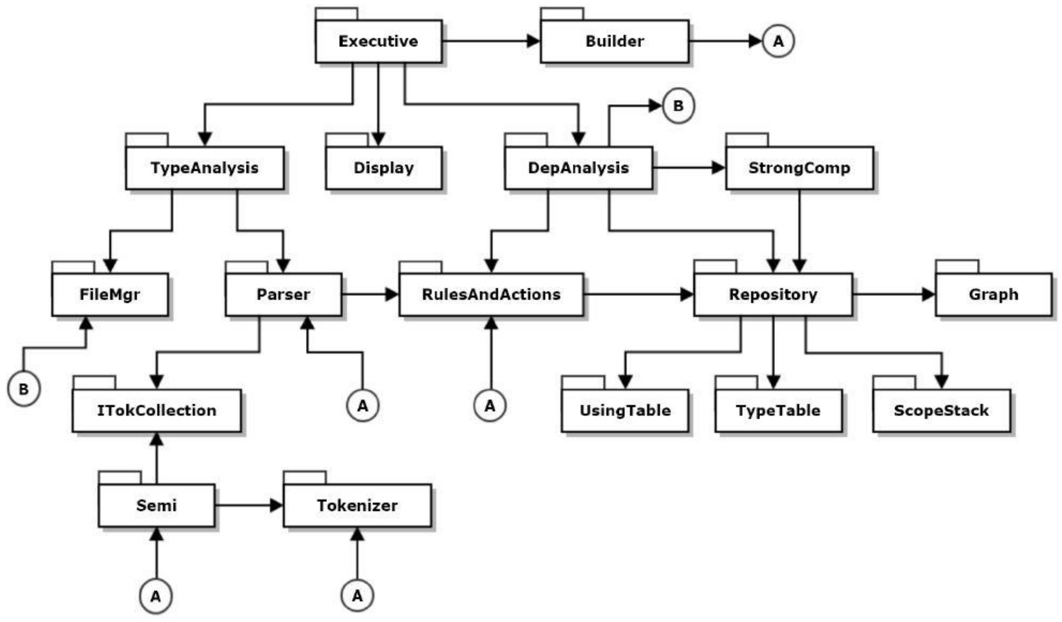

Figure3 overview

The figure3 above shows the package partition of the code Analysis module in the Server.

5.2 Tokenizer

The tokenizer is the essential part of the whole project. It should be able to identify different type of tokens include alphanumeric tokens, punctuator tokens, special one and two character token as well as comments. The partition of the Tokenizer is as bellow.

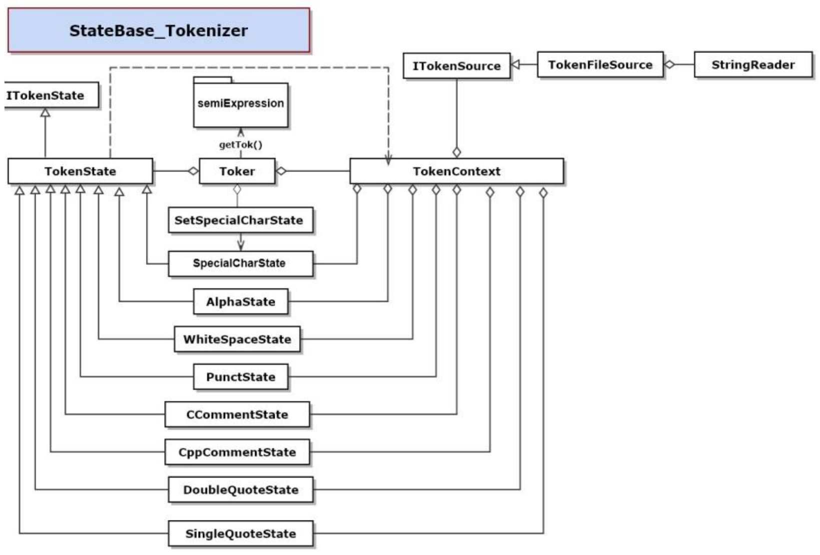

Figure 4 state-pattern Tokenizer

5.2.1 Toker

The Toker class is the wrap of the Tokenizer package. It consists an instance of TokenContext and SetSpecialCharState. SemiExpression and other users should only see and use the functions in Toker class. Users can get the next token or change the target SpecialCharTokens by calling the functions of Toker instance.

