**CSE 681 Software Modeling and Analysis**

**Project 3**

**Dependency Analyzer**

**Sahithi Desu**

**Graduate student, Computer engineering**

**Syracuse University.**

**Instructor: Jim Fawcett**

**Version 1.0**

**22nd October 2014**

Dependency Analyzer

# Purpose

The Dependency analyzer is an automated tool which will analyze different types, its relationships along with the package dependencies for the file sets that are present on one or more directories in local or remote machines. Dependency Analyzer provides client side user interface, where the files to be analyzed will be uploaded or selected according to the user’s choice and the results of the different relationships determined are displayed. Additionally, all the analysis of the files are done by the code analyzer component, present in the server. The server resides on the remote machine. This document deals with the description of the architecture of the dependency analyzer

# Introduction

Dependency Analyzer is intended to resolve names to declarations, expressions to types from certain actions and rules and finally displaying the results, i.e. types and their relationships along with its package dependency. A file in which a C# program resides may hold the following “types” like struct, enum, class, delegates, abstract class, non-abstract class, interfaces including all the built in types. These types in turn will hold different “relationships” with one another. This tool is intended to analyze the different relationships between classes and finds all the packages that are dependent on the current package. The above functionality is done on the files that reside remotely on different servers upon client’s request.

The Client side application should be able to

* The Client application is represented in the form GUI
* View the multiple files present in the multiple servers and select the files that has to be analyzed
* Search for the files in the server
* Upload the files to the server if necessary
* Display the package dependencies and type dependencies based on user’s choice

The Server side application should be able

* Listen to all the client requests
* Able to process clients requests which are sent in the form of XML or JSON
* Store all the uploaded files by the client either in the local file system or database.
* Process multiple client requests concurrent
* Analyze multiple files based on the client selection and give out type analysis, relationship analysis and package analysis.
* Perform Error Handling
* Should store all the recently analyzed information in the local cache.

These Identified relationships will help the user to understand the structure of the program and enable him to make the decisions wisely which plays a vital role in writing a quality code considering size and complexity metrics in to consideration. In addition, this tool enables the user to deal with complex system easily. This tool also generates metrics like nesting depth, size of the code, Fan IN (tells how many modules are dependent on a single module), Fan Out (how many modules a single module is dependent on), Modularity index about the source code. These metrics generated will help to analyze code complexity and help the users to develop the code with less complexity.

# Use Case Analysis

The behavior of the software that occurs when the user requests or interacts with the software is described as a “Use case”. This methodology helps to identify how this software can be used by a broad range of users and how do they interact with it, which is helpful for analyzing requirements.

* 1. **Principle Users**

• **Software Architect**: It is the prime responsibility of the software architect to make design choices and to recognize potential reuse in the organization or in the application.

If there is any system that can be build using the system which has already been analyzed, architect can use this tool for framing the architecture of the system by looking in to the structure of the system which has been already analyzed by the tool by looking at its class structure and its relationships.

• **Quality Analyst:** The role of the QA is to determine the quality of the code and validate whether the code has met the requirements or not. He can use the tool to analyze the quality by calculating the values of the metrics produced and reduce the complexity of the code. The types and their relationships that are analyzed by this tool will help quality analyst to calculate the degree of coupling and cohesion. For a code to have high quality, low degree coupling is preferred as the degree in which each modules depend on each other should be less.

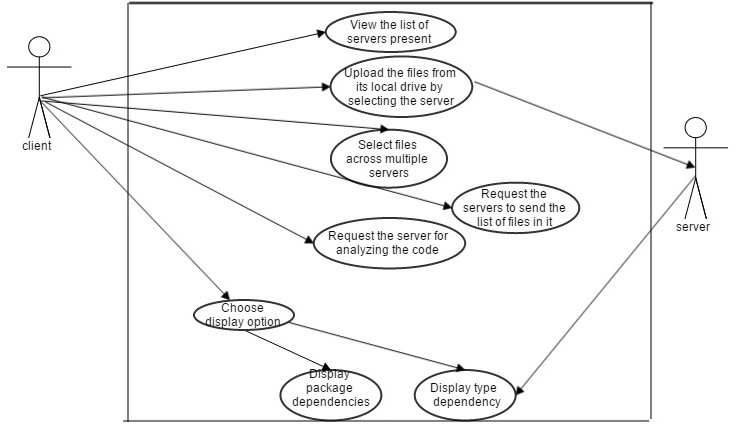
• **Developers:** Using this tool developers can identify the coding issues and bugs at the early stage of development cycle, passing better code into the later stage of development. He can estimate which area of code is prone to bugs by identifying the dependency relationships and looking in to the hierarchical structure of the classes. The more the depth of the hierarchical structure and dependencies, the more the degree of cohesion increases.

• **Customer:** Customer doesn’t care about the detailed working of the project. So, a comprehensive view would be sufficient for the customer to understand the working of the project. The relationships produced by the tool will help him to understand the necessary details.

To understand the complete working of the system, it can be viewed with respect to three major users

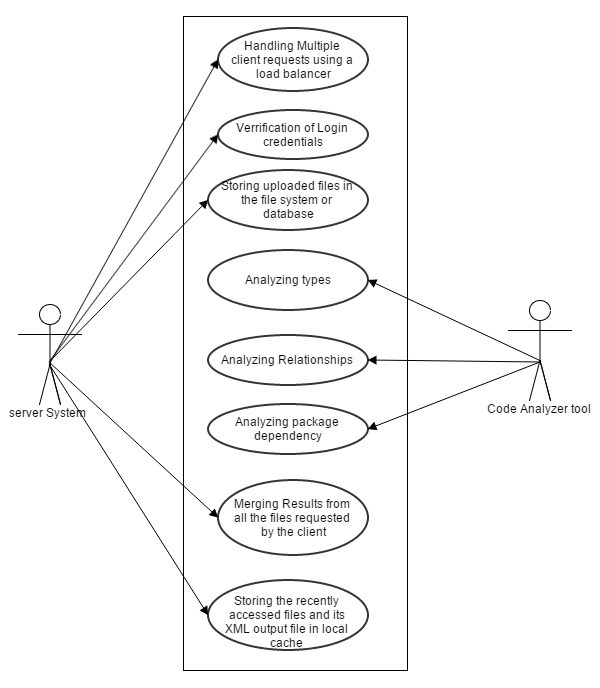
1. Client user
2. Server System

**Client User:**



* **Upload files**: The client can select a particular server from the list of servers and upload the file into the server.
* **View Server List**: Client can view the list of servers present remotely.
* **Request Files**: the Client selects the servers from the list of servers and requests the corresponding server to display its files in the respective servers from where the client can select files across multiple servers.
* **Select a Particular File**: The client has an option to select a particular file among the list of files which are listed. The metadata of the files along with the types, sizes and complexities of that particular files are shown.
* **Requesting server to analyze the files**: Upon selecting all the files list the client requests the server for running code analysis tool on these elected files.
* **Choose Display Category**: The user can select different options for the display. Depending on what option the user chooses, the system would display the corresponding information. The user would choose for either type’s relationships or the package dependencies.

