PROJECT #1

CODE ANALYSIS(OCD)

CSE-681 SOFTWARE MODELLING AND ANALYSIS (FALL 2018)

INSTRUCTOR: DR. JIM FAWCETT

DECEMBER 5, 2018

NIVETHA RAMACHANDRAN(SUID:724348897)

CONTENTS

Figures

1. Executive Summary

Code Analysis is the process of obtaining certain useful conclusions from a set of inputs provided. This application follows a server and client model, wherein the client raises requests and the server delivers. This is an application that takes certain files as input and performs operations like lexical analysis, type analysis, dependency analysis and determination of the presence of strongly connected components using them. Lexical Analysis is the process that takes in as its input, a string of characters, which it obtains from the set of input files and extracts the lexical content or the tokens from. These tokens are further grouped into sets, using which type analysis is performed to determine the type of the tokens present in the set. Dependency analysis determines if the files are dependent on each other based on the results obtained from type analysis. The presence of strongly connected components among the input files is determined by analysing the dependencies. In this project, the four functionalities specified above are carried out and thus enabling code analysis to be achieved.

The users for the code analysis application would be packages of other software and technical staff. Some of the users of the application are:

- Developers- Developers can use the code analysis application to analyse codes written by other developers.
- Quality Assurance Staff- The results obtained from the code analyser will help the QA team to determine the quality of the project developed.

The users and the uses of the Code Analyser will be elaborated further in Section 3 of this document.

The Code Analysis is implemented using the following components. They are,

- Tokenizer- Takes in sequence of characters as input and generates tokens as output.
- Semi Expression- Takes in tokens as input and groups them into sets called Semi Expressions.

- Type Table- Determines the type of the tokens present
- Dependency Analyser- Takes in the type table as input and determines the dependency between files
- Strongly connected component- Determines the strongly connected components present based on the dependencies determined.
- Message Passing Comm- Asynchronous message passing is done between the client and server using this component.

The components and their working are explained in detail in Section 5 of this document.

The critical issues which would crop up during the implementation of the Lexical Scanner are:

- There could be certain conditions which were missed out while designing a Tokenizer. Due to this, the tokens picked up could be erroneous. Then the further analysis would go wrong too.
- There could also be a possibility that the input to the Tokenizer is in a language other than C# or some other similar language, so due to syntax changes the Tokenizer will not be able to recognise the tokens unless modifications have been made.
- The project requires to look ahead by more than one character in the input string to conclude the state of the application and the kind of token encountered. But C# does not have any provision to look or peek ahead by more than one character.
- The Rules for performing type analysis and dependency analysis are specific to this project. It could be possible that a different scenario of analysis requires a different set of rules. In this case, modifications would be required.

The issues and solutions to these them are mentioned in the later sections of this document.

2. Introduction

Code analysis is an important phase in every software development cycle. No development is said to be successful until it is through with the code analysis phase. Code analysis can be static, that is, performed on some form of source code or object code, or it can be dynamic, that is, performed while the program is under execution.

In this project, functionalities like Type analysis, dependency analysis and strong component determination is done. Type Analysis is done to determine the type of the tokens extracted. Dependency Analysis is done to find out the which files are dependent on each other. Strongly connected components are determined using the dependencies. The Client and server model iis used to implement this application. The Client sends the requests to perform the functionalities and the server takes care of these tasks and returns the replies. The message passing communication service is used to pass the messages to and fro.

The packages are Client, Navigator, Message Passing Comm, type Analysis, dependency analysis. The project will be implemented using C# language with .Net 4.6 framework and Visual Studio 2017. The file needed for input will be provided by entering in a path. The file will be containing a C# source code but the Scanner will be able to accept in Java and C++ source codes as well, with certain modifications.

3. Uses And Users

- Developers- Developers can use the code analysis application to analyse codes written by other developers. The results obtained will help them interpret and understand the logic used better and thus perform modifications. Also, it is not advisable to have many strongly connected components within a project. Thus, the code analysis project will highlight the presence of SCCs' and thus the developers can investigate and resolve the same.
- Quality Assurance Staff- The results obtained from the code analyser will help the QA team to determine the quality of the project developed. As said before, the presence of strongly connected components indicates poorly of the project. Also, the staff can easily see through the various classes, functions, interfaces that have been used. This will speed up their job and enable them to perform quick and easy analysis of the project. Thus, deployment of the project speeds up too.

