

# **Remote Package Dependency Analysis**

## Operational Concept Document

CSE681 Project #4

Yuxuan Xing

SUID:486360997

Instructor: Jim Fawcett

Date: 3th, December 2018

1. Executive Summary	4
2. Introduction	7
2.1. Application Obligations	7
2.2. Organizing Principles	8
2.3. Key Structural Ideas	8
3. Uses	9
3.1. Myself	9
3.2. Instructor and TAs	9
3.3. Developers	9
3.4. Project Managers	9
3.5. Customers	10
4. Partitions	10
4.1. Tokenizer	10
4.2. SemiExpression	11
4.3. Parser	12
4.4. TypeTable	12
4.5. TypeAnalysis	13
4.6. DepAnalysis	13
4.7. StrongComponents	13
4.8. MessagePassingService	14

4.9. Client	14
4.10. Server	15
4.11. TestUtilities	15
5. Application Activities	16
5.1. Input source files or directories	16
5.2. Server accept message from Client	16
5.3. Extract tokens	17
5.4. Group tokens into sets	17
5.5. Build TypeTable	17
5.6. Do Dependency Analysis.	18
5.7. Find all Strong Component	18
5.8. Handle the exception	18
5.9. Pass the reply message with result back to Client	19
6. Critical Issues	20
6.1. File input handling	20
6.2. Performance	20
6.3. Exception handling	21
6.4. Demonstration	21
6.5. Multi-Client	22

6.6. Server Overload	22
7. Relationship and Difference with OCD_1	22
8. Conclusion	23
9. Prototyping	24
10. Reference	27

# 1.Executive Summary

This document illustrates the operational concept of the Remote Package Dependency Analysis System.

## **Introduction:**

Big system has become increasingly popular in today's software industry. In order to successfully implement big systems we need to partition code into relatively small parts and thoroughly test each of the parts before inserting them into the software baseline. As new parts are added to the baseline and as we make changes to fix latent errors or performance problems we will re-run test sequences for those parts and, perhaps, for the entire baseline. Starting this process requires an effective tools for lexical scanner tool.

For partitioning code into small parts, we are supposed to implement dependency analysis and build a graph to represent relationship between packages, then find the strong components of them by Tarjan Algorithm.

Finally, the system will run at a server, accepting message from client, handle them and then pass them back, dispatcher result on the client screen.

## **Uses:**

The Remote Package Dependency Analysis System package is design for six type of users, every user can choose the use mode in the command line.

### **Partitions:**

There are Seven main packages: Tokenizer, SemiExpression, TypeTable, TypeAnalysis, DepAnalysis, StrongComponents, Client, Server.

Tokenizer is used to extract words from a stream of characters.

SemiExpression is used to group tokens into sets.

TypeTable package provides a container for files' type, served as a bridge for further dependency analysis.

TypeAnalysis package provides a class typeAnalysis, with a TypeTable and function add, which is used to add Parser's result to TypeTable.

DepAnalysis package defined a class CsGraph, and a class DepAnalys, with the function BuildGraph, ConnectNode, build a graph represent the dependency information of the files.

StrongComponents package consists of functions FindConnect, Tarjan, to find all the strong components of the DepAnalysis.

Client package build the environment of client side of the system.

Server package build the server side of the system.

The project also includes many other packages to support main packages: FileSystem, Logger, Parser, MessagePassingService, TestUtilities. We will further explain these packages in the partitions part.

### **Application Activities:**

The application activities are divided into eight parts:

- Accept input files or directories, build the message and pass it to the server.
- Handle them by lexical scanner.
- Build TypeTable based on result from lexical scanner.
- Implement TypeAnalysis.
- Do dependency analysis, build the CsGraph.
- Get the Strong Components of the CsGraph.
- Pass the reply message back to Client.
- Display the result on the Client.

### **Critical Issues:**

In implementing Remote Package Dependency Analysis System, several critical issues shall be carefully considered. Their possible solutions have been given, and some of the issues are as below:

- File input handling: in which the Scanner get inputs and judge their readability.

- Performance: how to deal with extreme situation, such as large inputs, unreadable inputs.
- Exception handling: how to handle the possible exceptions
- Thread-Safe: How to handle situations where multi-requests are made, how to keep message safety between threads.