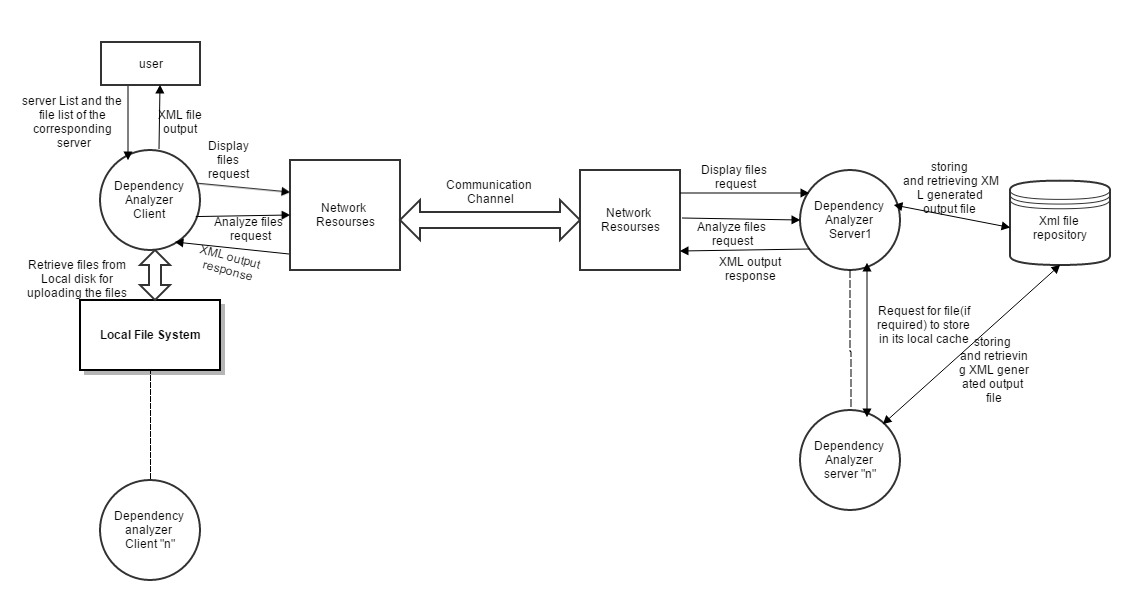
**Server System:**

****

* Handling Multiple Client Requests: Server handles multiple client requests using load balancer. This Load balancer will allocate the task for the server based on the server’s load in a round robin fashion.
* Verification of login credentials: The server checks with its database to check whether the user is an authorized user.
* Analyzing types, relations and package dependencies: Each server runs the code analyzer tool to identify different types, relations and package dependencies. The detail explanation of this will be discussed in the later part of OCD.
* Merging of results: All the partial results produced from the different file sets are merged in to a single XML file and it is given to the client.
* Storing in the local Cache and database: The generated recently generated XML output file is stored in the local cache and the same output xml file is stored in the “output” database.
  1. **Context Diagram:**

The context diagram describes about how the system being interacted with the surrounding environment. The dependency analyzer client interacts with the user with GUI. The user selects the server list and the files that are present in the corresponding server that are displayed in the GUI. Before that, client makes a request to all the servers to get the list of files present in the servers. The server sends all the list of files to the client as response. When client requests for analyzing files one of the servers accepts the request and give back the XML generated output file to the client besides storing it in the XML repository.

The XML file is generated inside the server which has accepted the request by running its code analyzer tool inside it. But before that, it makes requests to the servers which are present in the servers list given by the client to get the files that are present in that server and stores in to its local cache, thereby, running code analysis tool on them.



# Views

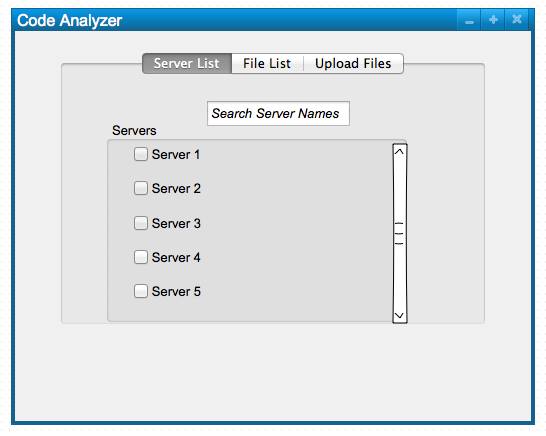
The Dependency analyzer implements graphical user interface in C# using the facilities .NET frame work 4.5. The use of GUI will be demonstrated and will be implemented in WPF (Window presentation Foundation).

4.**1 GUI for Login Screen:**



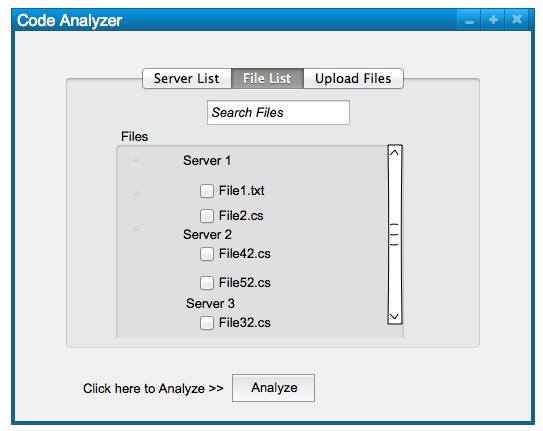
The GUI has a text box for the user name and the password, where the client can give his login credentials in to it. This is done to verify whether the user is the registered user or not.

**4.2 GUI for Selecting sever list**



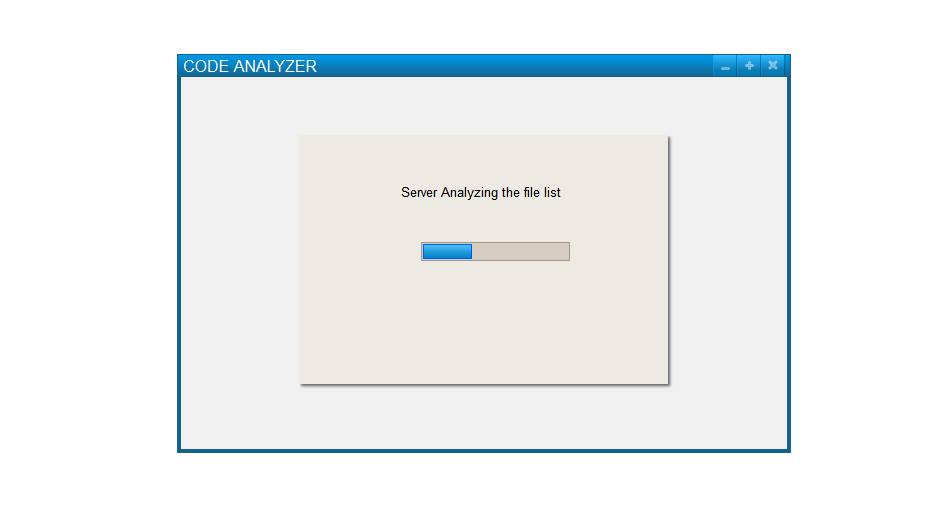
This has a list of check boxes which displays all the different servers which are remotely connected. The user is enabled to see the list of servers and select them by clicking the text boxes. The user has also option to search for the server name.