5.2.2 ITokenSource & TokenFileSource

ITokenSource is an interface that defines the function of opening a file and create stream. TokenFileSource implements that interface.

5.2.3 ITokenState & TokenState

ITokenState is an interface that defines function that TokenState should have. The TokenState instance in TokenContext class is actually a container of its subclasses instance, it also implement the ITokenState interface by Polymorphism, which means calling its subclass's functions.

5.2.4 SpecialCharState & SetSpecialCharState

SpecialCharState is one of the subclass of TokenState. It provides functions to extract special one or two characters tokens. And SetSpecialCharState class provides the function to change the target special character tokens.

5.2.5 TokenContext

TokenContext class act as a glut to organize the other classes in the package and it is actually the class who do the Tokenizer job. Once it was created by the Toker class, it creates an instance of TokenState and an instance of all the TokenStates' subclass. It also contains a instance of ITokenSource to open a source file. For each new char it peek from the stream, it enters the correct TokenState and call the function in the correct subclass of the TokenState to return the Token.

5.3 SemiExpression

SemiExpreesion package collects the Tokens from the Toknizer and divided them in to collections and each collection of the tokens is called a 'Expression'. It will consists of two components include a 'SemiExpression collector' class and a 'terminalCondition' class.

5.2.1 SemiExpression collector

SemiExpression keeps getting tokens form the tokenizer and appendix it to the expressions list until the terminal condition of an expression is satisfied.

5.2.2 terminalConditon

The terminalCondition class provides a function to make a judgement that if the terminalCondition is satisfied and SemiExpression collector should rely on this to decide when to create a new expression. When analyzing the c# code, the terminal conditions of expressons are usually special tokens include semicolon, open brace and closed brace. But this class should also be able to identify special cases like for(;;).

5.4 Parser

The Parser is a facility that built base on toker and semiExpression to parser files with certain extendable rules and actions.