## **2. Introduction**

With the remarkable advance in both computer hardware technology and software engineering technology, developers are able to create huge projects with complex structures to meet various demands.

When we develop large size projects, we often raise the issue that, does my code follow the coding principles, are my classes and methods designed and structured appropriately, how to improve my code's readability and beauty. A code analyzer help you to do this.

### **2.1. Application Obligations**

The primary requirement of Remote Package Dependency Analysis System includes accept source code file, extract tokens, implement dependency analysis, find strong components of dependency graph, display the result.



## 2.2. Organizing Principles

The organizing principles for the system are based on two classes Client and Server, Client is used to accept files and display result. Server is built for dependency analysis functions.

## 2.3. Key Structural Ideas

To implement Remote Package Dependency Analysis System successfully, some key structures are created.

- Build a class Toker to read words from source file.
- Create a struct context as shared data storage.
- Create two array of string to store special one and two characters.
- Create private ConsumeState worker class as a friend of Toker
- Create a TypeTable to store type information of files, supporting further dependency analysis.
- Build a TypeAnalysis, with a TypeTable and function add().
- Create a CsGraph, store the dependency information of the packages.
- Implement a BlockingQueue, to arrange the message between Client and Server.

## 3. Uses

### 3.1. Myself

I am the first user of this system. When I finish it, I will run the whole project and get the auto-test result. Analyze every line to make sure it meets the requirements.

Then I will input other source file to test its Robustness and scalability.

### 3.2. Instructor and TAs

After finishing the project. It will be submitted to instructor and teaching assistants for grading. They will read the documents first, run the project and review the auto-test result, determine whether this project meets all the requirements.

### 3.3. Developers

Developers are users who develop large system, they will use this system to analyze their source code. They usually need log file to store analysis result. The user interface for them are more detailed.

### 3.4. Project Managers

Project managers will use Remote Package Dependency Analysis System to capture general information of their projects.

They probably input big source file to the tool, so the tool should consider performance and storage of big data as an issue.

### 3.5. Customers

Customers will use Remote Package Dependency Analysis System for their own code. To figure out the structure and information of their code.

## 4. Partitions

### 4.1. Tokenizer

The Tokenizer package is one of the core package of this Lexical Scanner. It declares and defines a public Toker class that implements the State Pattern with a abstract ConsumeState class and derived classes for collecting tokens.

Toker identifies words(alphanumeric tokens) from a stream, throws away whitespace and optionally throws away comments. Special one and two character tokens are stored in two vector<string>, it may be changed by calling related functions.

Toker returns words from the stream in the order encountered. Quoted strings and certain punctuators and newlines are returned as single tokens.

The package consists of three main classes: Token, Context, ConsumeState.

And ten classes represent state. They all inherit from ConsumeState in public mode.

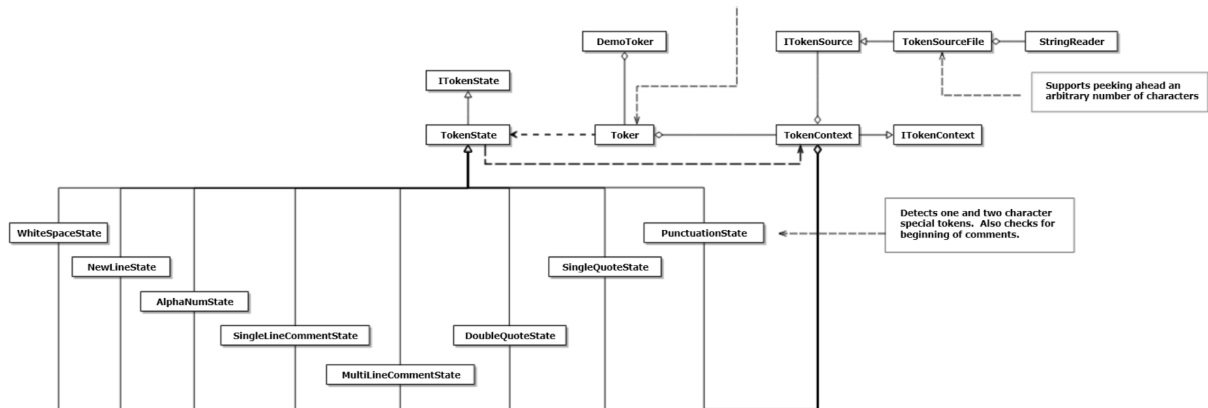


Figure 1:Token Class Diagram

## 4.2. SemiExpression

This package provides a public `SemiExp` class that collects and makes available sequences of tokens. `SemiExp` uses the services of a `Token` class to acquire tokens. Each call to `SemiExp::get()` returns a sequence of tokens that ends in `{`, `}`, `;`, and `'\n'` if the line begins with `#`. There are three additional termination conditions:

A sequence of tokens that ends with `:` and the immediately preceding token is `public`, `protected`, or `private`.

Each SemiExpression returns just the right tokens to analyze one grammatical construct, e.g. class definition, function definition, declaration, etc.

The SemiExp class also implement a class ITokenCollection, ITokCollection is an interface that supports substitution of different types of scanners for parsing. For example, XmlParts.

### 4.3. Parser

Parser package instances collect semi-expressions from a file for analysis. Analysis consists of applying a set of rules to the semi-expression, and for each rule that matches, invokes a set of one or more actions.

### 4.4. TypeTable

TypeTable package implements a Dictionary with string as key, List<TypeItem> as value, where TypeItem is a struct with two strings representing file and namespace. This Dictionary servers as a container for files type, also as a bridge for further analysis.

## 4.5. TypeAnalysis

TypeAnalysis package provides a class typeAnalysis, with a TypeTable and a function add() in it. This package is used to hold result from parser.

## 4.6. DepAnalysis

This package contains two parts.

First is a user-defined data structure, CsGraph. This class implements CsNode and CsEdge for the Graph, add, remove functions to modify CsGraph, walk functions to traverse the whole graph by DFS(Depth-First Search).

Second is DepAnalysis. This package includes a CsGraph as core element, and functions BuildGraph, ConnectNode to build the CsGraph based on using and reference relationship of input files set.

## 4.7. StrongComponents

This package contains a function: Tarjan. which is used to find all strong components of the CsGraph from DepAnalysis by Tarjan Algorithm. Store the result in a 2D List.

## 4.8. MessagePassingService

This package build the communication channel between Client and Server based on a BlockingQueue. The BlockingQueue consists of a queue and a lock. the lock servers as a monitor, all dequeue operation occurs inside the lock.