3. Application Activity

The activity diagram from the server's point of view is as below,

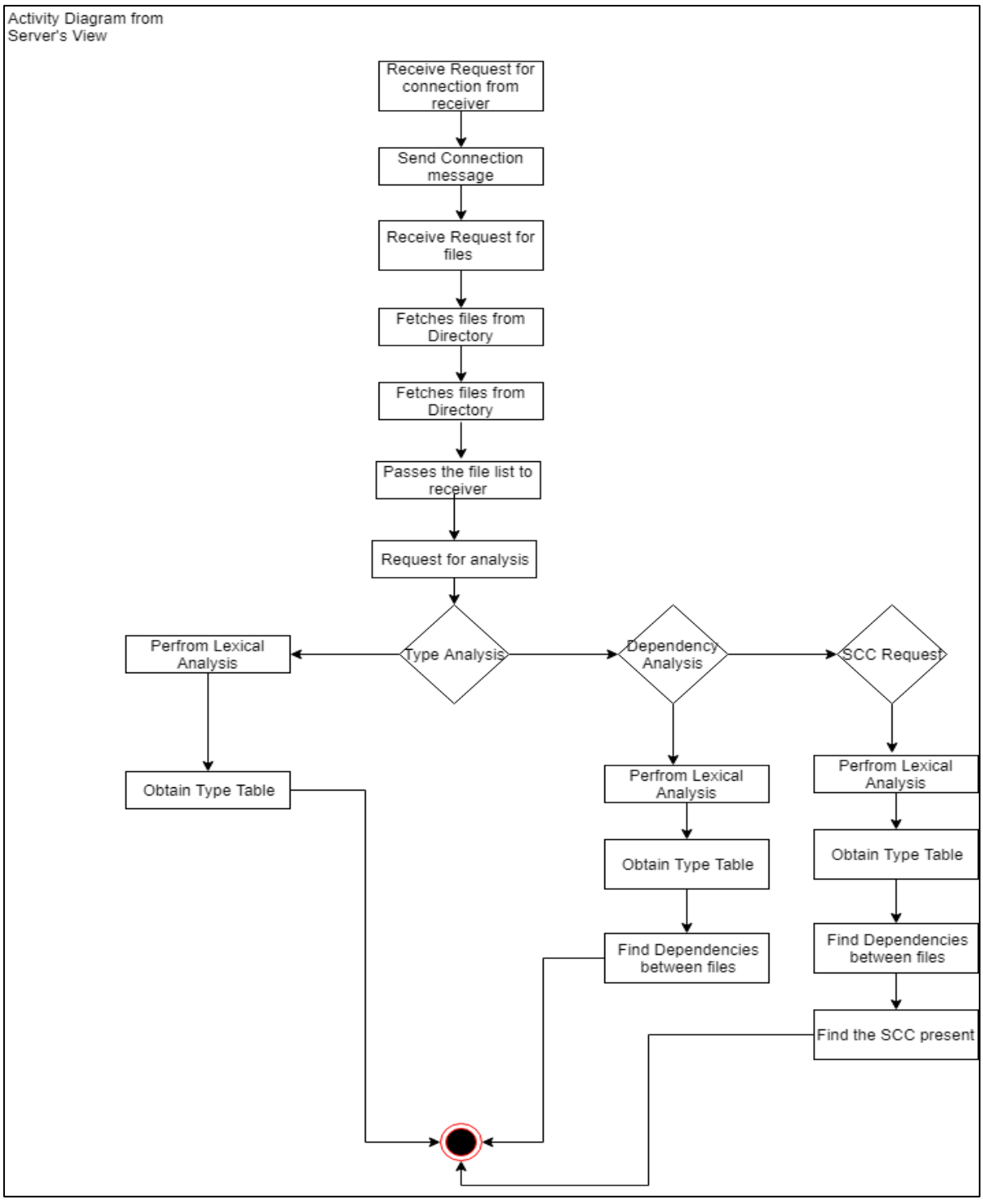

Figure 1: Activity Diagram for Code Analyser

The activities performed in the application are:

- The client raises a request to connect to the server. The message requesting a connection is passed to the server using the message passing comm, a asynchronous

form of message passing. The server receives the message and send a reply stating that connection is established.

