Composite Text Analysis
Operational Concept Document

Xincheng Lai
SU# 288299653
Syracuse University
CSE 681 – Software Modeling and Analysis

Revision 1.0
13 Sep 2013

Table of Contents

1. Introduction

1.1 Description and responsibilities

With the rapid development of information nowadays, people have tons of information stored in their computer or mobile devices. So they need a tool to support finding files by searching exact key words. This kind of tool is really convenient and power for people working in many kinds of fields. For example, a writer, who could deal with hundreds of articles each day, wants to find contents of an exact file, but only remember a few key words. Don't worry! Using this tool to find the exact file cannot be more suitable.

The Composite Text Analyzer (CTA) we call it is trying to fix this problem. This toolkit contains mainly two functionalities. One is helping people find the exact file set which hold fully qualified names with information stored in an associated metadata file, the other is a convenient tool for adding useful and related information to a specified file for searching from key words you have attached. This tool could be really helpful and convenient for everyone using text to storing information. Only he/she needs to do, is to generate the metadata file first and then he/she can search files by these key words some day later.

CTA's interface for people to use is nice but simple. It use commend line as input and output. People can input the format argument in the commend line to run functions, like creating a metadata file for a specified file, inserting attributes into the metadata, searching attributes in metadata file, choosing searching in a whole directory tree or not and finding the file set.

1.2 Executive Summary

This Operational Concept Document (OCD) is used to describe the project as described in the following specifications. The document describes this application from top-level architecture to specific module design, which will help the developer implement from button up, holding the understanding of the whole picture view of the application. The architecture part is enhanced by context diagram, activity diagrams describing the way each architecture's components talk to each other. After that, the design issues are presented illustrating concerns that the developer using this document may consider in implementation, including package diagram, events and critical issues.

1.3 Specifications

The Composite Text Analyzer (CTA) is required to support the following features:

1. Shall provide interface as main menu for manipulate the whole function options
   a) Support to choose use meta-tool to generate XML file for file.
   b) Support to choose file query by searching text strings.
   c) Support to choose metadata extraction.

2. Shall provide tool for creating one XML file for each file as metadata file.
   a) Supporting to input a specified path and the tool will generate a list of file name (except xml file) in this recursion file set on the screen for user to choose a file to set metadata. Or you can choose to go back to main menu.
   b) Supporting to input a file name in the previous list with some option to generate a XML metadata file. Or you can choose to go back tool's menu. The options are for inserting different attributes of the file into the xml file. Users can choose options like /T "Some description" to add description to this file, /D "A.h,A.cpp,B.h,B.cpp" to add the dependency files to this file and /K "This is some key words" to add the key words to this file. And after you choose any option to attach this file, the tool will generate a XML file in the same deep directory of that file.

3. Shall provide interface for query files by using text strings
   a) First supporting multiple arguments to set the scope for searching file, like files' pattern (only text file e.g.txt, .cs, .dat, etc.), path and recursion commend option"/R"
   b) Next searching files with text strings and options such as"/A" matching all strings or "/O" at least one string.
   c) Showing the searching results on the screen