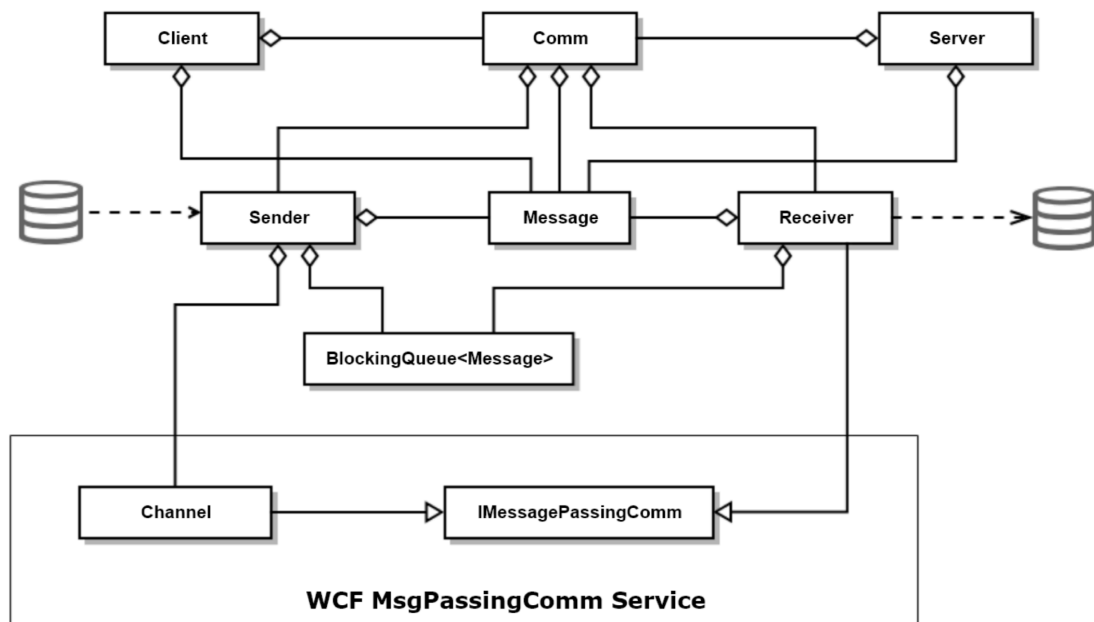


Figure 2: CsMessagePassingComm Class

## 4.9. Client

This package is a WPF App, with .xaml and .cs files to build graphical user interface and logical functions. the GUI will show files and directories on local address on initial, files and directories from remote address after connecting. Every Listbox implements mouse double click event to show content of file, it

also implements checked box to support multi-select. The “Analyze” button will send message with selected files to server.

#### 4.10. Server

This package build server side for system. all analysis process are run on this side. after receiving message with command and arguments from client, the server will call corresponding MessageDispatcher to handle the arguments, return a reply message with well-formed result as arguments back to client.

#### 4.11. TestUtilities

This package provides classes StringHelper and Converter and a global function Putline(). This package will be extended continuously according to format design requirements.



## 5. Application Activities

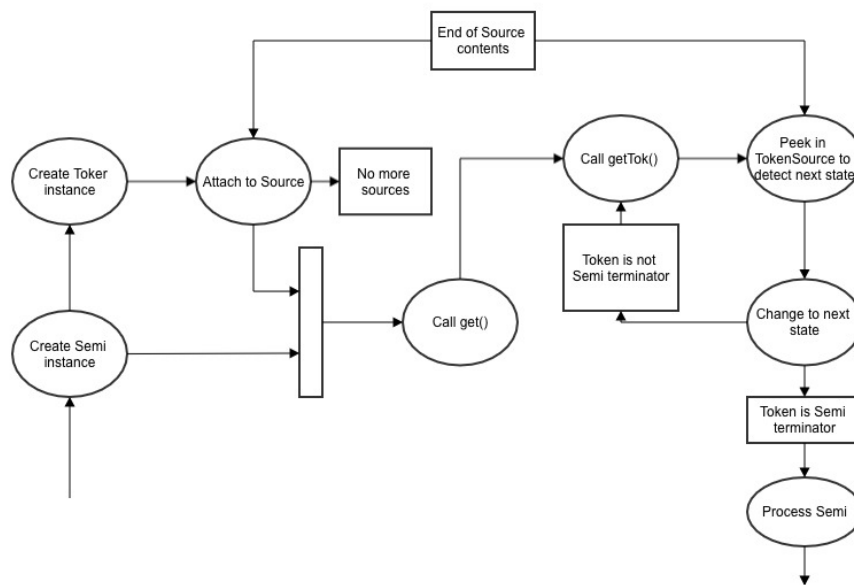


Figure 3 Lexical Scanner Diagram

### 5.1. Input source files or directories

First activity of the system should be getting the input from the users. Client will handle this input files and directories. For the directories, Client will extract all files under them recursively. put them into arguments of message, then be ready to send it to Server.

### 5.2. Server accept message from Client

After receiving message with command and arguments from Client, Server will call corresponding functions to handle the request from Client.

### 5.3. Extract tokens

Tokenizer has State Pattern class Token, Token identifies words(alphanumeric tokens) from a stream, throws away whitespace and optionally throws away comments. Quoted strings and certain punctuators and newlines are returned as single tokens.

### 5.4. Group tokens into sets

This package provides a public SemiExp class that collects and makes available sequences of tokens. SemiExp uses the services of a Token class to acquire tokens. Termination conditions are designed.

### 5.5. Build TypeTable

Implements a Dictionary with string as key, List<TypeItem> as value, where TypeItem is a struct with two strings representing file and namespace. This Dictionary servers as a container for files type, also as a bridge for further analysis.

## 5.6. Do Dependency Analysis.

Referring to TypeTable, connect two package if they share the same namespace, or share the same class within one namespace. Store the result in a graph.

## 5.7. Find all Strong Component

Implement Tarjan Algorithm, find all strong components of the graph representing dependency information. store the result in a 2D List.