- Now, the client requests for the display of the list of files present in the server, which the client needs the code analysis to be performed on. The server receives the message and retrieves the files from the source directories and the various directories within it. The file names are passed as a reply to the client.
- The client performs selection of the files and sends them to the server. The server receives the set of selected files.
- The client now chooses one of the available functionalities to perform: Type Analysis, Dependency Analysis or Find Strongly Connected Components. Once the client has made the choice, the request is sent to the server. The server then performs the required actions to come up with the results for the requests. For example, if the client requests for strongly connected components display. Then the server first performs lexical analysis, the output of which is used for type analysis, then dependency analysis is performed to finally get the strongly connected components present.

5. Partitions

The package diagram of Code Analysis Application is as shown below. It depicts the packages present in the Code Analyser and the flow from one package to another.

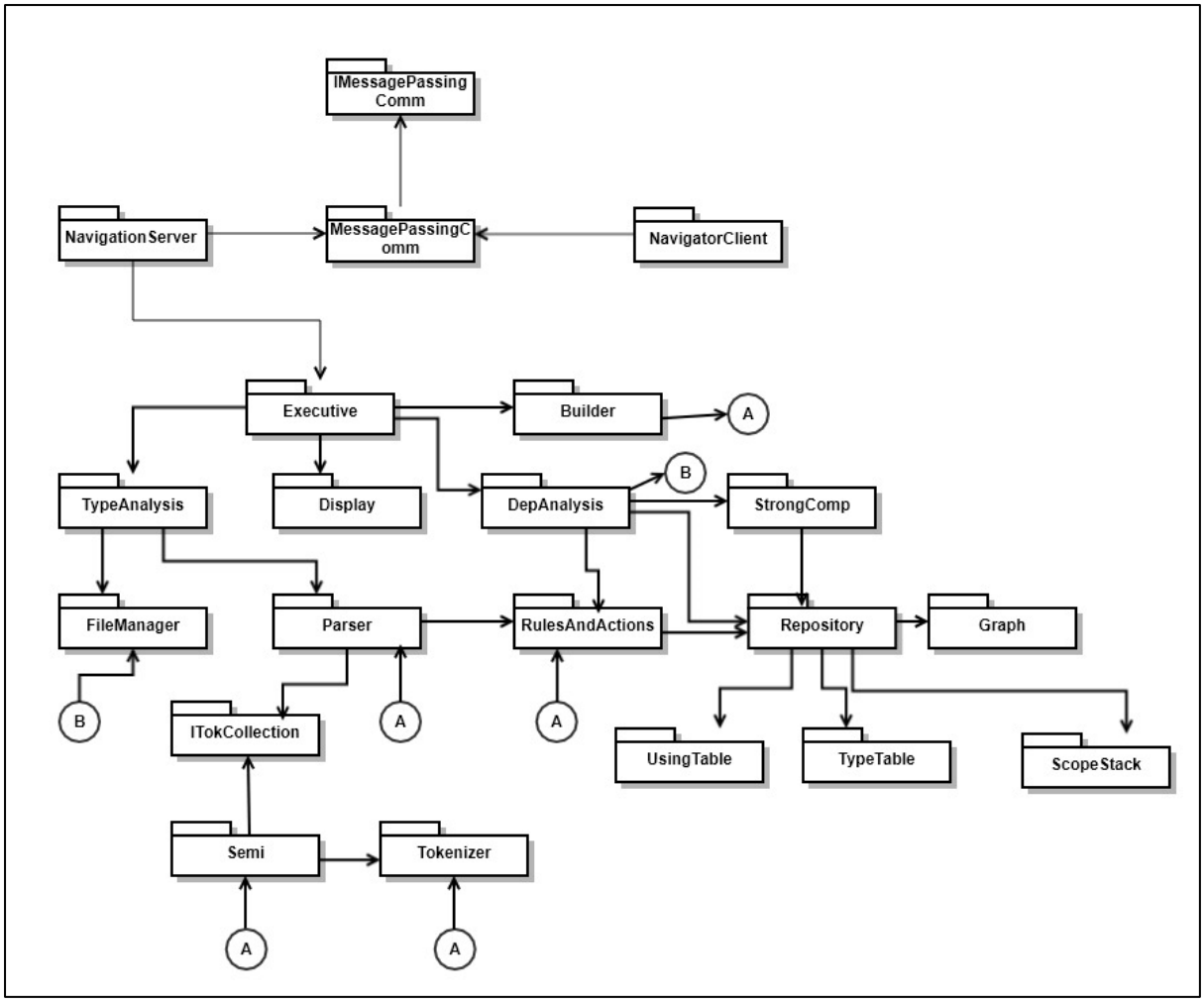

Figure 2: Package Diagram for Code Analysis Application

5.1 Navigator Client

The Client in the Code Analysis application does the job of raising requests to the server to obtain the various outputs required. The Client is provided with a GUI via which the Client chooses the tasks. Based on the task chosen the Comm package is used to pass the request messages to the server. The Client receives the messages and accordingly they are used to populate and display content on the GUI.

5.2 Navigator Server

The Server performs the task of receiving the request messages from the Client. Based on the commands present in these messages, the server calls on the various functions to obtain the ouput. The server calls on the Code Analyser package to perform these functionalities. The output is then sent back to the Client via the asynchronous message passing channel.

5.3 Message Passing Comm

It is an asynchronous message passing service which helps in the passing of requests and replies between the Server and the Client. The Server and the Client each are assigned a Sender and Receiver for the purpose of sending and receiving the messages being passed amongst them.

5.4 Executive

The executive is used to call on the various functionalities of the code analysis applications

5.5 Builder

The builder package does the job of performing parsing and lexical analysis.

5.6 Type Analysis

The package performs the task of determining the type of the tokens and generates a type table for every input file selected by the client. The types of the tokens determined are classes, functions, enums, delegates, structs, namespaces.

5.7 Display

The display package is used to display the various outputs required in the Code analysis application. It is consolidated package having all the functions to perform display of the outputs.

5.8 Dependency Analysis

The package is used to determine the dependencies present between the input files by accessing the type table via the repository. The dependency is determined by using the rules defined in the Rules and Actions package.

5.9 Strong Component

This package determines the SCCs' present in the input files via accessing the Graph through the repository.

