Operational Concept Document (OCD)

Remote Package Analyzer

Developed by: Vaibhav Kumar (SUID: 689190843)

Course: Software Modelling and Analysis (SMA CSE-681)

Instructor: Dr. Jim Fawcett

Date: 05th December 2018

Contents

Executive Summary

Incremental development has been one of the most popular and recommended software development methodology due to its various cost and time advantages. In any software system (henceforth referred to as system) that follows incremental development, every module is tested individually before it is integrated to the software baseline (or existing system) which is then followed by Integration testing. This whole process certainly reduces rework by making it easy through **code analysis** at every stage, consequently reducing the time and cost of rigorous rework involved in identifying the errors in the system if any. This code analysis is one of the many advantages of the remote package analyzer that we are going to build.

The question that arises here is why we need this remote package analyzer if we already have various good parsers like Lex, bison etc. already available? One simple and very logical answer to this is that the complexity and cost involved in deployment and portability of these existing parsers is much more than the benefits we are getting by using them just to do code analysis. Hence, providing ease of deployment, portability and future extensibility are some of the goals that we are primarily focused on while developing this system.

Apart from its usage in building code analyzers, the Abstract Syntax Tree (AST) (prospective implementation) generated by the remote package analyzer can be used in evaluating code metrics by other software systems. This lexical scanner can also be used for reviewing code for software/data security by an organization.

Introduction

The goal of this project is to develop a remote package analyzer using state-based tokenizer.

The remote package analyzer will be developed on .NET 4.7.2 framework with the use of C# 7.0, WCF and WPF. However, the extensibility and interoperability with different technologies will be considered.