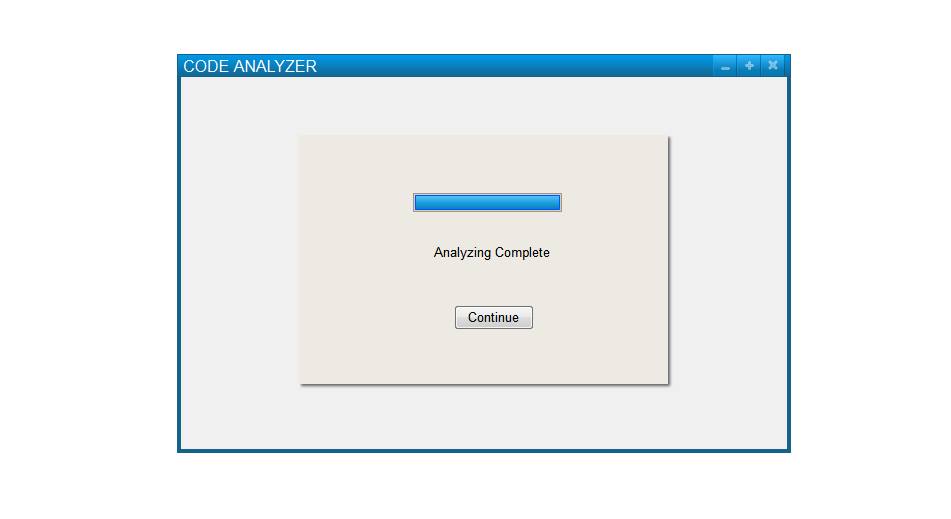
**4.3 GUI for Selecting File list**



The above GUI displays the file list under the each server 1. This enables the user to select the files under the each server. After the selection of files the user can click the “Analyze” button after which the server does the code analysis process.

**4.4 GUI for showing the Analyzing process:**

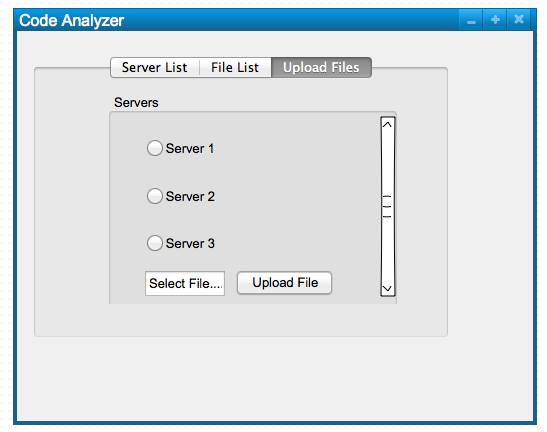




This shows the progress of the code analysis. This notifies the user, the progress of the code analyzer process in the server. Once the code Analysis process is done, it has a continue button. Upon clicking that, the user can continue to select the options for the display.

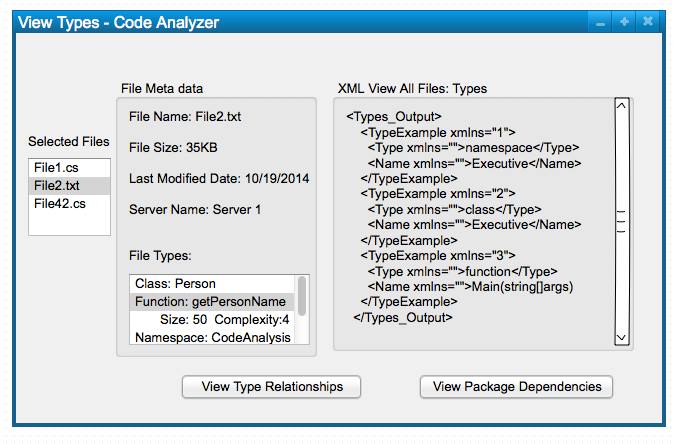
**4.5** **GUI for upload files:**

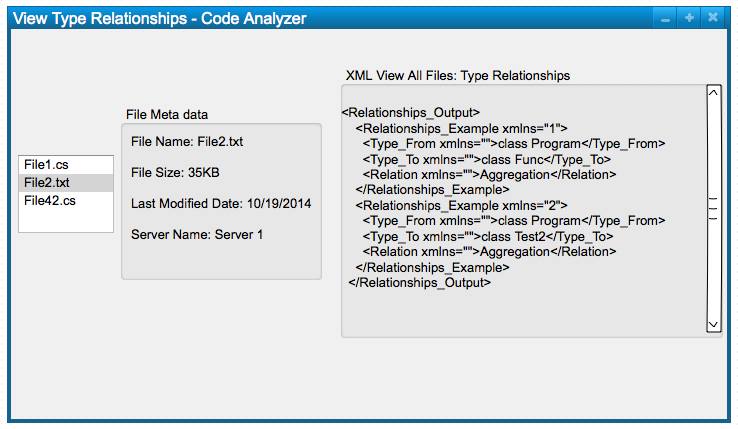
The client is also given an option to upload the files. If the Client wishes to upload a file to the server, he can select the upload files tab. Here the server list is shown in radio buttons. The user select any one of the server in that server list and upload the file in to the server’s local repository. The user can upload the file by browsing his files in his local disk.

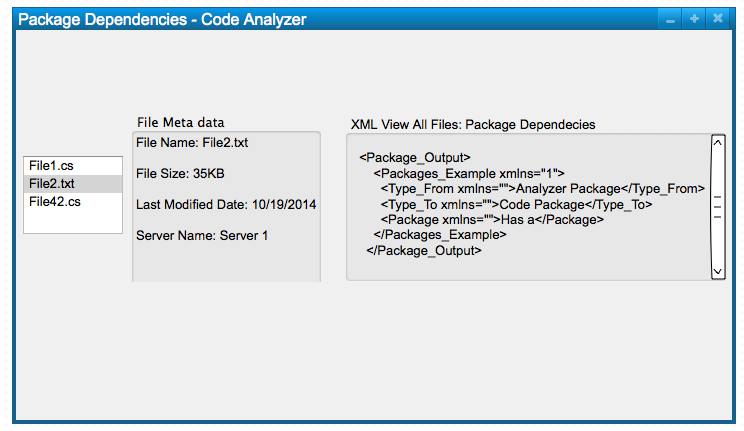


**4.6 GUI for display**

Once the Code analysis is done the client can see all the types that have been identified for the selected list of files. The selected list of files are displayed at the right corner where the user can see the metadata of a file which has been selected out of the files list displayed. The XML view of all the types of all the file list is displayed on the left most corner. The first GUI shows the display of all the types, the second and the third displays the relationships and package dependencies respectively. Once the type relations have been displayed the user is given an option to see either “View type relations” or “View Package Dependencies”, or both.







# Partitions

The Dependency analyzer can be divided in to different partitions.

