**Composite Text Analyzer (CTA)**

**Operational Concept Document**

**Piush Srivastava**

**Table of Contents**

[1. Introduction to CTA 1](#_Toc51016494)

[1.1 The goals of CTA 1](#_Toc51016495)

[1.2 The significance of CTA 1](#_Toc51016496)

[2. The Actors and Use Cases for CTA 2](#_Toc51016497)

[2.1 Actors 2](#_Toc51016498)

2.1.1 End Client....................................................................................................2

[2.1.2 Software Developer 2](#_Toc51016500)

[2.1.3 Project Manager 2](#_Toc51016501)

[2.1.4 Quality Team 2](#_Toc51016502)

[2.1.5 Software Architect 2](#_Toc51016503)

[2.2 Use Cases 4](#_Toc51016504)

[2.2.1 Use Case 1: User queries using single string 4](#_Toc51016505)

[2.2.2 Use Case 2: User queries using multiple strings 4](#_Toc51016506)

[2.2.3 Use Case 3: User queries for searching all strings 4](#_Toc51016507)

[2.2.4 Use Case 4: User queries for searching at least one string 4](#_Toc51016508)

[2.2.5 Use Case 5: User queries using single tag 4](#_Toc51016509)

[2.2.6 Use Case 6: User queries using multiple tags 4](#_Toc51016510)

[2.2.7 Use Case 7: User queries have recursive search 4](#_Toc51016511)

[2.2.8 Use Case 8: User can use command to create metadata 4](#_Toc51016512)

[2.2.9 Use Case 9: User views results of the text search 4](#_Toc51016513)

[3. A Command Line based Interface 8](#_Toc51016515)

[4. The Context of CTA 10](#_Toc51016517)

[4.1 The ‘File System Services’ interface 10](#_Toc51016518)

[4.2 Data flow within CTA 11](#_Toc51016520)

[5. The Modules of CTA 14](#_Toc51016521)

[5.1 Command processor 15](#_Toc51016522)

[5.2 Query Processor 16](#_Toc51016523)

[5.3 File Manager 16](#_Toc51016524)

[5.4 Token Analyser 17](#_Toc51016525)

[5.5 XML Processor 17](#_Toc51016526)

[5.6 Error Manager 17](#_Toc51016527)

[5.7 Relationship between modules and use cases 18](#_Toc51016528)

[6. Activities in CTA 19](#_Toc51016529)

[6.1 Relationship between the activities and modules 20](#_Toc51016530)

[6.2 Relationship between the activities and Use Cases 20](#_Toc51016531)

[6.3 The ‘Text Analyser’ activity 20](#_Toc51016532)

[7. A look at some critical issues 24](#_Toc51016533)

[7.1 Effects of voluminous input on memory usage and performance 24](#_Toc51016534)

[7.2 Security 24](#_Toc51016535)

[7.3 Versioning 25](#_Toc51016536)

[7.4 Flexibility and Usability 25](#_Toc51016537)

[7.5 Concurrent access of files 26](#_Toc51016538)

[7.6 Bad code 27](#_Toc51016539)

[7.7 Persistency within a CRA session (Use of hash tables) 27](#_Toc51016540)

[7.8 Persistency across CRA sessions (Use of XML data files) 27](#_Toc51016541)

[8. Summing it all up 29](#_Toc51016542)

## Introduction to CTA

This document details the software architecture pertaining to

### The goals of CTA

* CTA aims at searching text strings within a set of files placed in a particular path of the file system. CTA accepts the user query in the form of some file patterns and a path and some text strings and searches the files and returns the results to the user.
* CTA also accepts as input a set of XML tags (metadata) and searches for those tags within the files in the file path and returns the content of the tags for each file in the file set.
* CTA also accepts command to create the metadata file for a particular file.