Organizing Principle and Key Architectural Idea

Partitions

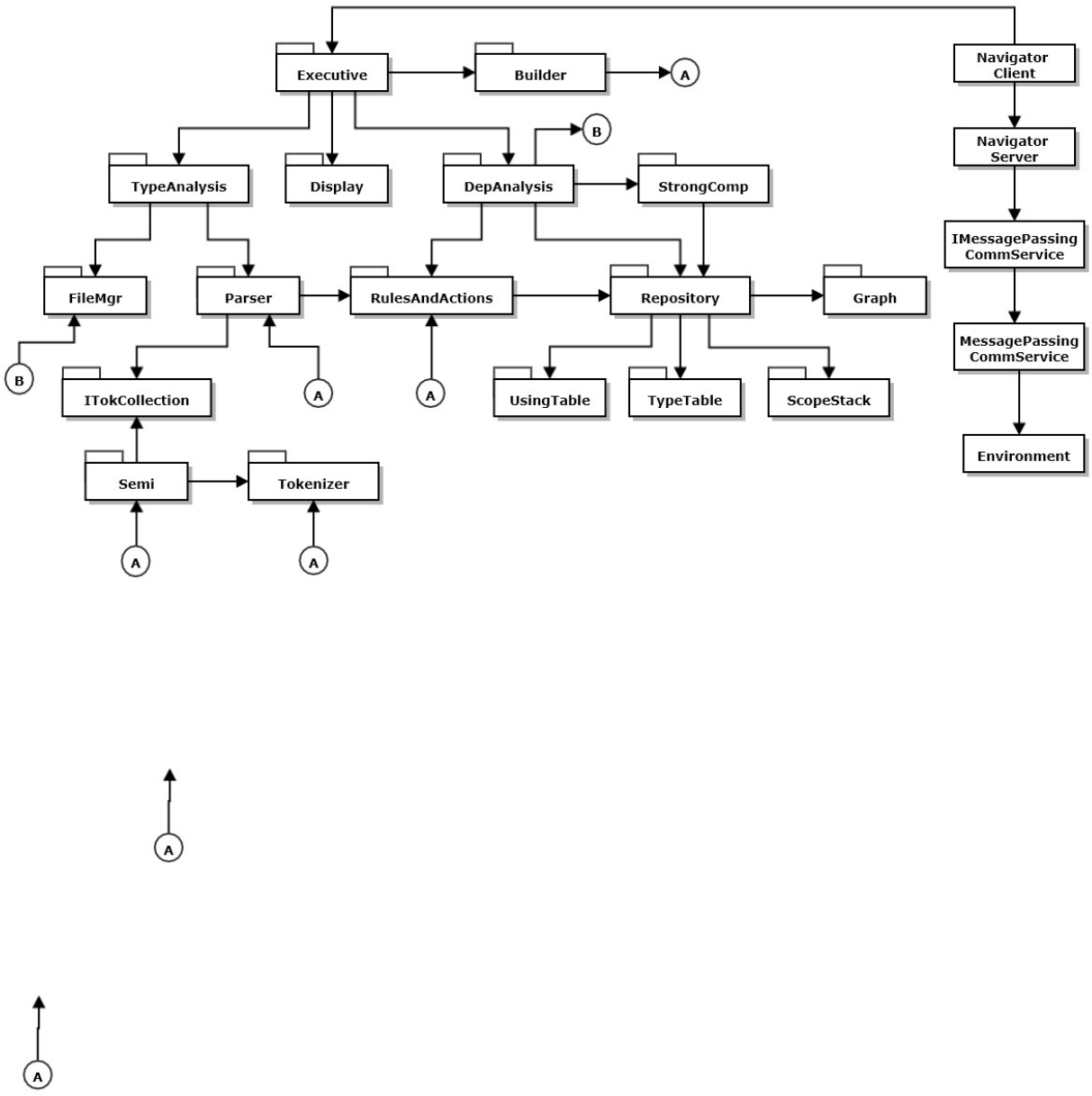


Fig 1: Overall Package Diagram of Lexical Scanner

- **Client Display:** Client display consists of user interfaces through which user interacts with the application.

- **Client Executive:** Client Executive is the entry point of the application which interacts with the scanner server.
- **Code analyzer Server:** Scanner server is the package which converts the input code file into stream of characters and passes this stream to the tokenizer.
- **Message passing comm channel:**
- **Parser**
- **Tokenizer:** Tokenizer package receives the input character stream from the scanner server and follows certain rules of terminating characters to form tokens which is then supplied to semi expression package.
- **Semi Expression:** Semi expression package repeatedly calls GetTok() method to read tokens. Semi expression package then follows certain rules of terminating character to form meaningful information for code analysis.
- **Common Utilities:** Common Utilities package provides reusable utility methods which can be used by any other packages.