1. Client Partition
2. Server Partition
3. Code Analyzer Partition

All these partitions are explained in detail in the diagrams listed below.

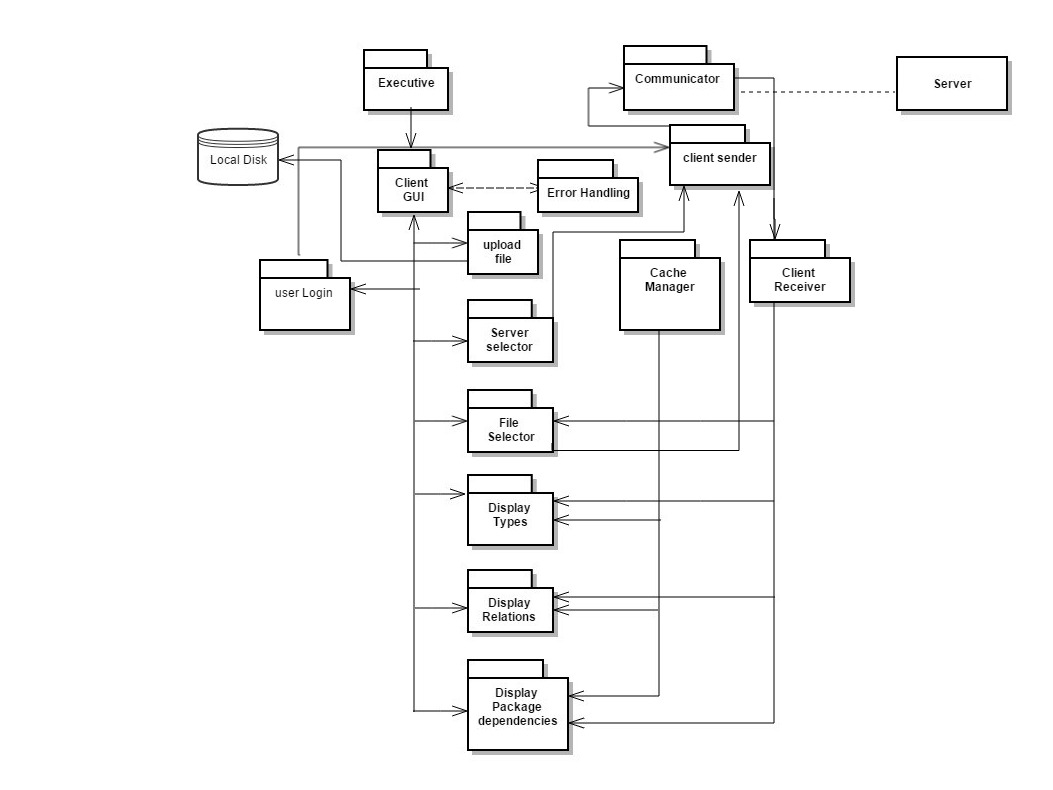
* 1. **Partition for Client dependency Analyzer**

Here Partition of the Single client among the “n” number of clients is shown in the figure.

The tasks done by the client can be divided in to following tasks:

1. Presentation of Graphical user interface.
2. Provides ability to show the different list of servers present remotely and list of files that are present with in the respective servers.
3. Sends the user’s request to the server.
4. Receive the response from the server.
5. Maintaining a local cache of recently XML files generated.
6. Providing uploading facility to the user, where the user can upload the files from his local disk.
7. Displaying all the responses given by the server to the user.

The partitions which are shown below are based on the tasks performed by the client. All the partitions in detail are discussed here.



*Partition diagram for client*

* **Executive:** Executive module is the main module which contains the main method for the execution of client. It loads the graphical user interface of the client through which user can interact.
* **GUI Module**: This contains graphical user interface through which the user can interact with the client. It takes input from the user and shows the output according to the request requested by the user. It has many tabbed planes like server list, file list, login, and display.

- **Login** tab enables the user to enter his login credentials and the form is verifies whether the user is valid or not.

- The **server List** tab displays all the list of servers present remotely to the user, enabling him to select those using checkboxes.

- The **File list** tab displays all the files that are present with in a particular server

- The **Display types** plane displays all the types to the user in an XML format

- **Display Relations** plane displays all the relations between selected files in the in the XML format.

-**Display Package Dependency** plane displays all the package dependencies between the selected files.

* **User Login Module:** When the user enters his login credentials in the GUI login page, it sends the user’s credential information to the client sender, from where those details are sent to server for validation. Once the validation is done the server is done with the validation it gives response back to the client telling the request is valid or not.
* **Client Sender Module**: This module performs its task when client wants to send its message to the server. This sends message to the server via communicator. This maintains a queue which has all the requests to the server and files which the client wants to upload.
* **Client Receiver Module**: This module performs its task when server sends a response to the client. It has a receiving queue which has all the files or the commands given by the server. It will wait till response it completely downloaded. After downloading the response it will parse the command and perform actions according to the servers’ response. For example if the response is file list, the receiver will send the file list to the **File Selector** Module which displays the files list of the particular server in the GUI. After the user selects the file list the **file Selector** gives the list to the client sender Queue which inturn goes to server via communicator.
* **Cache Manager**: This module handles the cache mechanism for the Dependency analyzer client. Whenever the client requests to analyze particular set of files, it will check with in its local cache whether the XML file for that file list has been generated, if it is generated, it just gives the XML file from its cache to the user rather than contacting the server for analyzing. This is done to reduce the network traffic and load on the server.
* **Server Selector**: When the user selects the server list form the GUI, it checks the server list and sends a command to each server in the server’s list. This command is for asking respective server for its file list. These commands are sent to the client sender module from where it is sent to the respective server through the communication module.
* **File Selector**: As explained in the above in the example stated in Client Receiver module, File selector takes the list of files that server has given and displays in the GUI. It also gives the list of user selected files to the server to analyze.
* **Display Component**: This has display types module, display relationship module and display package dependency module. When it receives an XML file form the Client receiver, it displays the content of that XML file in the display plane of the GUI.

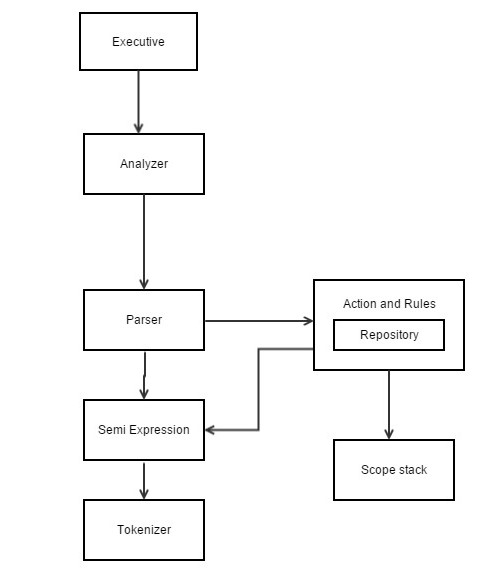
* **Communicator Module:** This module’s main responsibility is to handle the communication between the client and the server.