### The significance of CTA

CTA can be a very helpful as a search tool in a code repository which has several source code files for several projects across several versions. It can be very difficult to analyze the functionality of the software by just analyzing the code. So by using CTA the software developer or the software architect can search for specific keywords across particular set of files in case he is interested in understanding the functionality of the software related to the particular set of files and keywords used for querying.

CTA can be very useful even to a librarian as he has to search the for specific files with certain keywords for every request he gets. For example if the librarian receives a request for a horror book and knows the name of the book "Frankenstein" then he can simply use this tool.

## 2. The Actors and Use Cases for CTA

### Actors

CTA can be used as a software analysis tool as discussed earlier as well as used for as a general purpose text search tool. So below is the list and description of the possible users of CTA.

#### End Client

Software companies generally prepare solutions for their clients. Clients are always interested in monitoring the progress of the project. So CTA can be helpful to them in case they want to search for certain keywords in a set of daily or monthly status reports. For e.g. they would search for the keyword "defects" or the keyword "test cases" in a particular set of daily or monthly reports to know the status of the progress of the project.

#### Software Architect

The Software Architect has responsibility to develop innovative tools to automate the process of software development. Also the knowledge regarding the working of the tools can be maintained in the knowledge repository. So this repository can be used by new employees for their training. They can simply search the knowledge repository with the keywords on the topics taught to them so CTA can be helpful to them. Similarly if the architect needs to do some enhancements in the functionality of the existing tools then he would also need to update the knowledge repository to include the knowledge regarding enhancement done to the tool. He could search for specific files along with keywords in the repository which need to be updated.

#### Project Manager

Project Managers use status reports which show the progress of the project or reports regarding the resources allocated to a particular project. CTA can be a useful tool for searching such reports given the keywords. Also Project Manager do perform code inspection of the code developed by the software developers. The tool can also be code inspection where the managers can search the code repositories for the source code files with keyword such as the name of the developer.

#### Quality Team

The Quality team needs to keep a check whether quality practices are followed for the software project. This tool can be helpful to them in analyzing the quality reports.

#### Designer/Developer

This tool is extremely useful for developers since they would need to search the code repositories with certain file patterns and keywords in order to search files associated with a specific functionality. Also they would have to analyze the impact of their code changes on the functionality of the overall system, so with this tool they can analyze the impact and also study the certain functionality of the software which would be difficult to achieve by merely looking at the entire code for analysis. For e.g. the developer wants to know that changes to a particular routine would affect which files in the code repository then this tool would be helpful.

### Use Cases

The following use cases describe the types of interactions that a user can have with CTA.

#### Use Case 1: User queries using single string

The user can run queries from command line in the form of file patterns, path of the files and a single text string and get results in the form of a file set.

#### Use Case 2: User queries using multiple strings

The user can run queries from command line in the form of file patterns, path of the files and a multiple text strings and get results in the form of a file set.

#### Use Case 3: User queries for searching all strings

The user can also form queries with multiple search strings and specify the option to search the files for matches of all the strings specified in the command line query.

#### Use Case 4: User queries for searching at least one string

The user can form queries with multiple strings and specify the option to search files for match of at least one string specified in the command line query

#### Use Case 5: User queries using single tag

The text analyzer also provides the user with the option of searching metadata contents of a text file stored in an XML file, based on the input tag in the command line query.

#### Use Case 6: User queries using multiple tags

The text analyzer provides the user with the option of searching metadata contents of a text file stored in an XML file, based on the multiple input tags in the command line query

#### Use Case 7: User queries use recursive search option

User can also search subdirectories in a particular file path by using the recursive option in the command line. For this type of usage the CTA will recursively search the input strings in the present directory as well as the subdirectories.

#### Use Case 8: User can use command to create metadata

The user can also create metadata file using the CTA tool which would result in creation of metadata of a particular text file.

#### Use Case 9: User views results of the text search

The CTA tool provides the user with the results in the form of text search results in case of string search in the file set, or metadata contents in case of tag search in a file set (XML) or a response regarding successful creation of metadata file.

The tool also returns appropriate error messages to the user in the form of results.

