CSE 681 SOFTWARE MODELING AND ANALYSIS

INSTRUCTOR: JIM FAWCETT

PROJECT 4

CODE ANALYSIS TOOL: OPERATIONAL CONCEPT DOCUMENT

CHANDANA RAO

CONTENTS

EXECUTIVE SUMMARY

Software projects generally consist of multiple modules, and most of these modules are interdependent on each other. But excessive dependency between the modules will weaken the project. Say, if one of the modules crashes, then all other modules dependent on the faulty module will be affected. A dependency analysis must be performed to make sure that there are no expensive errors in the project. If a dependency analysis is performed at an early stage, the developer will know on how to develop the future modules and not indirectly affect the previous ones.

A code analysis tool is a software tool that analyses the code dependency between the set of modules or files and returns the analysis report. The purpose of this project is to build software tools for code analysis and this document represents the operational concept for the same. In this project, we are building a tool that

- Performs Lexical analysis on each source file using tokenizer and semi-expression
- Analyses all the different types in each file, based on the syntax
- For a given set of files, it performs dependency analysis on each of the pairs
- Finds the strong components, i.e., the files that are connected in a set of files, and return them.

The intended users for the code analysis tool are:

- Developers: to manage the code efficiently, and avoid more dependencies
- Quality Analysis Team: to check the performance of the project
- Software firms: to monitor the ongoing projects

In the further sections of this document, critical issues and the solutions are discussed. Adding to that, the inter module interaction between the modules in the tool, and package and activity diagrams are also explained.

INTRODUCTION

A code analysis tool is a software tool that analyses the code dependency between the set of modules or files and returns the analysis report. This project is built on an asynchronous client server model. The client requests for a service, and the server responds to it. Here, in this tool, the client requests for the files from the server and asks for the type analysis, dependency analysis, and strong component. The server performs the operation and replies the output to the client. This is the basic idea on the Code analysis tool. In the analysis part, the server performs lexical analysis, type analysis, dependency analysis, finds the strong component between a set of files. Lexical analysis includes generation of tokens using tokenizer and generating the semi-expressions from the tokens obtained. This is then used for the further analysis.

APPLICATION ACTIVITIES

Figure 1 shows the sequence of activities performed from the server side for the code analysis tool.

- The server receives the request from the client.
- The server connects to the client and sends a reply message
- The server receives a request for the files in the server
- It fetches the files from the directory and sends the list to the Client.
- The client then selects the files from the set of input files for performing the analysis
- The client can request for type analysis, dependency analysis, or strong component.
- Based on the type of request, the server performs the required set of operations and returns the results to the client.
- The client can select and deselect as many files from the list, and request for the analysis.
- It closes only when the GUI is closed.