**-** This creates a channel between client and the server through which client will interact with the server.

**-** When the request is sent from the client from , it takes the message from the client’s sender queue and gives the message to the server’s Receiver queue, from where server process the message and takes action accordingly.

**-** When the response is given by the server, it gives the message to the client’s receiver queue from where client takes the message.

* 1. **Partition for of the Code Analyzer tool:**

****

.

* **Tokenizer:** It reads symbols, punctuations and characters from the file stream attach to it and convert them in to meaningful tokens removing white spaces and comments.
* **Semi Expression Analyzer:** It builds semi expressions from the bunch of tokens which has to go through parser. These sequences end with ; or { or }.
* **Parser:** It analyzes the source code and performs type and function analysis. It detects code constructs defined by the interface contract rules. Each of these rules have a grammar construct detector and also a collection of interface contract of actions.
* **Rules and Actions:** This has all the code specific to the application mostly required for analysis. It defines all the rules which incorporates grammar constructs. These rules are used by the parser to analyze the expressions given by semi expressions.
* **Relationship Analyzer:** It detects the relationships between classes by using four detector functions which detects inheritance, composition, aggregation and using.

This tool is intended to analyze the different relationships between classes.

-Inheritance

-Composition

-Aggregation

-Using

* **Package Dependency Analyzer:** Package dependency analyzer analyzes all the package dependencies between all the complete set of files that are present. When we say dependency, to be specific if we say package A depends on package B, then it implies one of the following:

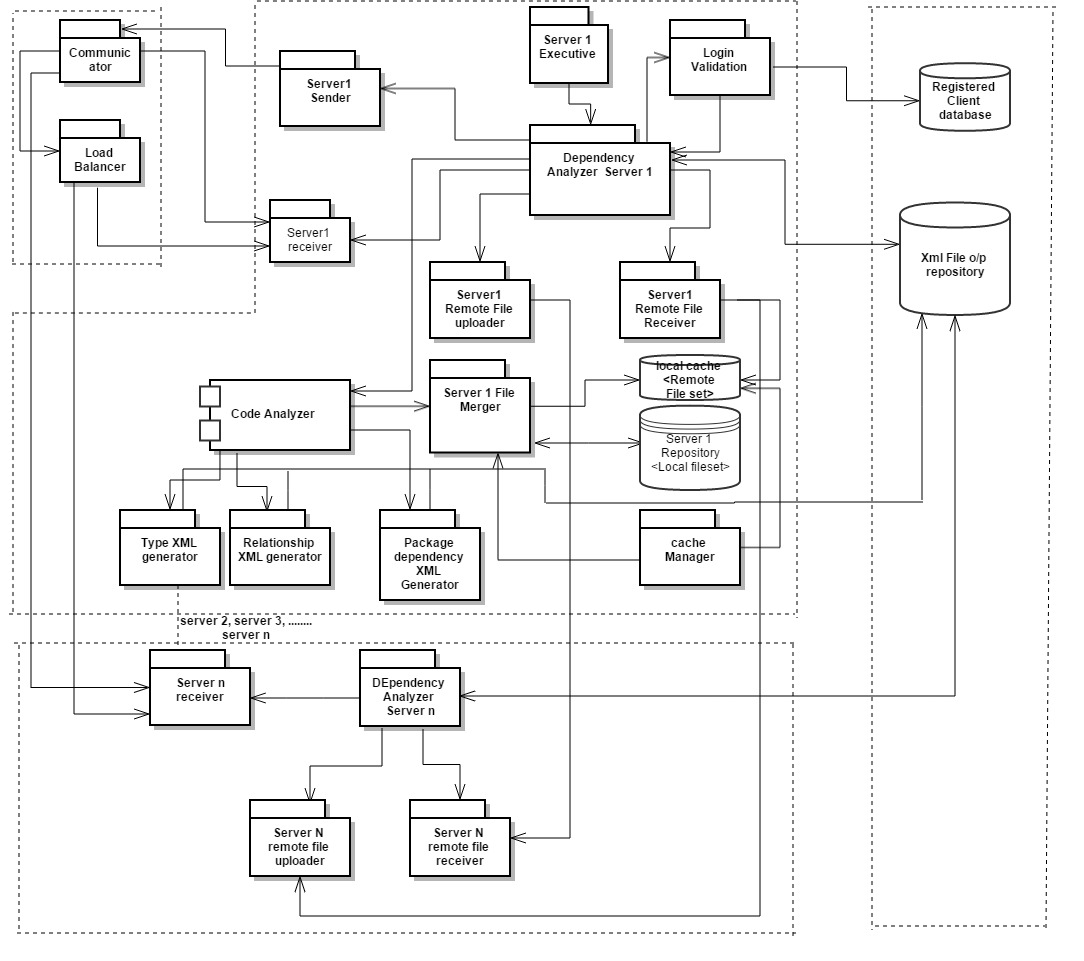
– Package A is using an instance of a type defined in package B.

– Package A has a Type that owns or contains a part of class B.

– Package A has a Type inheriting from a Type declared in B.

– Package B has a global function defined that is being used in package A.

* 1. **Partition for Server Dependency Analyzer:**



* **Server Executive**: It starts the execution of the server. It has the main method of the server.
* **Server Communicator**: It sets up the communication between client and the server. When the communicator receives the message from the client, it will parse the command to check what kind of request it is. If the request is to analyze the file list or checking the login credentials, then the communicator calls **Load balancer** which checks for the server which is of less load and allocates the task of code Analyses or login validation for the given set of files in a round robin fashion to the server. When the server selection is done the load balancer gives the list of file names to analyze to the selected server’s receiver. If the command is other than analyze then the communicator sees the sender server name in the message and directly sends the request to that respective server’s receiver.
* **Dependency Analyzer Server:** Dependency server is the main point of execution of server module. All the requests received in the queue will go to this module and depending on the type of requests this module gives the calls the other modules to perform their tasks. For example when the command is to analyze, it goes through the files list and if there is any file present in the foreign server, it calls the “**Remote File receiver module**” which will get those files from the remote servers.

* **Server Receiver:** This module puts the requests sent by the clients in to its queue. After all the Reponses have been received, it will dequeue and send those commands to the Dependency Analyzer Server to parsing.
* **Server Sender:** This module stores all the Reponses given by the server in a queue. It dequeues and sends the response to the client via communicator.
* **Login Validation:** When client sends a login validation request, the load balancer allocates that task to any of the servers depending on the load of the server. this modules checks with the client database whether he is an registered user or not and sends back the response to the client.
* **Local cache:** All the files of the remote severs will be uploaded here. The **cache Manager** doesn’t allow the “remote server file uploader” to upload the files When the code Analyzer is running in that server. And this will delete the files from the local cache when the code Analysis is done. The upload is done only when code analysis tool is not running in the server. This is done to produce reliable results after code analysis is done. If uploading and code analysis are allowed to be done in parallel, then, the code analyzer would do the analysis of on the files for other request’s set of files. This would result in a wrong output.