## 3. A Command Line based User Interface

The user can interact with CTA using commands fired from the command prompt. The user can type the search command on the command prompt and then specify the options for search.

* In order to search for text strings the user would have to type the command like search /T "some text" /T "more text"
* In order to search for all the text strings specified in the command line

in the file set the user would have to specify the /A option.

* In order to search at least one string specified in the command line in the file set the use would have to specify /O option.
* In order to search for metadata contents in XM file the user would have to

specify the option /M in the query

* In order to create metadata file for a particular text file the user would have to specify option /T (Description text), /D (Dependencies), /K (Keywords).
* The results returned from the queries would be written in a file and stored in another path as the results of the search may not entirely be visible on the console. Instead the result would be stored in a file and stored at a path. The user can then access this path and open the file to view the results.
* Error message would be visible on the console.

## 4. The Context of CTA

The following context diagram describes how CTA interacts with its environment.

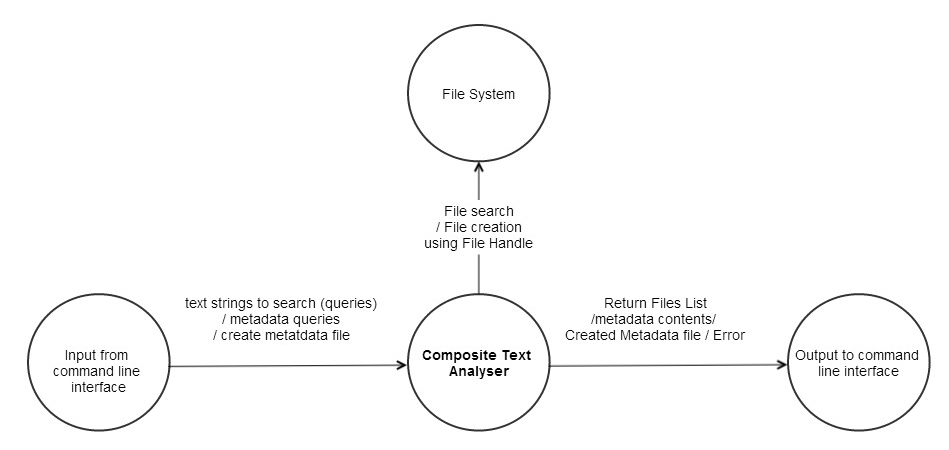


Diagram 1: Context diagram for CTA

As shown in the diagram above CTA accepts input from the user in the form of text strings to be searched within a file set, metadata tags (XML tags) to be searched within the metadata files or creation of metadata (XML files) for a particular text file through a command line interface.

CTA returns as output the list of files having the text strings, contents of the metadata tags within the XML file set or successful result regarding creation of metadata file (XML file) for a particular text file.

CTA uses the file system whenever it has to access a file for searching the text strings or the metadata contents. The CTA will use the file system services like open (for opening a file), seek/ read (for reading blocks of information within the file in order to search for the strings or tags) and close when the operation on the file is over. CTA will access the files in the file system using the file name and a file handle for a particular file.

### Data flow within CTA

The following data flow diagram depicts how the processing of CTA can be partitioned into processes and also depicts the flow of information that is necessary between these processes.