The class diagram for Code Analysis is as follows,

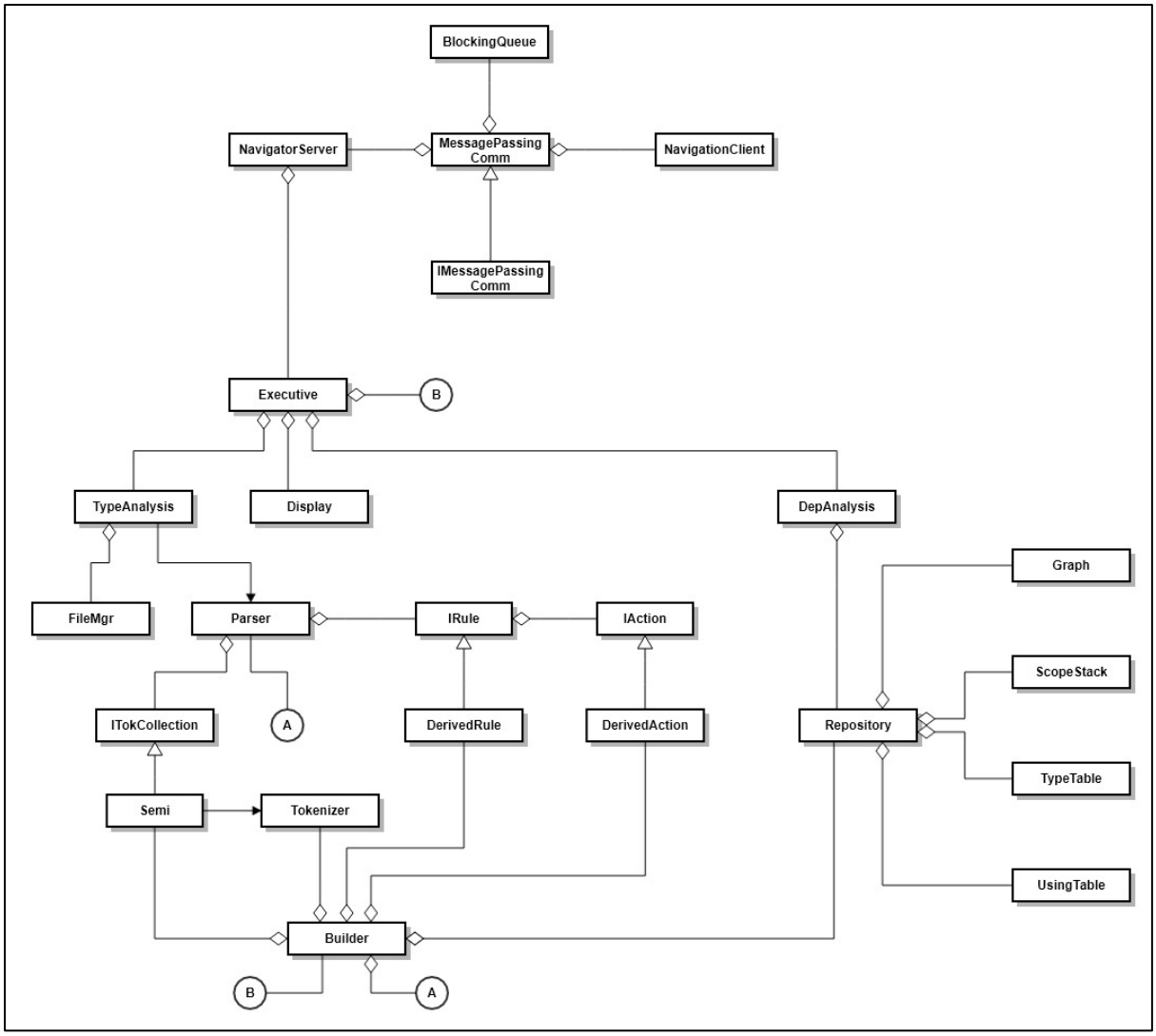

Figure 3: Class Diagram of Code Analyser

The classes and their functions are,

Message Passing Comm class is used to pass messages between the Client and the Server. The client requests for tasks to be performed to the server like type analysis, dependency analysis, strong component determination and connection. The server receives these requests and performs actions and sends replies to the client.

6. Critical Issues

The critical issues which could arise and the solutions to overcome them are as follows,

6.1 Application Specific Requirements

The application requires to look ahead or peek by at least two characters from the input source file, without removing the characters, to determine the current state and to gather the token. However, C# or .Net does not have any inbuilt provision to do so. Also, the Rules and Actions defined are specific only to this application. There could be a possibility that they need to be modified for other applications.

Solution- We can make use of a queue to store the series of characters and look ahead by a defined number of characters to determine the state. The characters wont be removed from the queue but it will be traversed through. The rules and actions need to be written in such a way that modification process is easy.

6.2 Performance and Flexibility

There could be a scenario wherein a totally new set of tokens need to be recognised and extracted. In such a case, making modifications in the existing code would be unnecessary and increase the time complexity. This is because, the conditions in place to extract the previous set of tokens wont be valid to recognise and extract the new set of tokens. Hence, it would be an additional load on the system to execute all of the functions when we know that only the new set of functions would be needed.