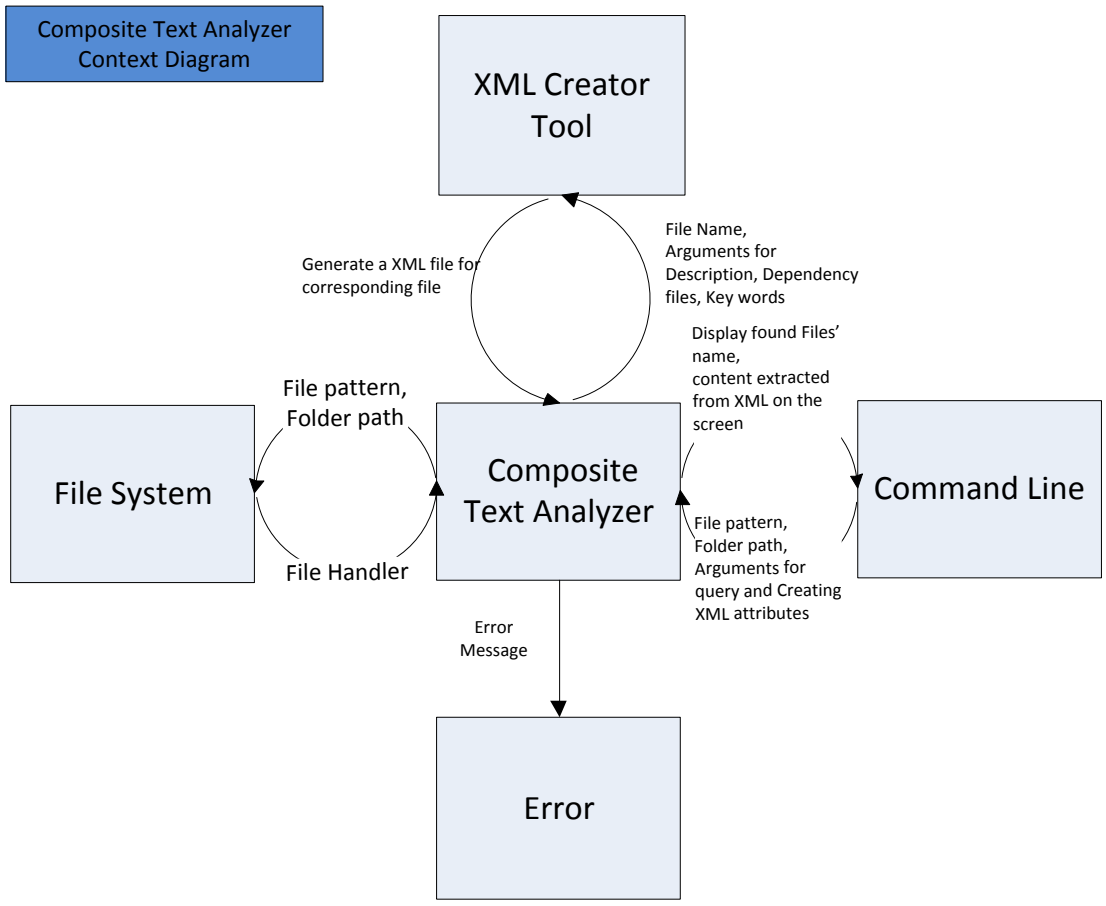
4. Shall provide interface for metadata extraction
   a) Supporting multiple arguments, like path, recursion option"/R" and searching tags of metadata"/Mtag1 /Mtag2".
   b) Showing the results of contents of each element of the metadata for every file in the file set.

2. Architecture

The system integrates all packages that run the independent function into a single environment. The mechanized in this system is that each tool which could be looked like package and they are partitioned into an interface and a set of core service, is initialized with different package with corresponding ensemble inputs and utilize the interface package of QueryProc to call the display package and then shows the results. The following context diagram describes the interaction between each layer.

2.1 Context Diagram

The context diagram below represents the interaction of Composite Text Analyzer with its external environment. The user provides inputs through the Command Line. The result of an analysis is also displayed on Command Line as well. Composite Text Analyzer also generates error messages when necessary and responds to error conditions with the help of suitable error handlers. It utilizes the system services in .Net Library to communicate with file system to carry on the file operations.



Need description for each parts.

2.2 Uses and Use Cases

2.2.1 Uses

There are six types of uses could be considered in the following sections:

1. Browsing all files' name with the specified pattern in a specified folder (which could be the whole directory tree rooted at that specified path).
2. Provide text analyzer to exact search files and approximately search files in a specified path by using strings.
3. Create XML file for storing the additional information attached to a specified file, which could be saved in the same directory of that file. We could check the dependency of file set with the attributes in the XML files; Check all keywords in the file set by search the tag" Keyword" of XML files; Check the description of each file in the file set.

2.2.1.1 Browsing all files' name

The most difference of this browsing way with Windows file browser is that CTA could provide straightforward view for user to see all files' name in a folder (include the subfolder). It could be convenient when looking into a folder which contains too many subfolders) to check the exact file exist or not. Because we need to unfold the subfolder in the file manage in Windows, Mac or other systems. The writer, developer, QA team and anyone who deal with many files could get benefit from this function in CTA tool.