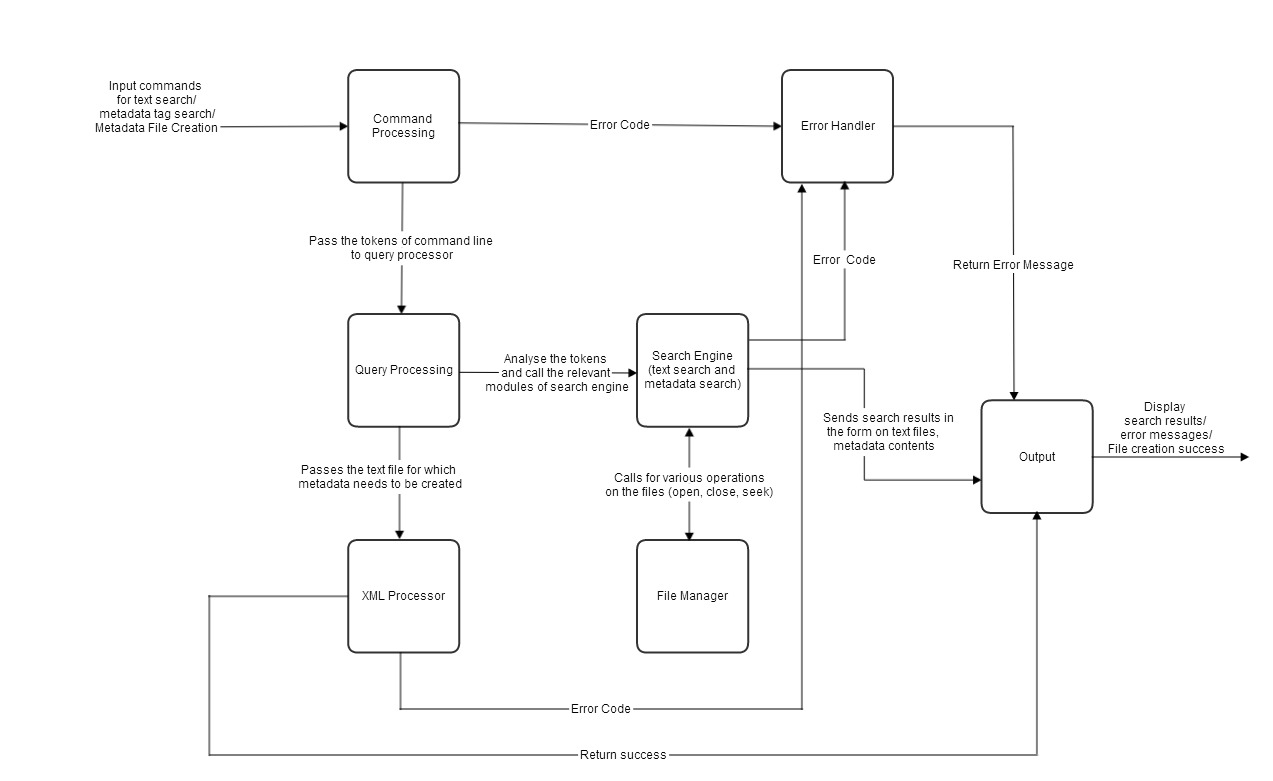


Diagram 2: Data flow diagram for CTA

The data flow diagram enhances the detail provided by a context diagram. More specifically, the data flow diagram depicts what happens inside the context diagram bubble titled CTA. It shows where the data input obtained from CTA’s environment is used in CRA’s processing. It also shows what part of CTA’s processing generates the output data that CRA gives to its environment.

As shown, the processing of CTA can be divided into seven ‘processes’. A glance at the name of each process reveals what processing it is responsible for.

* **Command Processing:**

This process acquires the input command given by the user and checks for the syntax of the command. It goes through each token in the command and searches the token in the dictionary to identify whether the token is a valid one. It returns error to the error handler module in case the command has an invalid token. If the syntax of the command is correct then the command processing module passes the control to the query processor.

* **Query Processing:**

This process analyzes the tokens received from command processing module

and decides whether a string search has to be performed or a metadata tag search is required within a file set or whether metadata file (XML) needs to be generated. If after token analysis it identifies that the search needs to be performed it passes the control to the search engine. If after token analysis it identifies that the metadata file needs to be created for a text file then it passes control to XML processing.

* **Search Engine:**

This process searches the text strings or the metadata tags in a specified file set based on a searching algorithm. It uses the FileManager process for performing file operations (open, read, seek) on the files present at the path. If it gets error then it passes the control to the error handler. If the search is successful then it passes the list of files to the output.