Class Diagram

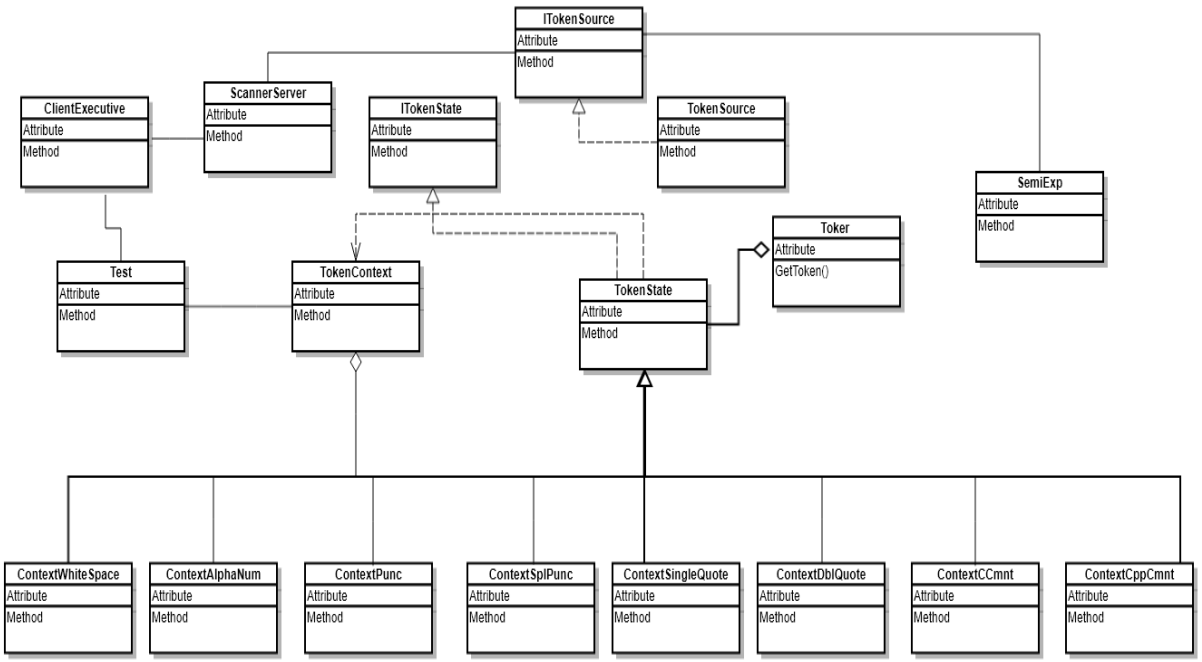


Fig 2:

Application Activities

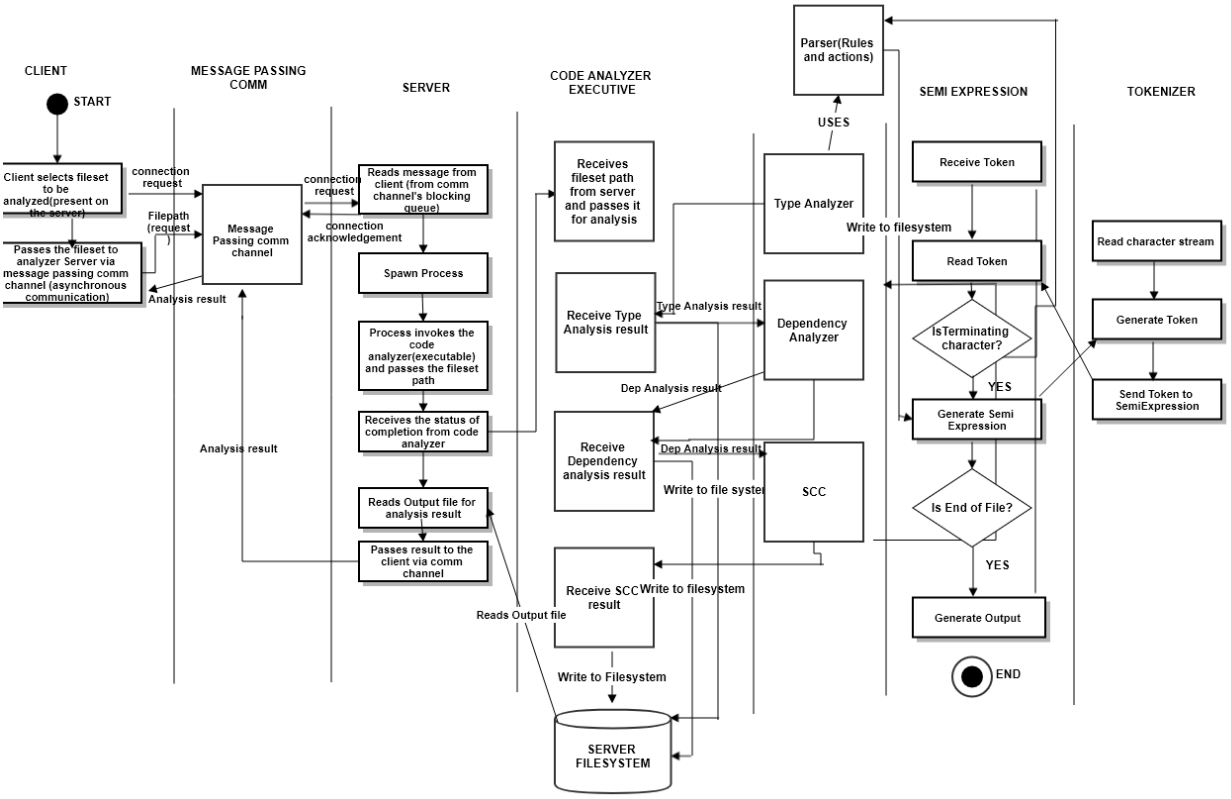
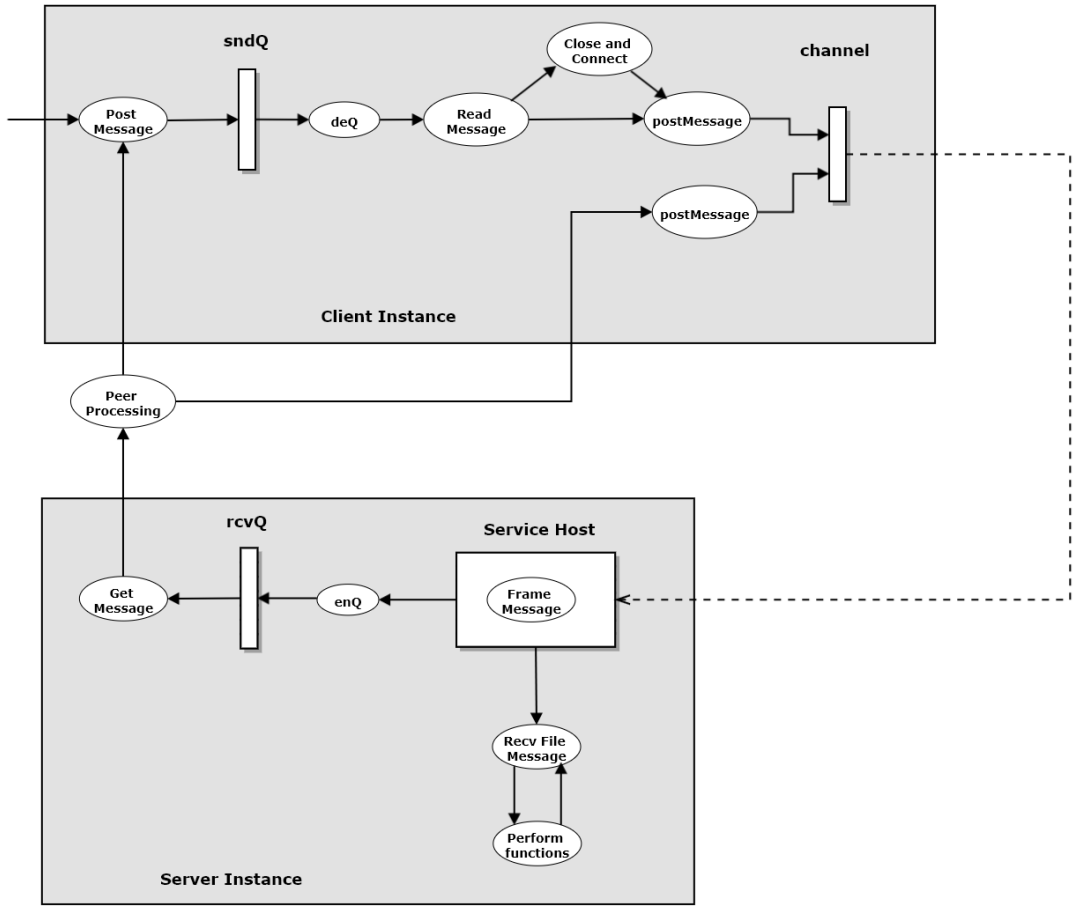
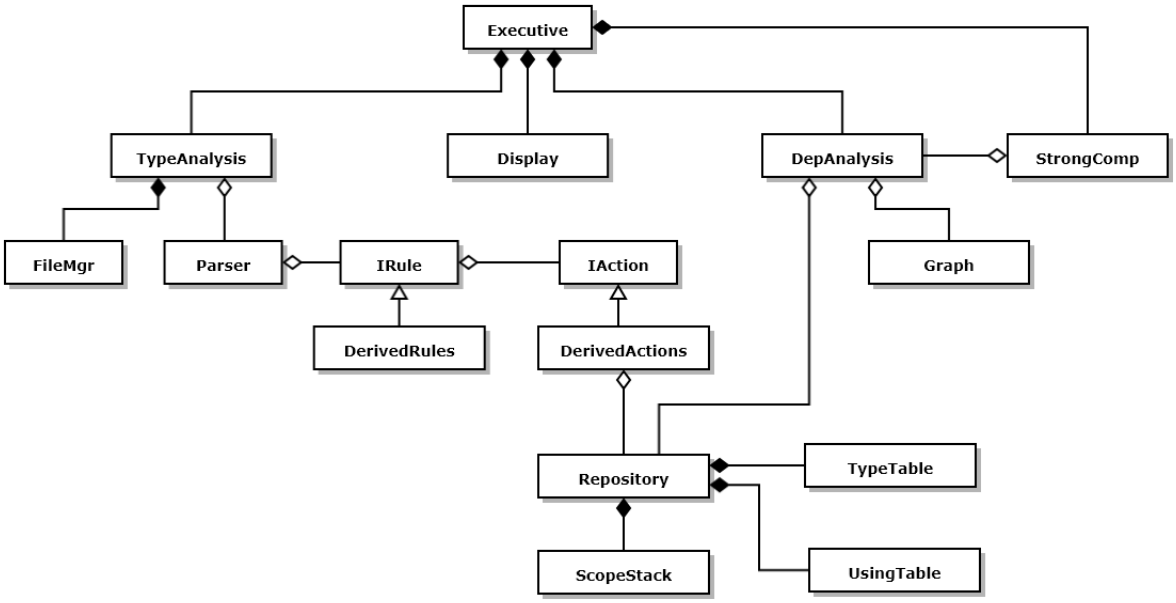