Solution- In order to increase the performance, we can use an Interface. Different classes can be developed which will use the same interface but will hold different functions to recognise and extract different sets of tokens. Thus, all the classes need not execute, only the one needed will be run. The implementation of an interface also increases the flexibility of the system because in case lexical analysis needs to be performed on a source file of a different language, then it can be easily done by developing a class which uses the existing interface. Thus, the application can easily be modified and it is adaptive to change.

6.3 Easier to use in the further applications

The main functionality of the code analyser is to generate correct set of tokens in order to come up with the correct set of Semi Expressions. In case there is an error in the token extraction, the further steps can be hindered and it will lead to erroneous results. Thus, we should ensure that the tokens are extracted correctly according to the directives and the semi expressions are generated as per the requirements.

Solution- The solution would be to develop the program in such a way that that the dependency between various classes is less, as in, doing modifications in one class wont affect the functionality of the other. Thus, even if we find that the tokens or the semi expressions extracted are not correct, the testing and the redevelopment will be a smooth process. So, this ensures that the results remain accurate and the other application packages using these results will run smoothly. Also, it should be ensured that the rules and actions specified are accurate and produce the right results.

7. References

The references used for this documentation are,

1) https://www.ijcsmc.com/docs/papers/October2014/V3I10201499a3.pdf
2) https://ecs.syr.edu/faculty/fawcett/handouts/webpages/CSE681.htm
3) https://www.tutorialspoint.com/uml/