* **FileManager:**

This process is responsible for performing the file operations and is invoked by the search engine. It returns a file handle to the search engine.

* **XML Processor:**

This process is invoked by the query processor and is responsible for creating metadata from tokens received from query processor. In case of failure or error control passes to the error handler. In case of success control passes to the output.

* **Error Handler:**

This process is responsible for identifying the type of errors and returning control to the output.

* **Output:**

This process will display the search results/error messages/return status of the command on the command prompt

## 5. The Modules of CTA and the interactions among them

The following module diagram describes how the CTA’s processing can be divided amongst a number of cohesive modules each responsible for addressing a specific small chunk of CTA’s functionality.

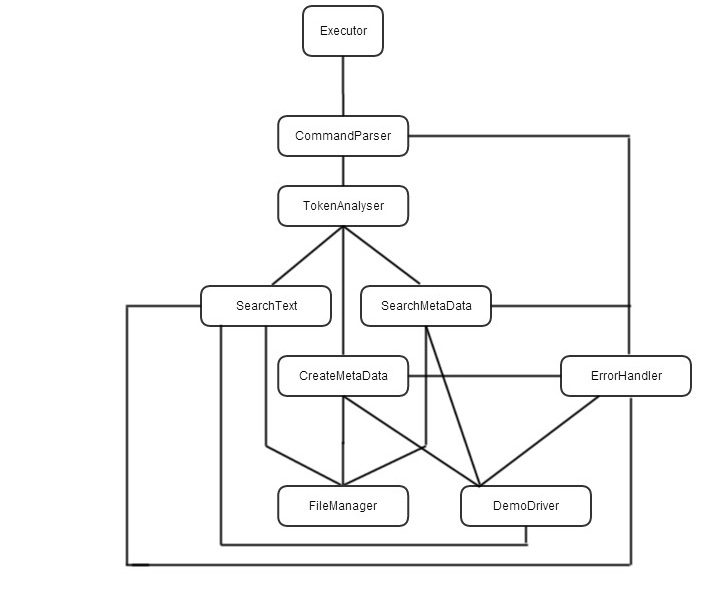


Diagram 3: Module diagram for CTA

As shown in the module diagram, CTA can be partitioned into 9 modules. The following list describes the responsibilities of each module as well as the interactions each module has with other modules.

### Executor

This module is the command line user interface for CTA

Some of the main responsibilities of the Executor module are listed below.

* Acquiring user input commands
* Passing the input parameters to the CommandParser Module

### CommandParser

This module is responsible for syntax analysis of the command.

The main responsibilities of this module are as follows

* Storing the tokens provided in the input command
* Searching the tokens in the symbol dictionary
* Returning error code for invalid tokens
* If there is no syntax error then the tokens are passed to the TokenAnalyzer Module

### TokenAnalyzer

This module is responsible for making the decision based on token analysis whether text search should be performed or metadata search should be performed or whether metadata file should be created for the text file.

The main responsibilities of this module are listed below

* Read the token sequence one by one until there are no more tokens
* Search the dictionary with the token to identify what operations (Text search, metadata search or metadata creation) need to be performed
* Based on the decision in step above call SearchText / SearchMetaData/ CreateMetaData modules.

### 5.4 SearchText

This module receives as input the text strings, the file path, file specification from the TokenAnalyzer and performs a search on the list of files. It returns the list of files to the DemoDriver Module. In case of error it returns the error code to the ErrorHandler module

### SearchMetaData

This module receives as input the metadata tags and performs a search with metadata tags in the file set. It returns the metadata contents of each XML file in the file set on success to the DemoDriver Module. In case of error the error code should be returned to the ErrorHandler Module.

### CreateMetaData

This module receives as input tokens from TokenAnalyzer module and creates Metadata file for the specified inputs. In case of success it returns the success status to the DemoDriver Module. In case of error it returns error code to the ErrorHandler module.