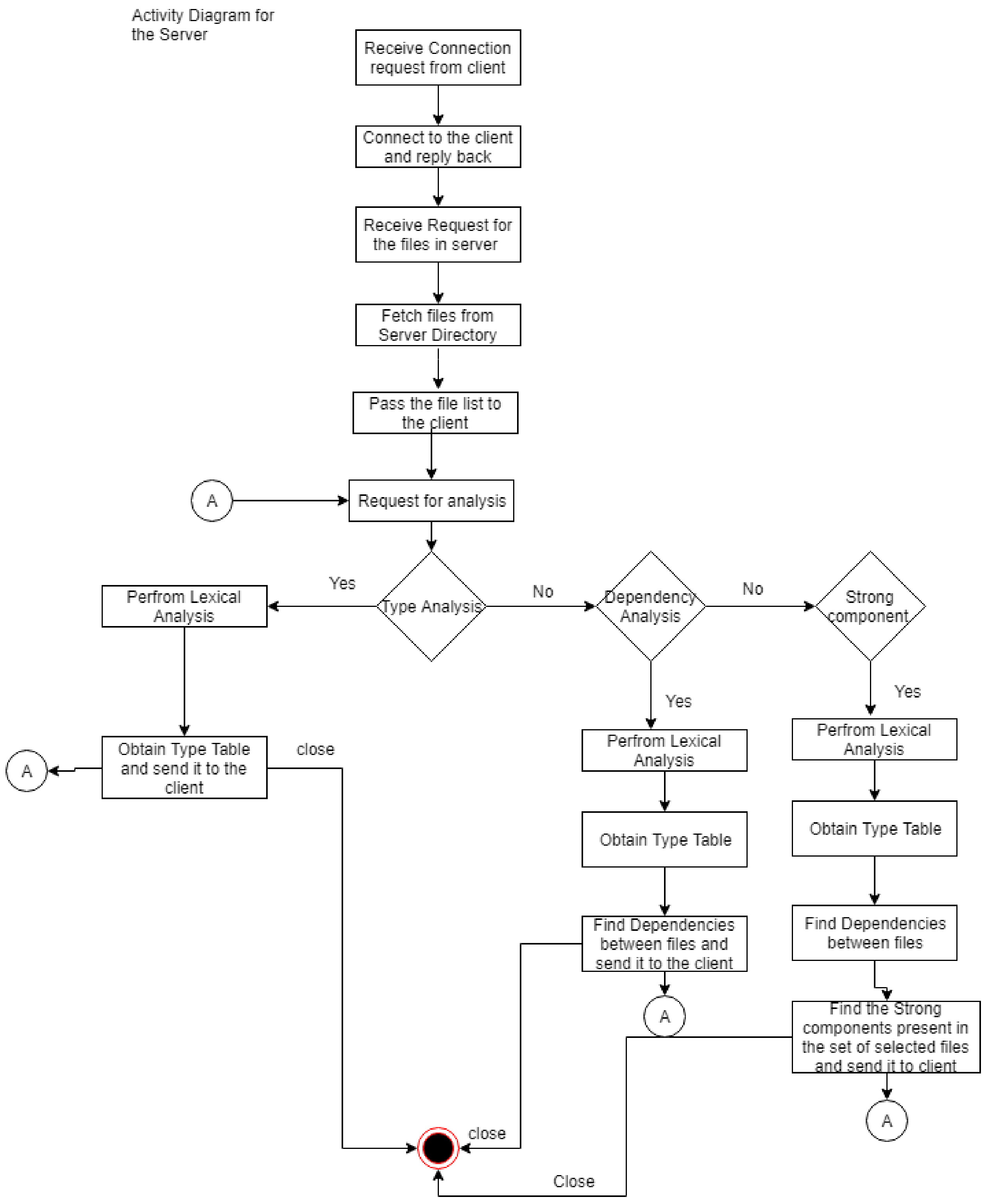
Activity Diagram for the Server
Receive Connection request from client
Connect to the client and reply back
Receive Request for the files in server
Fetch files from Server Directory
Pass the file list to the client
A
Request for analysis
Type Analysis
Yes
No
Perfrom Lexical Analysis
Dependency Analysis
No
Strong component
Yes
Yes
Perfrom Lexical Analysis
Perfrom Lexical Analysis
A
Obtain Type Table and send it to the client
close
Obtain Type Table
Obtain Type Table
Find Dependencies between files and send it to the client
Find Dependencies between files
A
Find the Strong components present in the set of selected files and send it to client
A
close
Close

Figure 1. Activity diagram for the server

PARTITIONING

Figure 2 represents the package diagram for the code analysis tool.