2.2.1.2 Search files with exact search or approximately search

This is the basic and unique function of CTA, because most time we can just use the search application system provided to find the file based on the file name. But the most time, what we need is to locate the file in a specified folder just depended on a string text or a couple of lines of text. By this single function, we can get a good performance of searching results. So anyone who can only remember a few words of contents in a text file can quickly find out which file he wants.

2.2.1.3 Creating XML for related file

This tool could be really useful when people want to attach additional information to that file. The information could be some description of this file, creation time, progress of the project, key words for locate or remember this file. And the most import attribute for this XML file could be the Dependency tag, which is really convenient for developer, QA team, architecture designer and even the daily word user, like writer, reporter and so on. After adding the file names into the dependency tag in the XML file, we can find out what's the relative to this file. And then people can quick find out the linkage between those file to solve the problem. And the

principle is the same for using the tags of keywords and descriptions.

2.2.2 Use Cases

As this is a simple application with two main functionalities, the mainly user population is the content providers, like developers, writers, members of quality assurance team. CTA provide friendly and convenient interface for users, so there are a few inputs before the text analyzing and processing the search function and the output is straightforward with the command line.

2.2.2.1 For Developer's Use

To Record useful information into XML file in the same directory with that file. To check the dependency file related to files, find out the file set by using the tag, like keywords, descriptions in the XML files.

The steps in the Command Line use case are as follows:
Choose the Metadata Tool option in the main menu.
Then input a path or a file pattern to get the current files in the folder.
Input the argument like "filename /T", "filename /D", "filename /K" to insert the content to the XML attributes. Developers can also rewrite the XML file anytime they want.
In the main menu, choose the metadata query tool, input the argument format like "/Mtag1".
The searching result of content of XML attribute will be shown on the screen of command line.

CTA tools used:
Metadata tool, Metadata search, XML Reader, Command line.

2.2.2.2 For Writer's Use

To look for a file name buried in tons of text files and folders. To use a single word or a couple of words can get the qualified name of that file containing those words.

The steps in the Command Line use case are as follows:
Choose the Text query tool option in the main menu.
Then input a file pattern, path and argument"/R".

Input the key text for searching with format "/T keyword /T keyword2 /O" (or /A).

All the files name where contains those keyword will be printed out on the screen of command line.

CTA tools used:
Text Search tool, File Manage, Command line.