### 5.7 FileManager

This module is called from the SearchText, SearchMetaData, CreateMetaData modules and is responsible for performing various operations on the files like (open, close, seek).

### 5.8 ErrorHandler

This module receives the error codes from the programs and returns the error messages to the DemoDriver Module.

### 5.9 DemoDriver

This module is responsible for displaying results in the form of file list returned by the text search, metadata contents of each file returned by metadata search , error messages and the status of the metadata file creation

### Relationship between modules and use cases

The following table shows a mapping of which module caters to what use case. This is useful in visualizing what user actions affect which modules.

|  |  |  |
| --- | --- | --- |
| **Use Case No.** | **Use Case** | **Responsible Modules** |
| 1 | User queries using single string | Executor,  CommandParser,  Token Analyzer,  SearchText,  FileManager,  Error Handler |
| 2 | User queries using multiple strings | Executor,  CommandParser,  Token Analyzer,  SearchText,  FileManager,  Error Handler |
| 3 | User queries for searching all strings | Executor,  CommandParser,  Token Analyzer,  SearchText,  FileManager,  Error Handler |
| 4 | User queries for searching at least one string | Executor,  CommandParser,  Token Analyzer,  SearchText,  FileManager,  Error Handler |
| 5 | User queries using single tag | Executor,  CommandParser,  Token Analyzer,  SearchMetaData,  FileManager,  Error Handler |
| 6 | User queries using multiple tags | Executor,  CommandParser,  Token Analyzer,  SearchMetaData,  FileManager,  Error Handler |
| 7 | User queries use recursive search option | Executor,  CommandParser,  Token Analyzer,  SearchText,  SearchMetaData,  FileManager,  Error Handler |
| 8 | User can use command to create metadata | Executor,  CommandParser,  Token Analyzer,  CreateMetaData,  FileManager,  Error Handler |
| 9 | User views results of the text search | DemoDriver |

## 6. Activities in CTA

The following activity diagram depicts the high level activities in CTA and along with the flow of logic in CTA.