## 5.8. Handle the exception

Firstly, the input file may not be the C++ or C#. The Lexical Scanner should do judgement at the beginning.

The content of the file may contain incomplete or wrong syntaxes, the Lexical Scanner should exit properly and store the former result, then give users notification.

The Server should also check the validness of command of message from Client, validness of files' address.

Exception Handle strategy usually uses try, catch keywords. Exceptions can be generated by the common language runtime(CLR),by the .NET Framework on any third-party

libraries, or by application code. Exceptions are created by using the throw keyword. The result will be displayed on console.

### 5.9. Pass the reply message with result back to Client

Results are stored in the arguments of reply message. after getting reply message from Server, Client will extract the result and dispatcher them, display on the result screen.

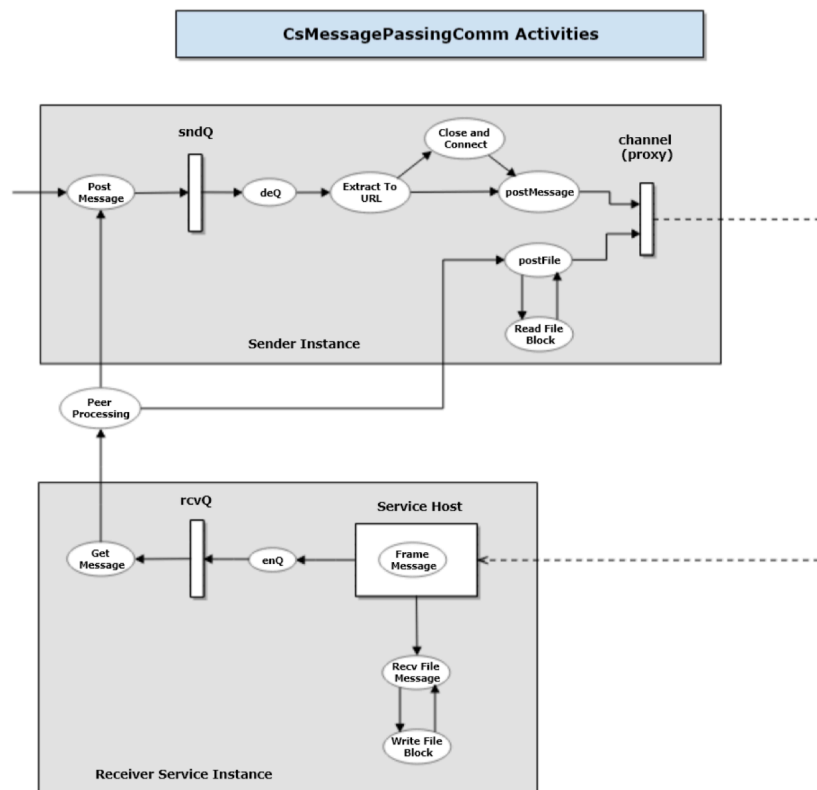


Figure 4: CsMessagePassingComm Activities

## 6. Critical Issues

### 6.1. File input handling

What kind of files can be accepted? How to deal with complex structure?

This issue concerns the entry of the tool, it is of vital importance.

Solution:

- The tool only accept code files with no error detected by Visual Studio.
- we build a GrammarHelper and ScopeStack to help deal with complex source code.

### 6.2. Performance

Every system of large scale should take performance into consideration. How many files the system can accept in a fixed time? How long it will take to handle an exception and how dose the system resume process after handling the exception?

Solution:

- implement multithread to process scanning process.

- When encounter with an exception, lock the thread, store the former result, go to handle the exception. After that resume the thread from where we stop.

### 6.3. Exception handling

What should the project do when encounter an exception?

Solution:

The content of the file may contain incomplete or wrong syntaxes, the Lexical Scanner should exit properly and store the former result, then give users notification.

### 6.4. Demonstration

How will the project demonstrate its performance and result? Is it clear to instructor and other users?

Solution:

Add an auto-test unit to the project, which will run automatically and test all requirements. The result will be displayed on a single console.

## 6.5. Multi-Client

What if more than one users want to use this system at the same time?

Solution:

Add more address to Client Environment, every client use one address, they all send message to Server.