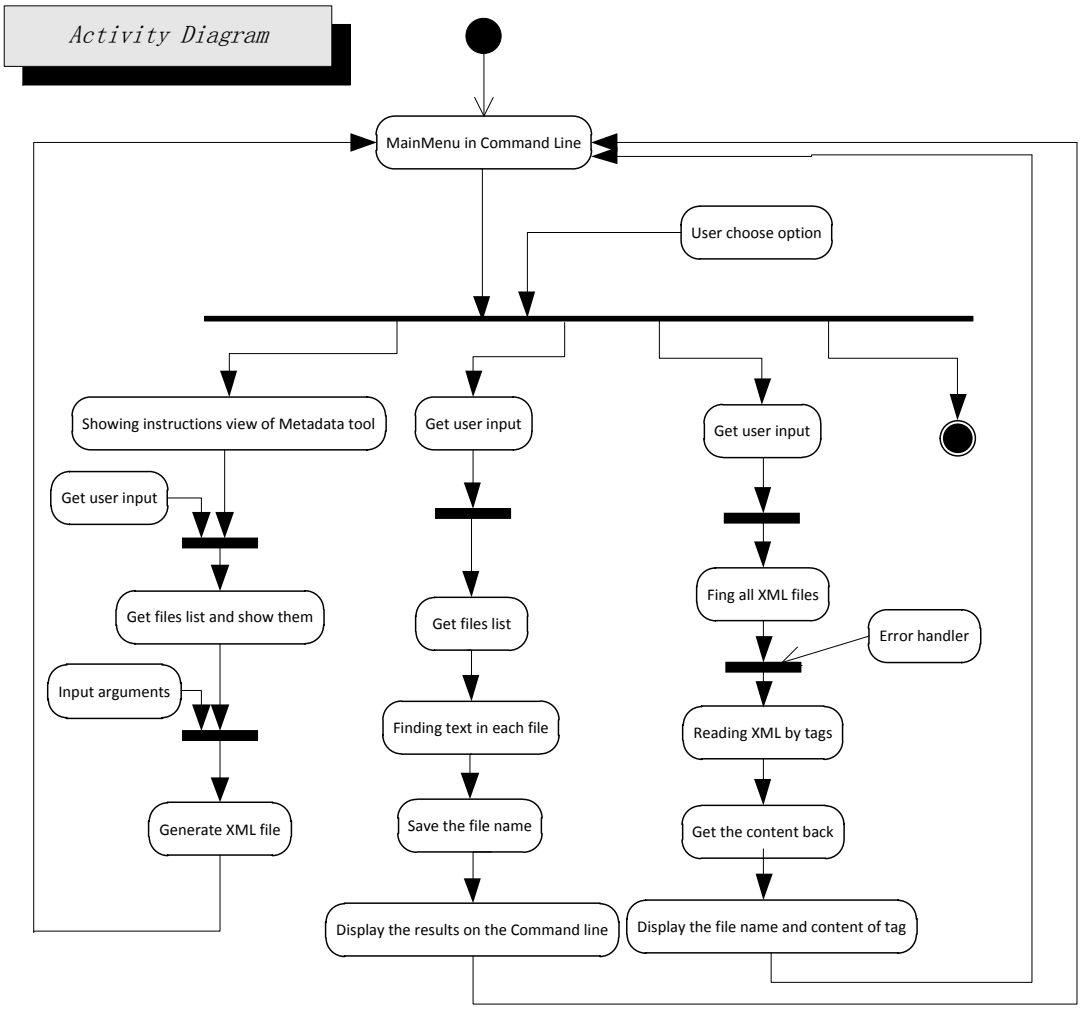
2.3 Activities

A Top-level activity of CTA could be:

1. Get user input
2. Get files list
3. First option: Generate XML metadata file
4. Second option: Find text in each file
5. Save the finding files in to a list
6. Display the find results
7. Reading XML by tags
8. Error handler to deal with defect metadata
9. Get contents of each XML file
10. Display the content results

The main menu is using command line and it waits for inputs given by the user. As long as the input is given, it displays the tree view structure. The users will have four choices in the main menu, where they can choose option to start to create a metadata XML for a file, start to query text file, extract the information from metadata file and end the application. In each processes, it provides the option to go back to main menu to start a new round operation.

The relationships between these activities are shown in the activity diagram on the next page.



Activity Diagram of Composite Text Analyzer (CTA)

2.3.1 Get user input

Different tools in CTA starts its processing after it gets the inputs from the user. The two higher-level activities are to read the pattern type of file and a path where to search files. But they are a little different, the one located in mid-left is need /Text /A or /O arguments for querying text files. The other one is used for querying content of metadata, so the argument for this part is /Mtag. As for the last User input activity on the left side, it is used for get the elements to insert in the XML file. Therefore, the arguments in this part are /T, /D or /K.

2.3.2 Get files list

It get files name depends on an argument "/R", if it says "/R", the list of file will get the all files in the directory tree. After get the inputs, the XML generator will show the results of all files name in that folder for user to choose a file to attach metadata file. As for the text query tool, it will store those file name in its list. And the XML extractor will get the path where to extraction information.

2.3.3 First option: Generate XML metadata file

Based on the previous input arguments, the XML metadata tool will generate a XML file in the same directory and store the corresponding elements and contents.

This activity could have some sub-activity run in this core module (to see in the following part). It will detect different situation based on how well the arguments have been input.

1 Check the file existence. If the specified file is not existed, the command line will get the error message, saying that we cannot create a metadata when the file doesn't exist. And return to input activity. Or continue followings.
2 Check the correctness of the input arguments. Here the argument for XML generator is /T /D /K, if the user unconsciously input other arguments, it could also trigger the error message. An return to input activity. Or continue followings.
3 Open a new file handler for XML file. If the this XML file cannot be open, skip this file and emit an error message to command line with file name.
4 Based on the different arguments; use the XML writer to insert the content with tags into the XML files.
5 When creating the XML file in the same deep directory of that file, if that file already has an XML file, it will be rewritten the new contents with that tags. So the previous contents in that XML file will be gone.
6 Finish generate metadata file and close the file.