Note : Please zoom the diagram in order to view the details

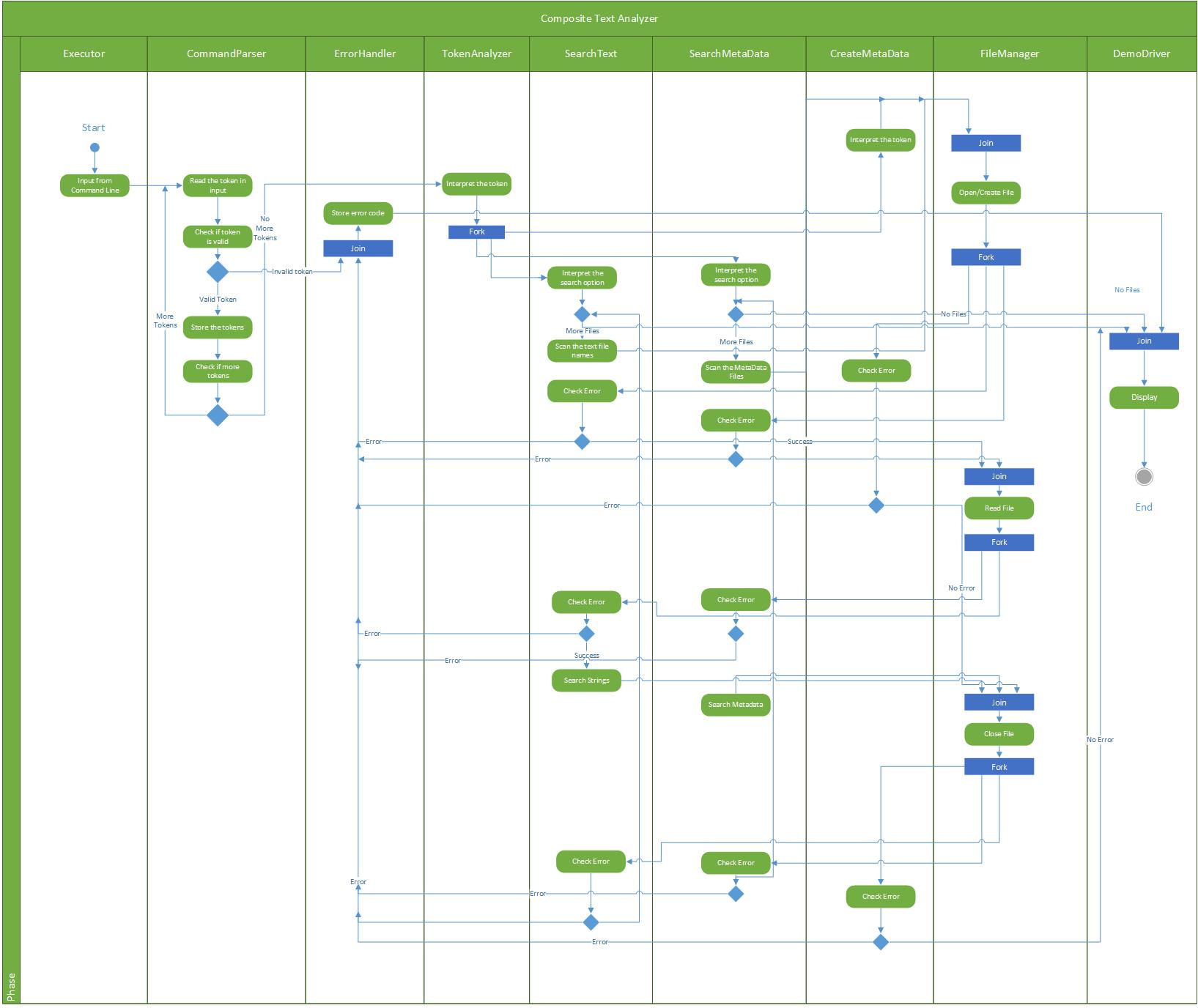


Diagram 4: Activity diagram for CTA

The activity diagram reveals all the typical activities that occur from the beginning to the end of a CTA session.

### Relationship between the activities and modules

A mapping can be observed between CTA's modules and the activities as shown in the diagram above.

* The activity of "Input from Command line" maps to the Executor module. The Executor module will provide the command line interface where the inputs would be accepted from the user from the command line.
* The activity "Read token in input", "Checking validity of the token" maps to the CommandParser Module. Since the CommandParser Module would be responsible reading the various parameters (token) of the input command and then checking whether the parameters are syntactically correct. The CTA maintains a token dictionary which would be searched for checking the validity of the tokens. Also there would be checks whether the input string to search is present in the command line.
* The activity of "Store Error Code" can be mapped to the Error Handler Module. Store Error code would collect all the errors from the different modules and display it to the command line interface.
* The activity of "Interpreting the token" can be mapped to the modules TokenAnalyzer. "Interpreting the token" would be the decisive activity and would decide whether the control should pass to activities in SearchText, SearchMetaData or CreateMetaData. This is the actual Query Processing Module described earlier. The activity "Interpreting the token" would also be involved in the module CreateMetaData as this activity would analyze the tokens and perform the processing of the metadata file creation based on the input tokens.
* The activity of "Interpreting the search option" would be mapped to the modules SearchText, SearchMetaData. This activity would be responsible for identifying whether it should perform recursive search, search all the given text strings in the input, searching at least one text string in the input or searching the metadata tags.
* The "Check Error" activity can be mapped to SearchText, SearchMetaData, CreateMetaData and is responsible for checking failure/ errors while processing.
* The activities "Open File", "Read File", "Close File" are mapped to FileManager Module. "Open File" is responsible for opening the file, "Read File" is responsible for reading the file, "Close File" is responsible for closing the file.
* The "display" activity is mapped to DemoDriver module and is responsible for displaying the search results, error messages, File creation success message to the user.