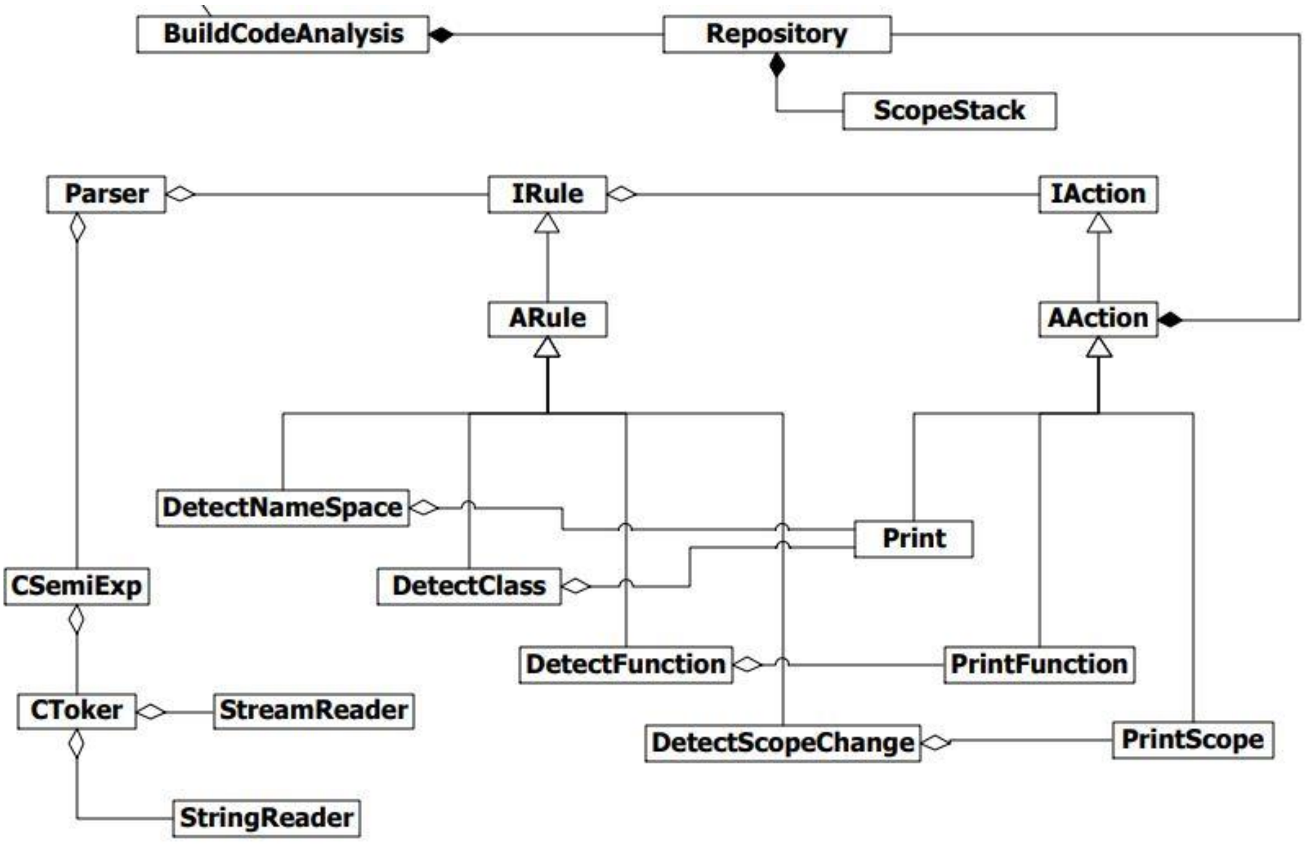

Figure5 parser

5.5 Type Analysis

The Type Analysis package is to build a parser with appropriate rules and actions to parse the files and find all the user defined type. And build a type table to store it.

5.6 Dependency Analysis

The Type Analysis package is to build a parser with appropriate rules and actions to parse the files and find all the types which is used in one file and defined in another. By parsing the files and compare it with the type table, it can find all the dependency between files.

5.7 Strong Component

The Strong Component is a implementation of tarjan algorithm to find all the strong component base on the information of the dependency detected in the Dependency analysis.

6. Critical issues

6.1 Open source file failure

If the Tokenizer fails to open the source file, it will unable to get the stream file and the operation after it will crash and not possibly output the correct result. There are multiple reasons lead to the failure of opening the file such as the path is not correct or the file is already broken.

Solution: If the attempt to open the file fails, the program should stop immediately and throw an exception for user to debug.

6.2 How to peek more than 1 char to identify 2-character tokens

C# don't allow programmers to peek more than 1 character from a stream nor going back to the previous char. So that it could be a problem to determinate whether the token is a 1-char or 2-char token without risking to loss information.

Solution: This problem can be solved by wrap and rewrite the peek function by using a queue. Each we try to peek the character in the stream, we peek the queue first. If the queue is empty, we step forward in the stream and put the characters from the stream into that queue. By this way we are able to peek more than one character without losing any information.

6.3 How to identify special c# expressions such as for loop

Not all the c# expression is supposed to end with an semicolon, open brace or close brace. There are special cases that the terminal condition should be adjusted to omit some of the tokens. For example, the semiExpression should consider the for loop "for(;;)" as a singe expression and omit the semicolon inside it.

Solution: list or the special case in c# expressions and write special terminal conditions for them respectively. Since there are not too much special cases in c# expressions, this can be done in a reasonable time.

6.4 The GUI may struck when the dispatcher is executing commands.

Solution: create different thread to display client and dispatching messages.

6.5 Same class name in different namespace

Solution: also store the namespace name in the type table and design a function to determinate whether it is a dependency by comparing the name space they use.

7. Reference

http://www.ecs.syr.edu/faculty/fawcett/handouts/webpages/cse681.htm
https://www.tutorialspoint.com/uml/uml_basic_notations.htm