## 6.6. Server Overload

How to handle the situation that too many requests at the same time?

Solution:

Implement synchronization, assign every request with a thread ID, use lock to arrange these threads.

# 7. Relationship and Difference with OCD\_1

This OCD demonstrates the whole structure and functions of the Remote Package Dependency Analysis System, the functions are run at Server side, which include handling lexical input. so the tool demonstrated by OCD\_1 is part of the functions of this system.

This system consists of two side: Client and Server, Client accept the input from the users and display the result from the Server. The OCD\_1 only describes the tool Lexical Scanner.

This OCD describes the package information, which are all implemented. OCD\_1 is written before the actual implementation, so many contents are based on prediction or comprehension. some contents are inaccurate or wrong.

## **8. Conclusion**

In conclusion, Remote Package Dependency Analysis System can complete following basic functions:

- Input and read source code file or directories from Client
- Send message with file address to Server.
- Extract tokens
- Group tokens into sets
- Analyze dependency of the input files or directories.
- Find all strong components of the dependency result.
- Handle the exception
- Pass the reply message with result back to Client.
- Display the result on the Client.



The OCD illustrates the system aimed users and what they can get from the Remote Package Dependency Analysis System. Different modes meet different requirements of users. Remote Package Dependency Analysis System's main functions are achieved between two main packages: Client and Server.

The OCD also present detailed activities during the Scan processing. OCD demonstrates critical issues in the process.