### Relationship between the activities and Use Cases

We have already discussed the mapping of Use Cases to Modules. A mapping can also be observed between CTA’s Use Cases and several of these activities. For example, the Use Case ‘Use Case 2: User queries using multiple strings’ can be mapped to the below activities

* 'Input from command line '
* 'Read the token' and 'Check if token is valid'
* 'Store the token'
* 'Check if more tokens' activity which will check all tokens in input command
* 'Interpret the search option' activity in the module SearchText
* 'Check Error'
* 'Display'
* In case of invalid token 'Store Error Code' activity would be called

## A look at some critical issues

Let us now look at some issues that are critical to the design of CTA. The design of CTA must address all these issues.

### Performance Issue

If there is a huge volume of text files to search within a particular directory then performance can be a critical issue in handling such huge file set.

Below are the two cases where there could be performance impact

* There are huge volume of text files in the search directory
* There could be huge text files on which the search queries are run

**Possible solution :**

* We can arrange the data in the files in a sorted manner (based on the ascii values) then perform a binary search on the data. In this way the search should be quick on given huge file set.
* Other solution would be to store the data within the files in hash tables having a key for a set of words. The key could be created based on the ascii values of the words in the file. e.g. all the words starting with alphabet 'a' in the text file would be given a key 1, words starting with 'b' would be given a key 2. This way by organizing data within files we can search strings faster.
* The data of the entire file set returned by the search query can be processed in batches. e.g. if the data in our file set is 1 GB and the size of the data structure the CTA is using to store supports 0.2 GB. Then CTA can fetch data from the file set in batches of 0.2 GB. So in this way 5 disk fetches would be required.
* In order to prevent repetitive searches we can maintain a history of previous searches and their results in a file. So if user fires a query on command line then the history would be searched first and if the query is present there then the results would be returned. If the query is not present in history file then the directory would be searched.

### Flexibility and Usability

**Issue :**

CTA module should be developed in such a way that it should can be integrated in other software systems as a third party tool. The CTA system should be flexible enough such that it can be merged or integrated with such third party tools and serve as a powerful search engine

**Possible Solution :**

The functionality of CTA should be highly modularized (split into various modules) and there should be effective message passing among the routines.

The CTA tool should be usable as well. For e.g. if there is a future enhancement in the CTA tool which has a requirement for a GUI. Then the module should be flexible enough so that the tool can be enhanced to support GUI functionality.

### Concurrent access of files

**Issue:**

There could be a case where two users concurrently try to create metadata file for the same text file. In this case there could be chances the data entered into the metadata file by the first user could be overwritten by the second user. To avoid such inconsistency there should be some concurrency control mechanism.

**Possible Solution:**

* A possible solution to handle this case is to create a temporary lock file if the metadata file is being accessed by one user. If the other user tries to create the metadata file for the text file then the program will first check whether there is a temporary lock file for the metadata file. If yes then the second user would receive a message that the metadata file is busy. Once the first user commits the changes, the temporary lock file would be deleted and would be ready for access by the second user.

### Bad code

**Issue:**

If the user gives following types of input

* The text string to be searched is very lengthy
* The number of text strings to be searched are too many
* The user specifies a binary file in the file patterns while giving input

**Possible solutions:**

We will look at the possible solutions to the issues mentioned above one by one

* The solution to the first issue would be to define the limit of the size of the text string that can be searched.
* The solution to the second issue is define the limit on the number of text strings that can be searched
* The solution to the third issue is to have a preliminary check on the input file type and return an error message that the program supports only text files

## 8. Summary

We have discussed on the below points pertaining to CTA

* Potential Uses and Actors
* User Interface (Command Line Interface)
* Context Diagram
* Data Flow Diagram (Architectural overview)
* Partitioning of the CTA into modules
* Activities taking place in the CTA
* Interactions between the modules
* Mapping of activities to the modules
* Mapping of use cases to modules
* Mapping of activities to modules
* Critical Issues