Fig 3. Overall Activity Diagram of the remote package analyzer


sndQ
Close and Connect
channel
Post Message
deQ
Read Message
postMessage
postMessage
Client Instance
Peer Processing
rcvQ
Service Host
Get Message
enQ
Frame Message
Recv File Message
Perform functions
Server Instance



Major components

GUI

Interactive client display to receive input by navigation and to display the analysis result

Message passing comm channel

Implements asynchronous message passing communication using blocking queue. It is the communication channel through which remote clients and server are connected.

Server

Spawns a process to invoke code analyzer executable.

Type analyzer

Analyses all the user defined types in the input file.

Dependency analyzer

Analyses all the dependencies among input files using type analysis result.

Strong components

Analyses all the strong components based upon dependencies.

Parser

Generates type tables using certain rules and associated actions.

Tokenizer

The Tokenizer package is responsible for generating tokens which can further be used by Semi Expression package to generate semi expression. A tokenizer reads through the input character stream. As soon as any terminating characters like whitespace, transition from alphanumeric, a punctuator etc. is encountered, it generates a token. This whole process is carried out till end of stream is reached.

The Tokenizer package contains a Toker class that is responsible for extracting different types of Tokens such as alphanumeric tokens, punctuator tokens, special character tokens, single and multiline comments etc.