2.3.4 Second option: Find text in each file

After CTA gets the fully qualified names in the folder set, it can start to search files. The following activities are done for each file in the list of all matching texts or match at least one text, which depends on previous argument, /A or /O:

1 Check the file usage. If the searching file is not existed or cannot be opened, the command line will get the error message, saying that we cannot open that for searching text. And return to input activity. Or continue followings.
2 Check the correctness of the input arguments. Here the argument for searching the file are /Text, /Text /O or /A, if the user unconsciously input other arguments, it could also trigger the error message. And then return to input activity. Or continue followings.
3 When the program looks into a file, it will load the content of that file to memory and compare it with input texts, if the arguments say"/O", the detector function once finds at least one matching text from /Text 1 to /Text n; the result list will store the name of this file. If the arguments say"/A", the detector will come through all text searching. Until the last Text N is found, it can return true, and program stores this file name into the result list.
4 When finishing one file searching, then we close that file.
5 Until the program goes through all files in the set, display the finding results on command line.

2.3.5 Third option: Reading XML by tags

In this tool, after CTA gets the path from user input, it can start to search metadata. The following activities are done for each metadata file in the list of all matching tags:

1 Check the metadata file usage. If the searching file is not existed or cannot be opened, the command line will get the error message, saying that we cannot open that for searching text. And return to input activity. Or continue followings.
2 Check the availableness of the related text file. If the related text file is not existed, the command line will get the error message, saying that we lose the source file which has a metadata file"…". And return to input activity. Or continue followings.
3 Check the correctness of the input arguments. Here the argument for searching the file are /Mtag1, /Mtag2. If the user unconsciously input other arguments, it could also trigger the error message. And then return to input activity. Or continue followings.
4 Using the XML reader package to read the content from XML file with provided tags. And then put the contents into a result list.
5 Close that XML file and display the results of content.

2.4 Summary

The top-level architecture of the Composite Text Analyzer is composed of three layers: UI interface, processing part and error handler. Each layer performs a certain function. The application would use the UI layer to interact with the user. And the

processing layer is used to perform all functionalities, including generator XML file, doing XML extraction, File searching, and input validation. As for the error handler layer, it will come through all the processes of the application, it will instruct users to input the arguments in the correct way. The following is the main part of each module's function and how they can get interact with each other.

3. Details Design

When having a high-level understanding of CTA architecture, more details design will be given in this part in the OCD file. We need to separate the task singly and make them partition with modules (packages). After that the developers could start to button up coding modules to realize the whole application gradually.

3.1 Critical Processing Modules and their Interactions

CTA consists of eight modules:

1. Executive Module
2. Command Line Module
3. QueryProc Module
4. Metadata Tool Module
5. XMLProc Module
6. Text Search Module
7. Metadata Search Module
8. FileMgr Module
9. XML Reader Module
10. Error Handler Module
11. TestDriver Module