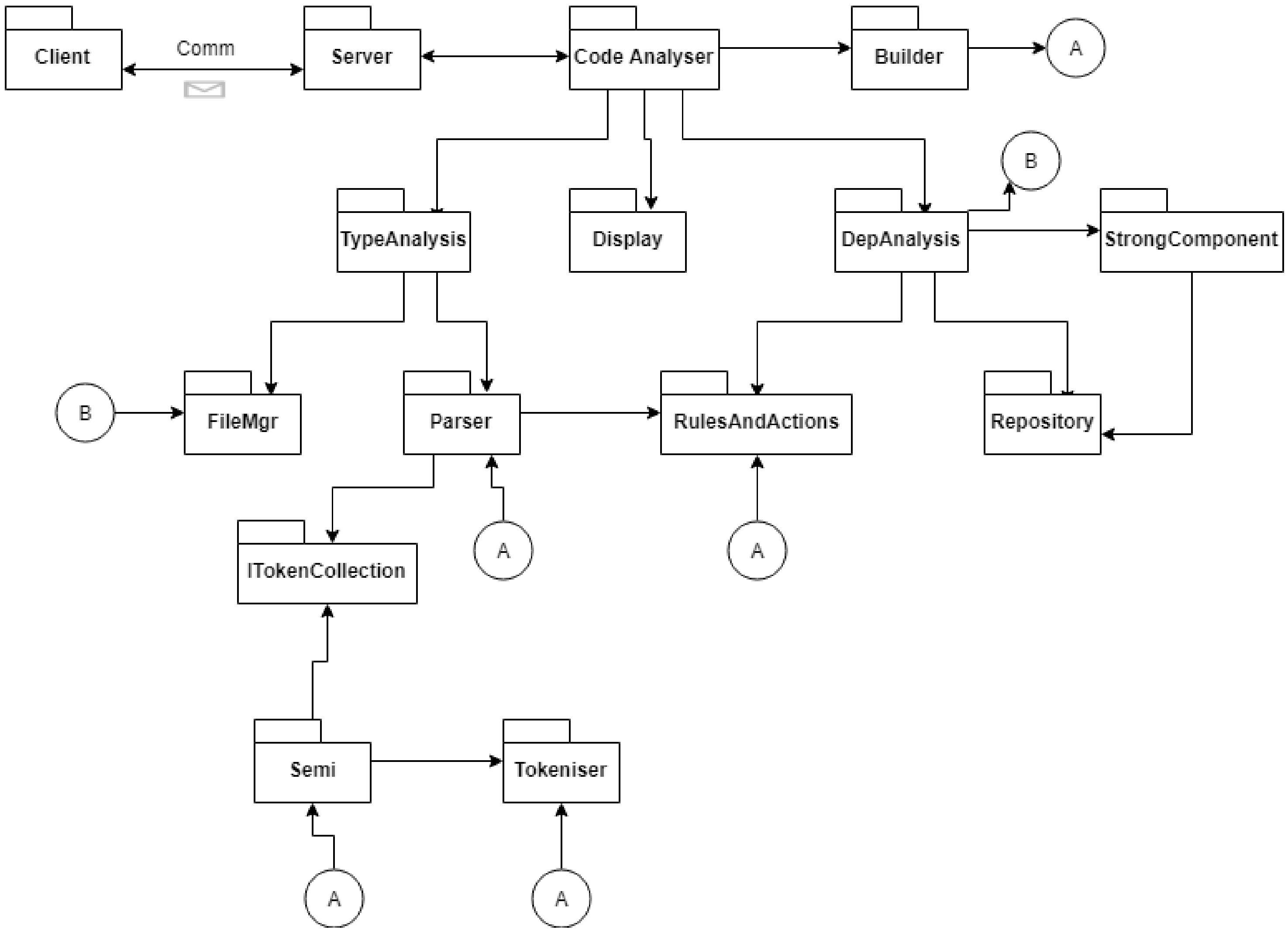

Figure 2. Package diagram for Code analysis tool

TOKENISER

This package performs the lexical analysis of extracting the tokens from the source file. It identifies the tokens like alphabets, numbers, punctuators and white spaces. The token boundaries are white space characters, new line characters, comments, string boundaries, and the separation between alphanumeric and punctuator characters.

SEMI:

This package groups tokens into meaningful sets that contain the information needed to analyze the code. It ignores white spaces and comments, and the remaining tokens are analyzed against the grammatical constructs of the program.

ITOKENCOLLECTION:

This package collects the tokens semi expressions from their packages

PARSER:

This package defines the rules and actions for code analysis.

RULES AND ACTIONS:

This package defines the rules that each have a grammar construct detector, and also a collection of actions for each rule.

REPOSITORY:

This package is used in passing the data between the actions.

TYPEANALYSIS:

This package performs the type analysis for each file and returns type table that contains the grammatical constructs of the program, like namespace, class, delegates, structs, enums, and functions

DEPANALYSIS:

This package takes in the type table, and checks which files are dependent on each other, for a particular set of files.

STRONGCOMPONENT:

This package finds the strong component between a set of files.

DISPLAY:

This package displays all the output from the code analysis.

FILEMGR:

This package contains the path of all the files in the server.

CODE ANALYSER:

The server makes use of this package for the code analysis operations.

SERVER:

This package handles all the messages that are given by the client and performs the required operations. It also gives back the output through message passing.

CLIENT:

Using this package, the users can communicate with the system. The tool takes in the input, sends it to the server and fetches back the output for the clients. This is done through Communication messages.

USES

- The code analysis tool is used in analyzing the dependencies between the modules in a software project.
- It is used for the websites, to analyze the interconnection between the webpages, and how to handle them
- The code analysis tool helps in maintaining the software quality issues during the development stage.
- The tool can be extended to different applications, like detecting a pattern using the keywords, checking if a document has met certain standards, extracting useful information from a pile of webpages or documents with the help of tokenizer.

CRITICAL ISSUES

- Handling multiple rules for detecting the grammatical constructs in type analysis can become complicated
  Solution: We make use of Rules and actions, that handle each rule and the corresponding action separately
- Passing multiple messages across client and server would take time and result in timeout

Solution: The messages are passed asynchronously

- Expanding the code analyser to different languages

  Solution: Since we make use of single responsibility principle, it is easier to add the rules for the analysis in lexiacal analyser, type analysis, and dependency analysis

REFERENCES

- https://ecs.syr.edu/faculty/fawcett/handouts/CSE681/presentations/SoftwareStructureResearchExcerpts.pdf
- https://ecs.syr.edu/faculty/fawcett/handouts/CSE681/Lectures/StudyGuideOCD.htm
- https://ecs.syr.edu/faculty/fawcett/handouts/CSE681/Lectures/CSE681codeL2.htm
- https://ecs.syr.edu/faculty/fawcett/handouts/CSE681/Lectures/Project4-F2018.htm