Here is the activity diagram of the Tokenizer package

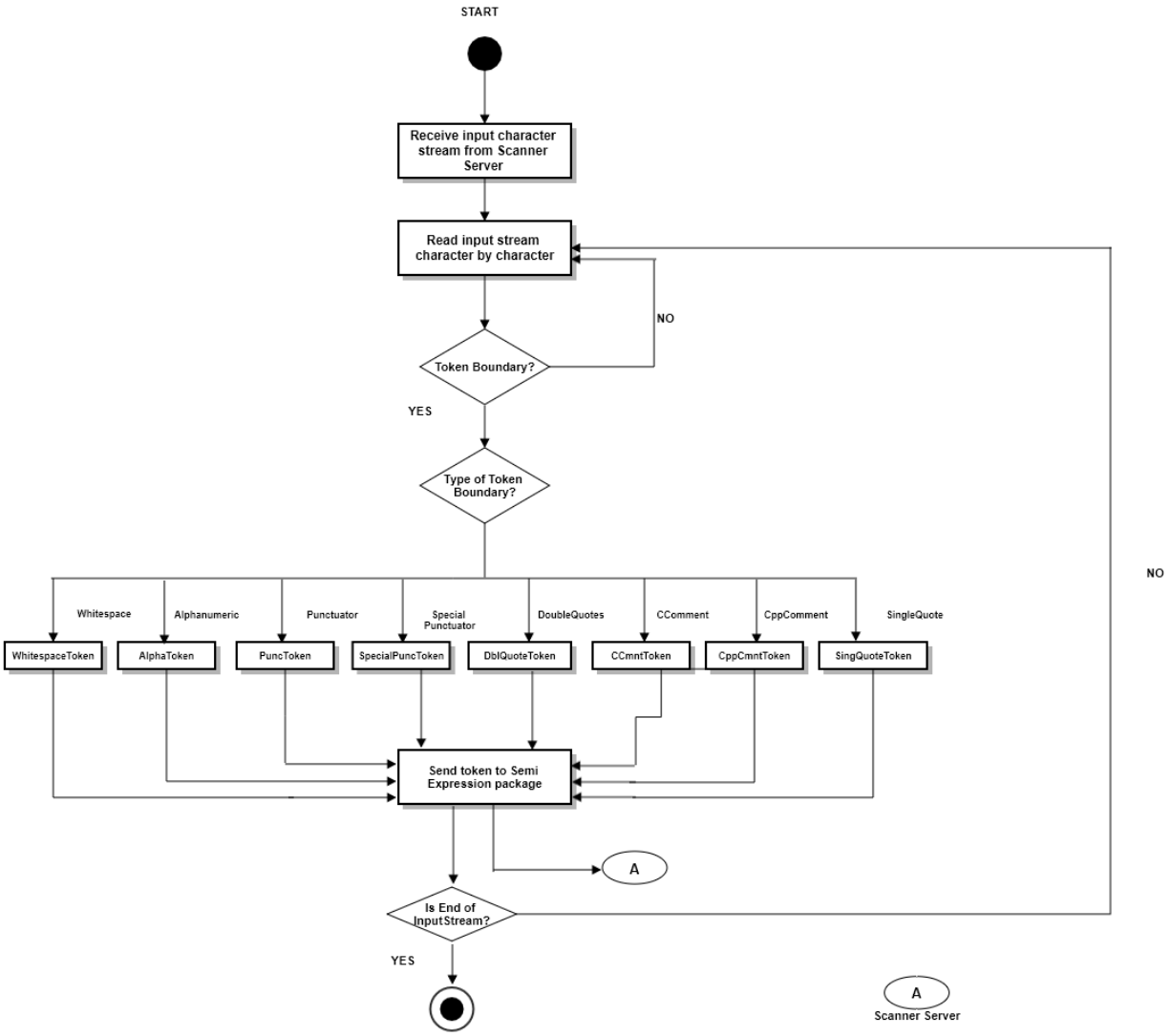


Fig 4: Activity Diagram of a Tokenizer

- The scanner server provides input character stream to the Tokenizer package.
- The Tokenizer package receives the stream from scanner server and starts reading.
- The Tokenizer package has the Toker class that reads the stream character by character and keeps on reading until a token boundary is encountered.
- Token boundaries can be transition to alphanumeric characters, single or multiline comments, punctuators etc.
- Depending upon the previous and new token boundary read, a new token is generated and characterized.
- Newly generated token is then sent to the SemiExpression package.

Semi Expression

SemiExpression package is primarily responsible for generating the meaningful information that is necessary for analysis. It mainly consists of SemiExp class which gathers tokens to form a meaningful piece of information necessary for analysis.