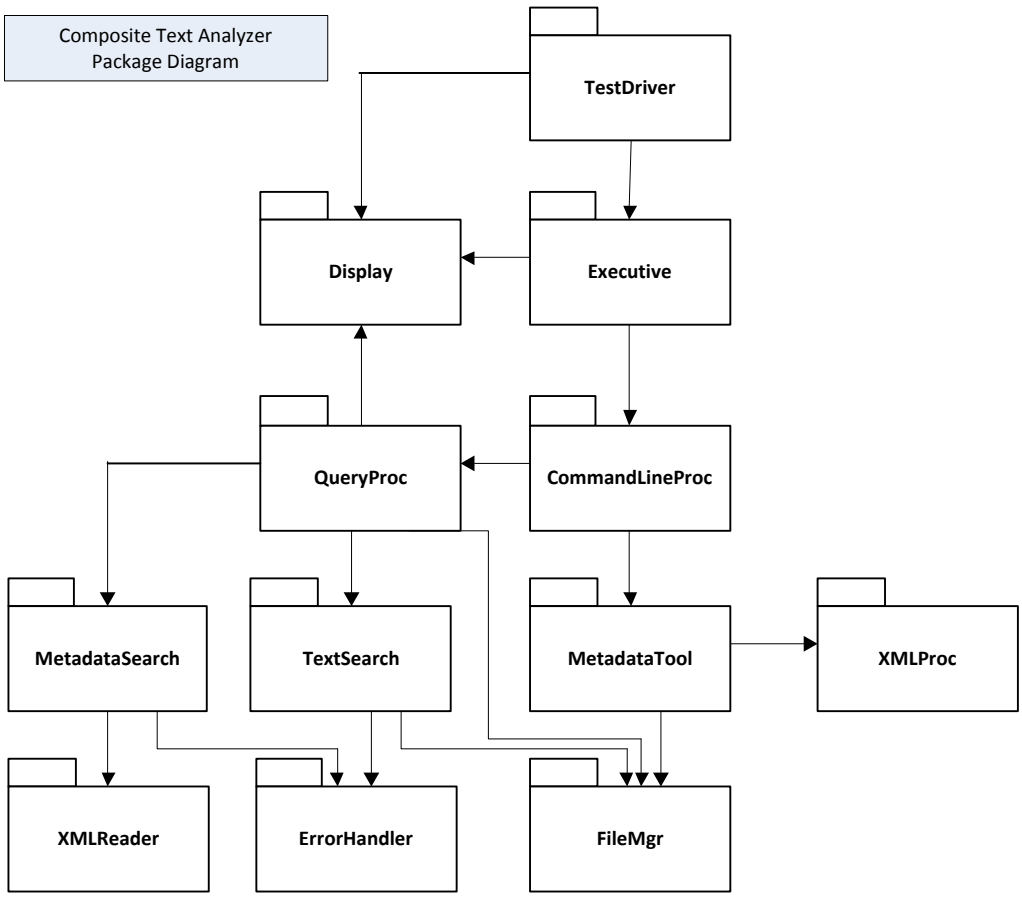
As shown in the module diagram below, Executive Module is the executive main module that drives the CTA processing, and it depends on several packages:

1 Display module, will show the instructions of operations and the result list of each query from QueryProc module.

2 Command Line Proc module, as UI for user to choose run Metadata Tool or querying. After users have chosen one operation with correct arguments, it will generate object of QueryProc Packge or MetadataTool package.

As for lower level modules like XML Reader Package, XML Proc package and FileMgr package, these could support the main querying function module. And these three modules are used as written by the original author, Dr. Fawcett.

Following is the diagram of packages and with the description of responsibility and interactions between each package.



Package Diagram of Composite Text Analyzer

3.1.1 Metadata Tool module

This module will get the a path from Command Line and then push this name to FileMgr package, where it get back file names in that path, showing all file name on screen for user to choose.
It is also interactive with the Command Line proc, which means that it will extract the

arguments from it and then reuse the module XML Proc to write metadata file, where contains the class XML writer and XML reader, which provided by Phd. Fawcett.

It has the following classes:

- FileMgr: get all file name in the path.
- XML reader: read an existed metadata file.
- XML writer: insert content with tags into the XML file.

3.1.2 QueryProc module

This module could be looked as a factory package, which will interact with Display package, Text Search package and Metadata Search package; it will get the arguments from command line to create the corresponding searching module. It provides interface class to display and search.

It has the following classes:

- IQuery: provide the interface for text search and metadata search.
- Display: showing the results to the command line.
- Text Search: run the main function of text search.
- Metadata Search : run the main function of metadata search.

3.1.3 Text Search module

This module will extends from IQuery package, where has already given some interface needed to be overwritten. The main function is provided for searching the file name by text strings, one or more than one. It needs to interact with two packages, FileMgr and ErrorHandler package. The core function to find file name is accepted two kinds of argument,"/O" and "/A".

It has the following classes:

- Text Search: provide the main text searching functions.
- FileMgr: get the searching scope of the specified path and return the whole file names in that scope.
- Error Handler: when the User provides invalid arguments, it will send error message to command line, skip the error file and to continue the searching processing.

3.1.4 Metadata Search module

This module is also extended from IQuery package, where has already given some interface needed to be overwritten. The main function is provided for searching the metadata contents in the searching scope. It accept at least one argument"/Mtag", so it will go through all xml file in that scope and use XML Reader to get all content into a result list. If a file does not have an associated metadata file, the analyzer will emit an error message and continue processing the remaining files.

It has the following classes:
Metadata Search: provide the main metadata searching functions.
FileMgr: get the searching scope of the specified path and return the whole file names in that scope.
Error Handler: when the User provides invalid arguments or a file doesn't have an associated metadata file, it will send error message to command line, skip the error file and to continue the searching processing.