## 9. Prototyping

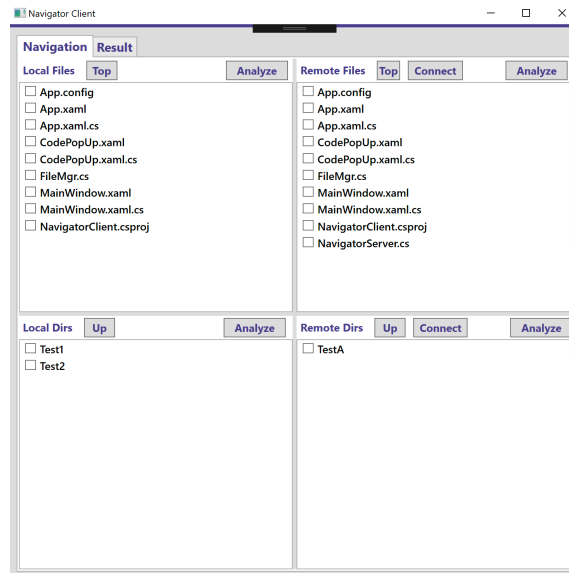


Figure 5: Client GUI

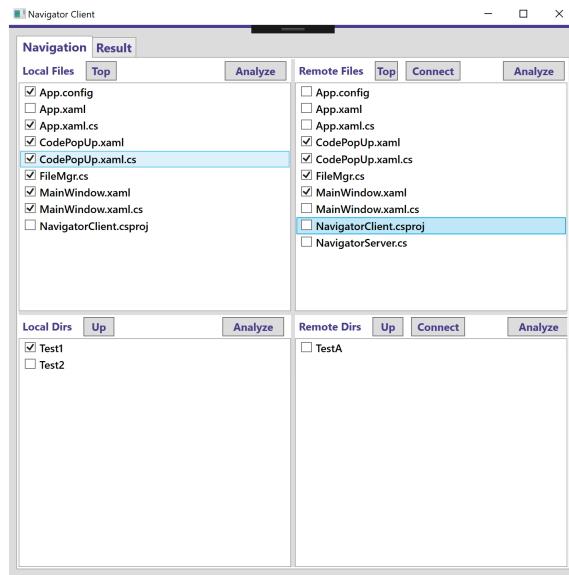


Figure 6: Select Files or Directories

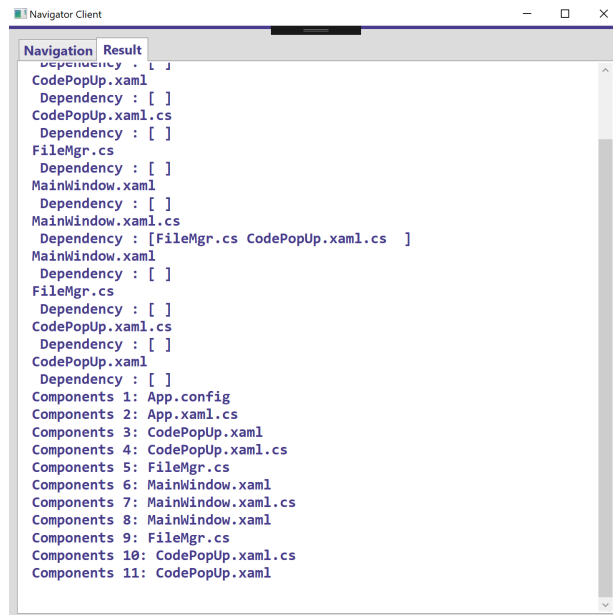
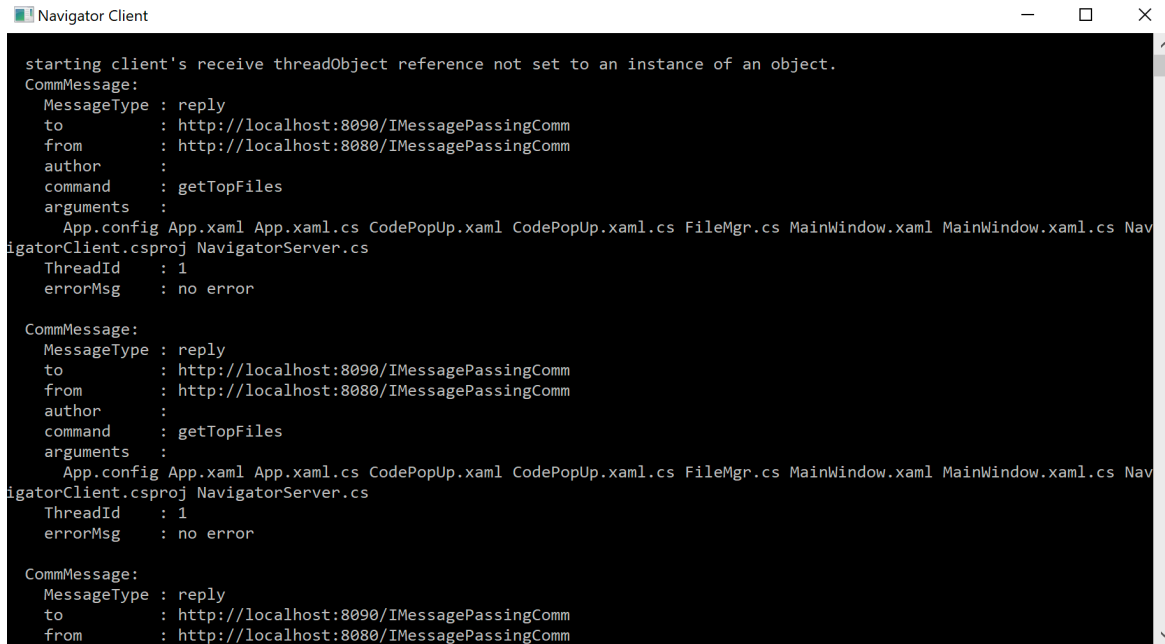