* **Server Remote File uploader:**  This module has been given access to upload the files in its server’s repository in to the local cache of the server which has requested for the files. When it gets the request from the remote servers for uploading the files, this module will upload the files to the requested server’s local cache.
* **Server Remote File receiver:** This module task comes in to picture when the server gives the list of files and the corresponding servers to it. This module, makes requests to the corresponding server’s “**Server Remote File uploader**”  
   to upload its files in the requested servers’ local cache.
* **Server File Merger:** This module merges the selected files from the local server repository and the files from the local cache to perform code analysis on this set of files. Once the Code analysis on all files in its repository are done, they are deleted by cache manager.
* **Code Analyzer:**  This module performs code analysis on the files which are in the in the File Merger. First it will perform the type analysis and store the type tables in the merger repository. Then it performs second parse on the file list to produce relationship list and the package dependency list and these lists are stored in the merger file repository. Once the entire code analysis is done all the XML files for the corresponding tables are generated and the tables in the merger repository are deleted. The generated XML files are stored in the XML repository along with the files list and servers list.Depending on what the client is asking for display the Dependency Analyzer server will take the corresponding XML file from the repository and sends to the client.
* **XML generator Component**: This component has Type XML generator, relation XML generator and package dependency XML generator. It stores the generated XML files in the **XML file repository** with the file set and their corresponding servers.
* **XML file repository**: This is being accessed by all the remotely available servers. This consists of all the XML output files that are generated by all the remote servers and these files are accessed by the remote server which performed the code analysis to give them to the client when it has requested for the display.

# Activity Diagrams

The activity of the dependency code analyzer can be divided in to sub tasks

1. Performing Login operation
2. Selecting Server list and file list to analyze
3. Performing Code Analysis for the selected files
4. Displaying the XML generated output files

The activity diagrams for these diagrams are drawn one by one to get the clear understanding of activity flow in the system between client and server, when various tasks are performed.

* 1. **Activity for Performing Login operation:**

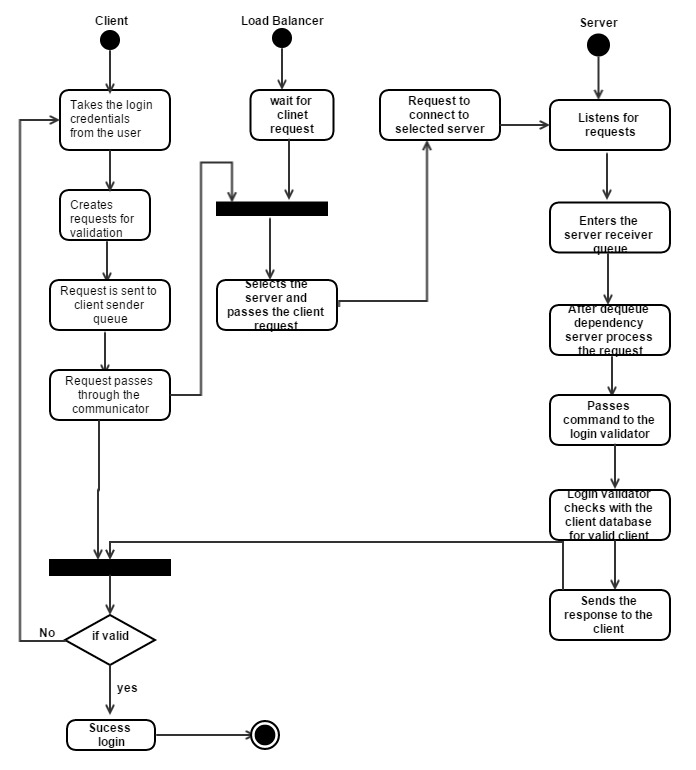
The following are the tasks done while this operation is performed:

- The Client takes the login credentials from the user.

- Client creates a request for validation with client id, username and password in its request.

- Client’s request is sent to the client’s senders queue from where it is dequeued and passed to the server via communicator.

**Activity Diagram for Performing Login operation:**

****

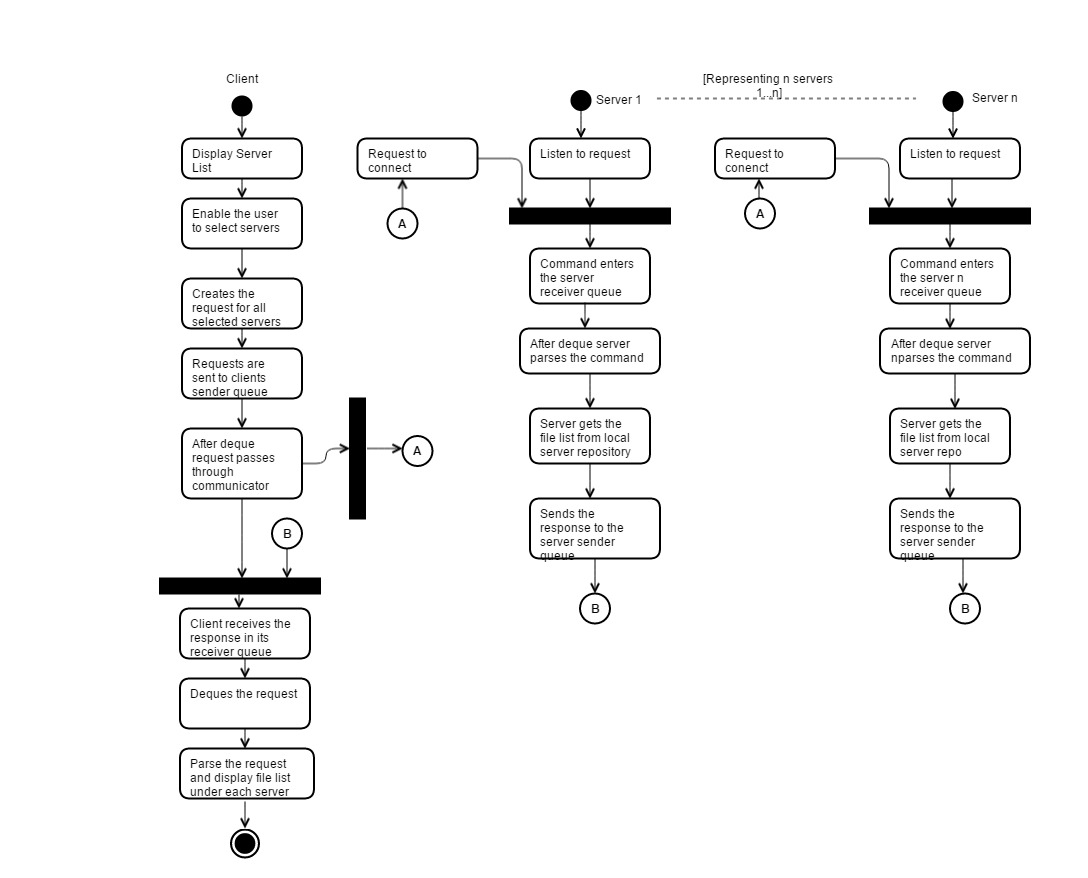
* The Server communicator checks for the request and if it a login request, it sends the request to the load balancer, which sends the requests to the server which has less load. Load balancer does this by allocating tasks to the servers in a round robin fashion, By doing this the load is almost equally distributed among all the servers and even if one server doesn’t work the Dependency analyzer will still give the results for the client
* After selecting the server the load balancer sends the request to the selected server’s receiver queue.
* Once the request is dequeued, it is parsed by Dependency analyzer server and the request is given to Login validator.
* Login Validator checks the details of the login credentials with the client database to check whether the request is from valid user or not.
* After the check is done the response is given back to the server sender queue from where it reaches client via communicator.
* The client decodes the response from the client’s receiver queue and tells the user whether he is a valid user or not.