3.1.5 Error Handler module

This module is created a showing error message to command and it's a good display helper when coding the program and showing the searching results.

It has the following class:
Exception: handle the error situation and showing the error result on the command line.

3.1.6 File Management module

This is a flexible module to find folders' name, files' name, where the most codes are provided by PhD. Fawcett. And the module will traversal and match pattern based file finder class. This class is based on directory navigation and command line argument parsing demonstration programs.

It has the following class:

- FileMrg: This class has several main functions: One is findFiles function which goes through all subdirectories under one directory, when it get the argument"/R". The other is to get all the files matching a pattern string. The usually operation is connect with a path and a pattern.

3.2 Events

The important events are captured by warning to be shown on the command line throughout processing. They are several non-critical events listed here as well, for reference.

- No matching files to process: when users having input the first phase arguments like pattern and path, there are not any matching files existed. So the CTA stops processing of search, but it can still have option for generating XML metadata file.
- Files in that scope are not text file: If the finding results of all files in the first phase are not the text files, CTA still cannot process the text query. So it will give warning information on command line and then back to main menu.
- The performance when the CTA deal with a query in a great large file sets. The way CTA storing a large file sets' names is using list variable, it has enough space for storing huge amount of name in that list. But as for the content of a large file, the current way is to use System provide Stream Reader to extract all the content of that file into a string. And then do the querying function. It could be low performance, if querying in a large set of large files.

3.3 Summary

Package-oriented decomposition of the high level modular architecture results in the Package Diagram presented. This is a good way for developer to start to write the code for single module first and do test it work well. Then they will be able to integrate those modules as a whole system. As we consider these non-critical events, it shows some preventable event we can deal with, and it can also give a better performance.

4. Critical Issues

4.1 Error arguments

Situation:
Error arguments always happened when the user unconsciously input the keyboard or without notice the format of arguments, like "space, slash and space or just slash".

Solution:

Each time the users need to provide arguments, the command line will first give an instruction of current arguments needed to be given. And then in the code, first it will detect the number of the argument and then test whether the input arguments are valid or not. If they are valid inputs, they will be passed into the following procedure. However, if the inputs are invalid, it will send an error message to the command line, and let the user to input the arguments again.

4.2 Unreadable file

Situation:
It means that when the CTA come about to carry the query processing, the queried file cannot be open to read. There are many reasons causing these problems, like the file's format cannot be read with the API system provided, protection setting and the privilege of that file, disk(or network) hardware problems. But the most common situation is the conflict, which means that other application is concurrently using this file, executing the reading or writing function.

Solution:
When the application tries to open a file with API the system provided, it needs to check the file handler first each time before the program tries to read or write it. If the current file is being used, it cannot process operation and return a warning or error message to command line. And the application can start next round to read the following files. Although it's a little bit annoying, it's a good and safe way to keep application run well.

4.3 Existed metadata file

4.3.1 Existed XML file without associated text file

It means that the associated file is missing. It always happens if a file or one of its parent directories get renamed or removed, when the CTA is being processing.

Solution:
Before the CTA does the metadata query, it needs to check the existence of all the associated file with the metadata. If that file is not existed, it will cause an error message on the command line and continue to do metadata query. And the metadata extraction will not get information from that metadata, so the result could not include that content of the metadata.

4.3.2 Existed XML file need to be re-write

When the users try to use the XML creator to generate a metadata for a file, the metadata file could already be there. So what we decided is to re-write the metadata attached that file and the previous metadata will be gone. But before rewriting the metadata, it will send a warning message on the command line to let the user know that this file could be rewritten and lose the previous data.

5. Conclusions

The purpose of Composite Text Analyzer is to support management of large sets of document files.
It is really convenient toolkit for any work areas.
It is a good way to manage documents files with associated metadata file, which can be created by uses themselves.
It provides clear interface of all operation instructions for user.
It provides easy understanding result showing on the command line.

6. Appendix

Class diagram

Prototype code, will be shown on project 2
Executive.cs
CommandLine.cs
QueryProc.cs
MetadataTool.cs
XMLProc.cs
Text Search.cs
Metadata Search.cs
FileMgr.cs
XML Reader.cs
Error Handler.cs
TestDriver.cs