Figure 7: Display Analysis Result



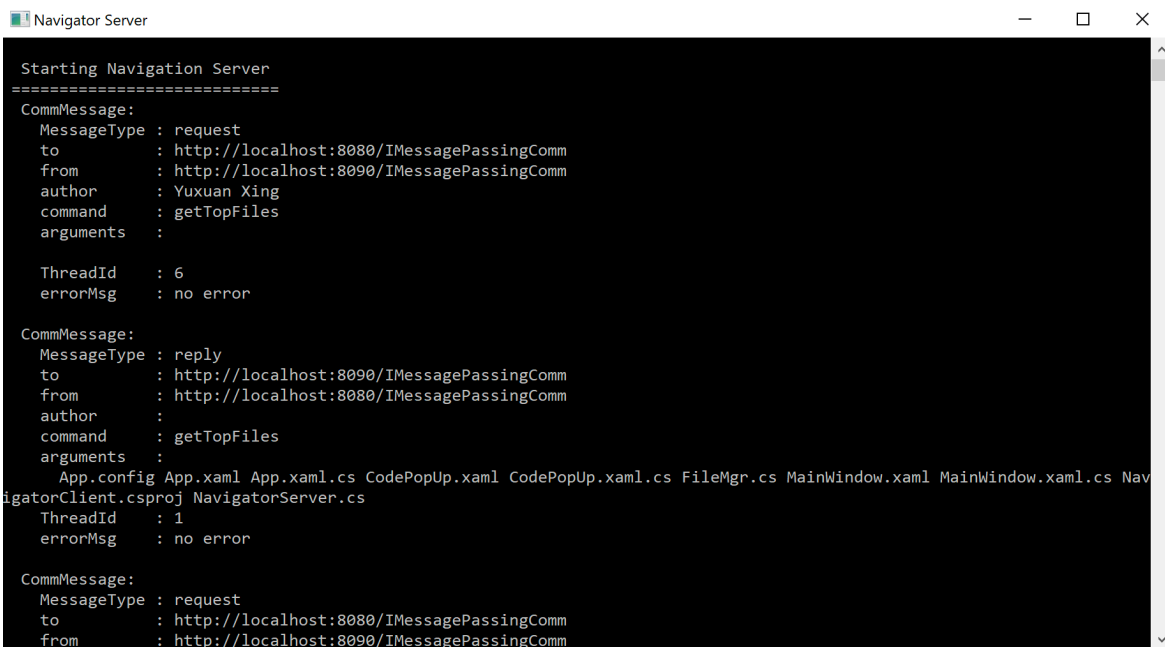
```
Navigator Client

starting client's receive threadObject reference not set to an instance of an object.
CommMessage:
  MessageType : reply
  to           : http://localhost:8090/IMessagePassingComm
  from         : http://localhost:8080/IMessagePassingComm
  author       :
  command      : getTopFiles
  arguments    :
    App.config App.xaml App.xaml.cs CodePopUp.xaml CodePopUp.xaml.cs FileMgr.cs MainWindow.xaml MainWindow.xaml.cs Nav
igatorClient.csproj NavigatorServer.cs
  ThreadId     : 1
  errorMsg     : no error

CommMessage:
  MessageType : reply
  to           : http://localhost:8090/IMessagePassingComm
  from         : http://localhost:8080/IMessagePassingComm
  author       :
  command      : getTopFiles
  arguments    :
    App.config App.xaml App.xaml.cs CodePopUp.xaml CodePopUp.xaml.cs FileMgr.cs MainWindow.xaml MainWindow.xaml.cs Nav
igatorClient.csproj NavigatorServer.cs
  ThreadId     : 1
  errorMsg     : no error

CommMessage:
  MessageType : reply
  to           : http://localhost:8090/IMessagePassingComm
  from         : http://localhost:8080/IMessagePassingComm
```

Figure 8: Client Console



```
Navigator Server

Starting Navigation Server
=====
CommMessage:
  MessageType : request
  to           : http://localhost:8080/IMessagePassingComm
  from         : http://localhost:8090/IMessagePassingComm
  author       : Yuxuan Xing
  command      : getTopFiles
  arguments    :

  ThreadId     : 6
  errorMsg     : no error

CommMessage:
  MessageType : reply
  to           : http://localhost:8090/IMessagePassingComm
  from         : http://localhost:8080/IMessagePassingComm
  author       :
  command      : getTopFiles
  arguments    :
    App.config App.xaml App.xaml.cs CodePopUp.xaml CodePopUp.xaml.cs FileMgr.cs MainWindow.xaml MainWindow.xaml.cs Nav
igatorClient.csproj NavigatorServer.cs
  ThreadId     : 1
  errorMsg     : no error

CommMessage:
  MessageType : request
  to           : http://localhost:8080/IMessagePassingComm
  from         : http://localhost:8090/IMessagePassingComm
```

Figure 9: Server Console

## 10. Reference

- <http://ecs.syr.edu/faculty/fawcett/handouts/CSE681/Lectures/Project1-F2018.htm>
- <http://ecs.syr.edu/faculty/fawcett/handouts/CSE681/CodeAnalyzer/>
- <http://ecs.syr.edu/faculty/fawcett/handouts/CSE681/BestProject1s/TianYou.pdf>