* 1. **Activity For performing Selection of server and files list for the selected server**

Once the Login credentials are verified the Server and File selection activity is performed. The below listed are the list of tasks performed.

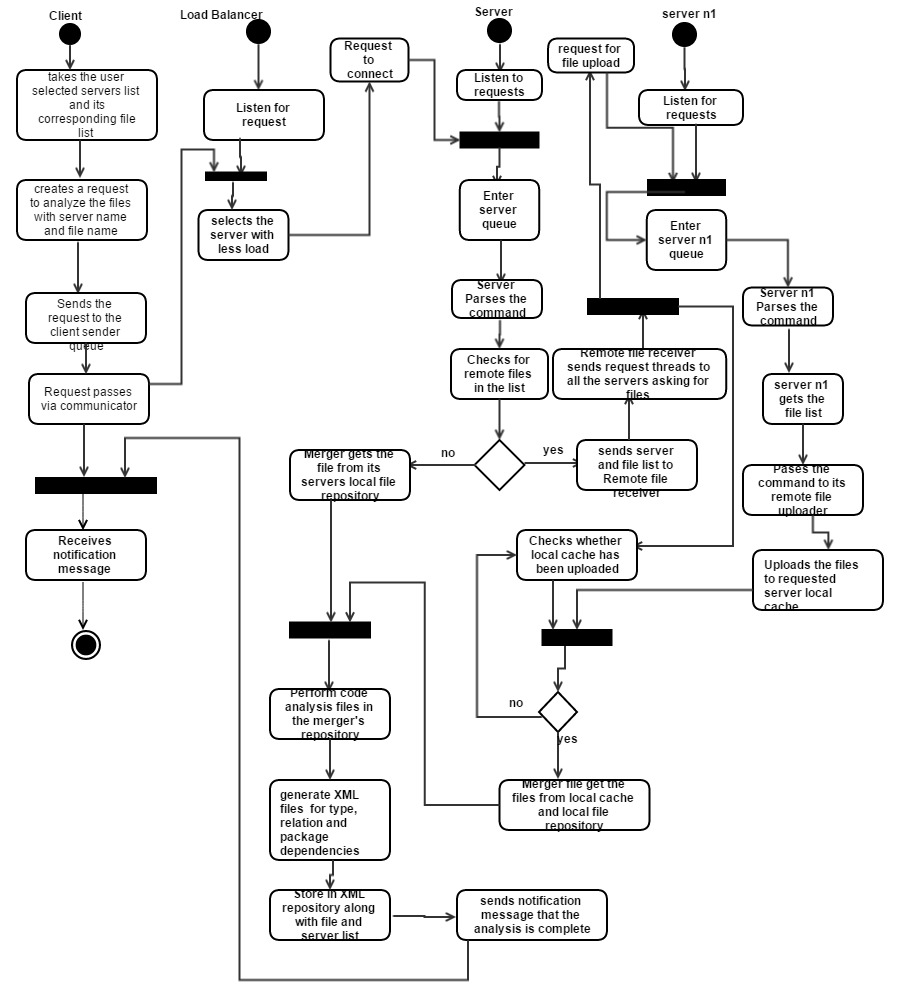
* The Client displays the server list in its GUI.
* It enables the user to select the server list from the list of servers available
* Creates the request for all the selected servers. With client id and the server ids for all the selected servers.
* The requests are sent to the listed servers parallely and it enters into the respective server’s receiver queue via communicator.
* When the requests are dequeued from the queue, the server parses the command and gets the file list from its local file repository.
* The list of these files are given as a response to the client via communicator.
* The client waits till it gets the response from all the listed servers. Then it decodes the responses from the queues and displays the file list under the respective server name thereby, enabling the client to select the file list.

**Activity Diagram for Performing Server and file selection:**

****

* 1. **Activity for performing Code Analysis on the listed Files.**
* The client enables the user to select the file list from the displayed file list in the GUI.
* Client creates a request for analyzing the file set with serverid, corresponding file names and along with the message “Do analyze”
* The client sends the request via communicator. The Communicator checks for the “Do analyze” message. If it is present, it sends the requests to the load balancer which selects the server to analyze the file list.
* The selected Server receives the request in to its server’s receiver queue. When the request is dequeued the server parses the request.
* If the server identifies any file name that belongs to the remote server, it sends the file name and the corresponding server id to the Remote file receiver. This looks in the command given to it by the server and sends the request to the remote server which is there in its command. The request that this gives to the server is upload request, with sender server id and the receiver server id.
* The remote server to which the request has been made receives the request in its server’s receiver queue. Now when the request is dequeued the remote server parses the command and gives the file list in the command to its remote File uploader. This will upload all the files present in its received command to the requested server’s local cache.
* When the file requestor sends request, it continuously checks whether all the listed files are uploaded in the local cache. It is done by calculating the count and comparing it with the count of remote files. If both the counts are equal then the File merger will creates a list with selected file names in its local repository and the file names in the local cache and upload these files in to the merger repository and perform code analysis with the code analysis tool present in the server.
* If the server doesn’t find any files that belongs to the remote server, it just sends the command to the file merger to upload the selected files from the local repository to its repository.
* Once the Code analysis is done the XML generator generates the XML files and the generated XML files are stored in the XML repository along with the server list and their corresponding file list.
* After Code analysis is done the client will be sent a notification message that the code analysis is done.

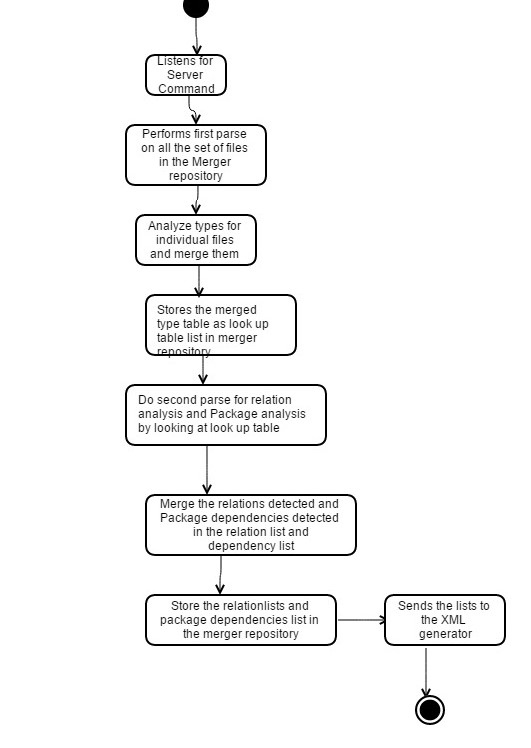
**Activity Diagram for Performing Analysis operation on the files selected:**



* 1. **Code analyzer tool activity:**

This explains the tasks performed by the code analysis tool in the server.. As discussed in the above activity, when the merger performs its upload operation the server gives the command to the code analyzer tool to analyze the list of files which are present in the merger repository. The steps in which the analysis is performed are discussed here.