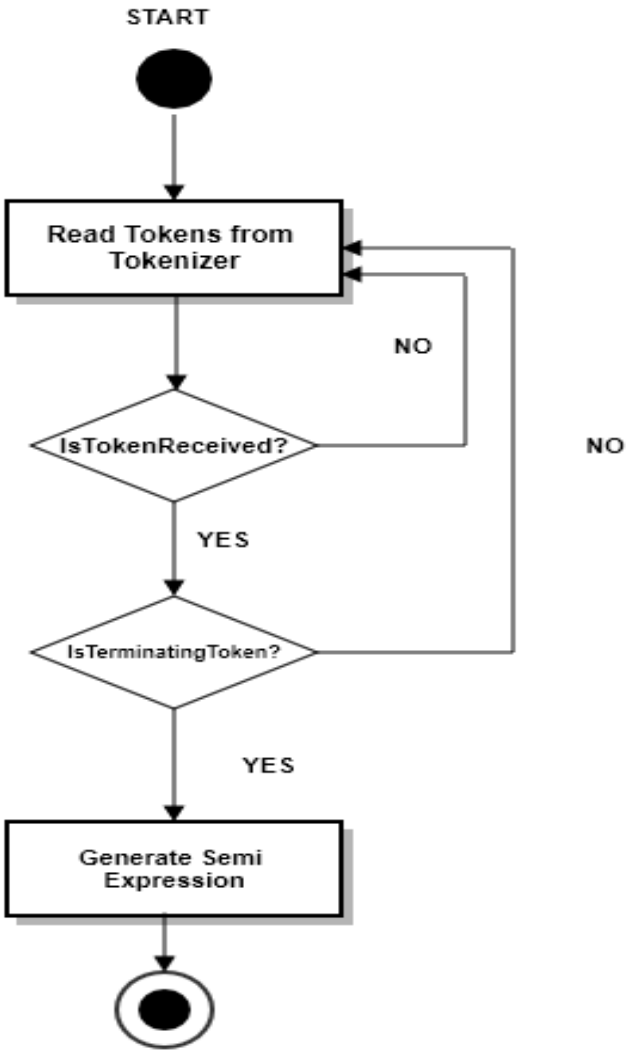


Fig 5: Activity diagram of SemiExpression package

- The SemiExpression package repeatedly calls the GetToken() method of Toker class to read tokens.
- As soon as token is generated by Tokenizer package, SemiExpression package reads it.
- SemiExpression package continues to read the tokens until a terminating token is read.
- Depending on terminating token, a meaningful piece of information or Semi Expression is generated and provided as the output.
- This process keeps on going till end of the file is reached.

Test Harness

The lexical scanner will also have a Test harness package which will provide test suite of test cases to run the scanner. Sample test case scenarios will be created throughout the development process.

User and Uses

Code Analyzer

The lexical scanner will be useful in code analyzers. In this case, the code analyzer software will be the user which uses the lexical scanner to generate Abstract Syntax Tree which further is used to run analysis algorithm and generate relevant reports.

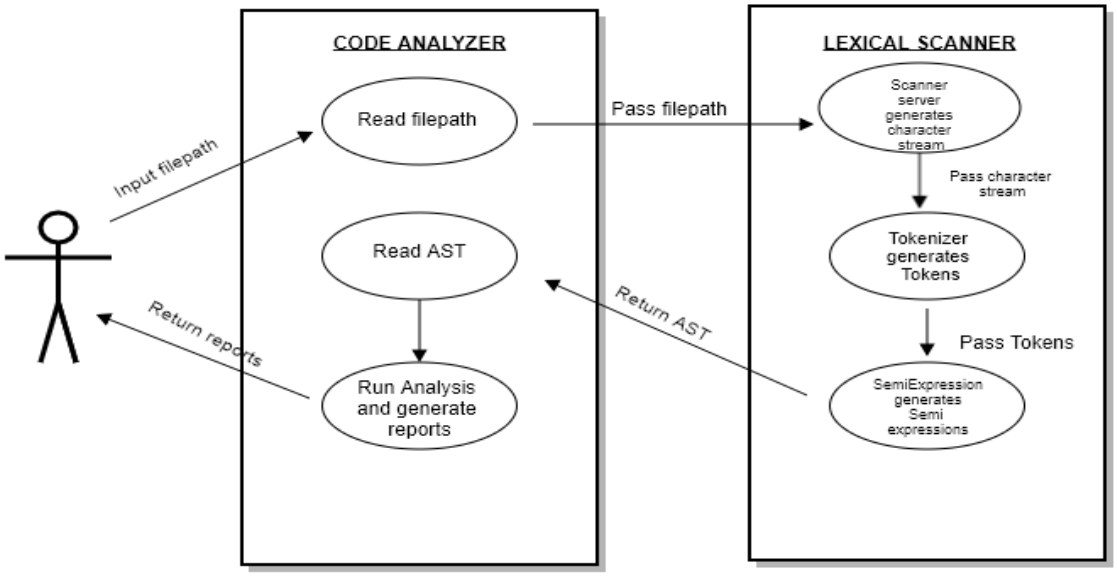


Fig 6 : Use Case diagram for code analyzer

Above is the use case diagram of an analyst using scanner through code analyzer for code analysis.