**Activity Diagram for Code Analysis tool:**



Here are the list of tasks performed by this tool

* Code analyzer tool listens for the server command.
* In first parse it identifies all the types and merges all the types of all the files in the merger repository in to a look up table.
* This look up table is stored as a list in the Merger repository.
* It performs the second parser for performing the relationship analysis and package dependency analysis. This is done by checking in to the lookup table for deriving the relationships between the types that were identified.
* All the results of the relations across the files are merged in to a single list and stored in to the merger repository. Similarly, all the package dependencies across the files are merged and stored as a list in the repository.
* Once these lists are created, they are passed to its respective XML generator to generate the XML file.

**5.5 Activity for Performing Display operation:**

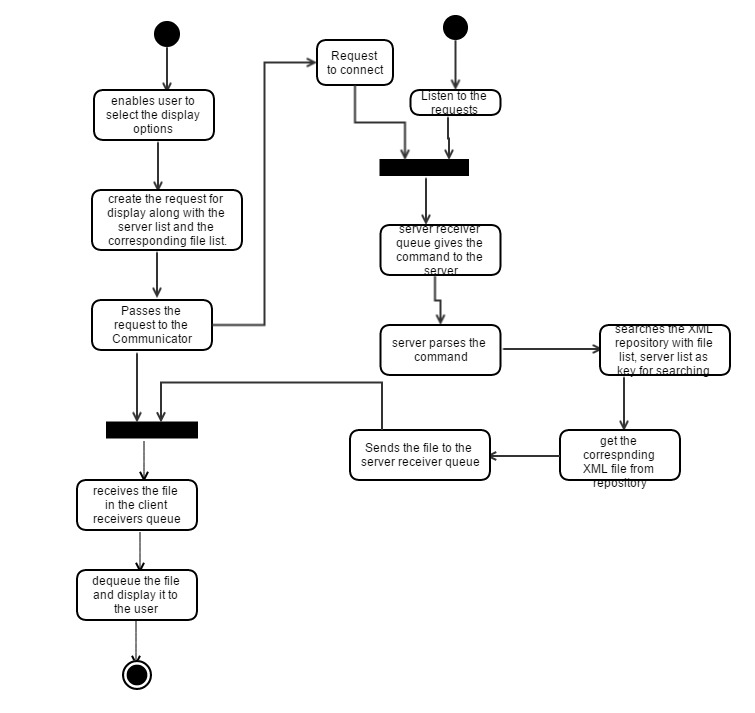
Once the server has analyzed the code, it sends a notification to the client. Upon receiving the notification the client enables the user to choose his display option. According to what option the user chooses the server will display in its GUI accordingly.

The detail explanation of this is explained by the activity diagram drawn below.

The list of tasks which are performed here are:

* Client enables the user to select display option
* Client creates a request to for the display with Display option, server list, its corresponding file list along with the command “Display”.
* This request is sent to the server and when the server parses this request it creates a thread which searches the XML repository which server name, file names forming the key. When the key is matched it takes the XML file according to the display option and sends the file as a request to the client. When the client receives the request it displays the XML file content in the display plane for the users to view the output.

**Activity Diagram for Performing Display task:**

****

# Critical Issues

Identifying critical issues will help a lot to identify the pitfalls that might occur during developing, designing and maintaining. This issues need a special focus to improve the performance, reliability, accuracy of the system.

* 1. **Cache Coherency and Cache management:**

This design implements a cache at a client side so that the network traffic is reduced and the performance is improved. But there might be a chance where the cache can give updated results since it is unware of what modification has happened to the files in the server side.

**Solution:** We can use flag or token whose value will be set whenever the files are modified. Thus the client will be aware whether the results returned are outdated or not.

* 1. **Existence of Duplicates files among the servers:**

There can be a situation where same file can exist in different servers. In that situation the client can happen to select the same files for processing, which creates unnecessary network traffic and results in redundant data at the output which is not reliable.

**Solution:** This issue can be resolved by using file metadata information. At the startup time, all the servers that are present remotely can collaborate among each other by comparing the metadata of the files and delete the duplicate files found.

* 1. **File Contention**

**“**File contention is a situation where we have two or more nodes attempting to transmit a message across the same wire at the same time”.

This tool is a client/server model which allows multiple client requests to the same server. All the requests are sent in a queue and they are taken by the server to provide service from the queue. The multiple client requests on the server are processed by cresting multiple child threads to do processing. Each child thread takes the request from the queue and performs analysis.

In this situation we can have multiple requests to analyze the same project on the same server. So child threads would be performing analysis on the same set of files at the same time resulting in the file contention.

**Solution:** To resolve this issue, we can provide locking system. The child thread which receives first will acquire the lock and once it is done with the analysis it releases the lock so that the other threads can perform analysis.

* 1. **Termination of connection between client and server:**

This tool is network based client/server application. In this situation there are chances of the connection being terminated between client and the server. Suppose if the connection terminates while sending the response to the requested client which has already sent certain amount of data to the client, then the client would end up in receiving incomplete message.

**Solution:** Acknowledgement messages can be sent between client and the server to inform that the response has been received completely. Otherwise this can be resolved by creating a thread which establishes the connection after certain delay of time. This delay time can be configured later in the implementation.

* 1. **Load Balancer going down:**

In this design when the client sends the message to analyze, the request goes to the load balancer for allocating the request to the server which has less load. This plays a crucial part in allocating the tasks to the server. There can be a situation where the load balancer will stop functioning properly. At this time the request to analyze will not reach the server at all.

**Solution:** To overcome this issue**,** constant monitoring should be done and alert notifications should be given to the stake holders who can resolve this issue.

* 1. **Archiving issues:**

The dependency analyzer tool allows analysis to be performed even after the file modification is done. If there are any updates to the new files the new version of the XML file should be stored in the XML repository. After a certain period of time this will have many older versions of the files which won’t be used.

**Solution:** Server ca be programmed in a way that it should be able to remove the files in the XML repository that are not used for a certain amount of time.

# Summary

Dependency Analyzer tool can be used by various users who want to perform analysis on their set of files or the files that are resident on the servers. Since this application is based on a client server module, the user can get perform the analysis of files from a remote machine. This tool works on the files which are based on c#. In further enhancements this tool will be enhanced to analyze the files of other languages also.

Each module is discussed in this document and a module diagram is given to show. The flow of the process from the user’s input till the output generation was clearly explained in the activity diagram. Critical issues of this project has been discussed here.

# References

* Sample OCD: “Test Harness “Sample (Given in Jim Fawcett’s Website).
* For definitions, sample codes, Diagrams: Jim Fawcett Lecture #1, #2, #3 and handouts
* MSDN – the Microsoft Developer Network

<http://msdn.microsoft.com/en-us/library/aa717050.aspx>