Use Case Scenario (Remote package analyzer)

- The analyst provides the filepath to the GUI
- Filepath is passed to the server via message passing comm channel.
- Server spawns a process to execute code analyser and passes the filepath.
- Code analyser runs analysis and writes the result to the server file system
- Server reads the output file and sends the output to the client via comm channel.
- Client displays the output to the GUI

Code review

The lexical scanner can be used by any software development organization's code review team for checking the coding standards such as naming conventions, complexity, code dependencies etc.

The code reviewer of the organization can read the AST generated by the lexical scanner and evaluate the metrics and take necessary actions if needed to improve the code. These code review can also help in understanding the security threat in the code.

Compilers

Lexical analysis is one of the primary responsibility of any compilers. Compilers are responsible for *translation* of one language to another language (e.g. machine language). This makes it very important for compilers to have lexical scanner. The lexical analysis by any compiler involves the generation of lexemes, which are the character sequence matching the tokens.

Linter Tools

Linter tools (or Lints) are the tools used for identifying programming errors. Linter tools easily identify the warning, error and threat in a code file. This lexical scanner thus can be used along with the linter tools to improve the code constructs and remove threats.

Other Software Packages

As already explained earlier, our lexical scanner can be integrated with other software packages to help those software packages to carry out their tasks.

For instance, our upcoming projects 3 (type based package analysis) and 4 (remote package dependency analysis) will be using our implementation of lexical scanner developed in project 2.

Critical Issues

Correctness

The primary objective of the lexical scanner is to provide meaningful information (in form of Abstract Syntax Tree) that can be used for analysis. Hence, the correctness of output is of the primary issue of the lexical scanner. This makes it much important to *follow correct and precise grammar required for parsing*.

Deployment

Ease of deployment and operation was one of the first motivation to develop this lexical scanner even though there are very good lexical scanner like Lex, Bison are already present.

Since this scanner is dedicated towards a specific purpose, it is not a very big software package which makes its deployment and maintenance easy and cost and time efficient.

Interoperability

The output generated by lexical scanner is primarily used by other software packages or applications. These other software packages could be developed using other software platform and programming languages. This makes our lexical scanner's interoperability as one of the most critical issue.

Since we are developing this application on .NET 4.7.2 platform, the presence of Component Object Model (COM) will help us in developing an interoperable lexical scanner.

Server Load and resource utilization

Resource utilization is an important issue when it comes to software development and its implementation.

Since the lexical scanner is developed for a specific purpose, it is not a big application. Unlike other bigger scanners like Lex, the load on server and memory utilization by lexical scanner will be minimal and optimal. Following better coding standards and writing optimal codes can also help in reducing resource utilization.

Extensibility

Increasing future applications of the lexical scanner is an important provisioning idea while developing this application. Extensibility of the lexical scanner to allow future growth hence becomes an important consideration.

Following the "***Singularity principle of software development***" and dedicating specified responsibilities to each package makes it easier to embrace future changes without any major complexity and rework.

Robustness

Ability of any software system to cope up with unpredicted future failures is robustness of a system.

Developing smartly with a wider set of failures forecasted and ability of the system to handle exceptions makes our lexical scanner more robust. But there are many situations or failure conditions that can't be predicted before deployment and operation. So the

application may face some failures in future. Ability of the software packages to accommodate future changes required to handle unpredicted failures makes it easier to handle future failures after deployment.

Conclusion

As an essence, we can say that the lexical scanner developed with a specific purpose can prove to be a very useful software system. Wide range of applications also make it important to make this lexical scanner scalable and extensible.

Using precise business logic and grammar for developing the scanner is an important point to take into consideration. The Abstract Syntax Tree generated from the scanner has to be structured and well formatted.

Hence the two core software packages in this implementation are the Tokenizer and the SemiExpression packages. Apart from the abovementioned two packages, a very well designed test package is also an important component of this application.

References

1. https://en.wikipedia.org/wiki/Lexical_analysis
2. https://medium.com/dailyjs/gentle-introduction-into-compilers-part-1-lexical-analysis-and-scanner-733246be6738
3. https://en.wikipedia.org/wiki/Lint_(software)
4. Project help content and handouts provided by Dr. Jim Fawcett on his website.