Remote Document Vault
Operational Concept Document

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

Revision 1.0
21 Oct 2013

Table of Contents

1. Introduction

1.1 Description

With the rapid development of information nowadays, people have tons of information stored in their computer or mobile devices. However, the invention of Internet makes people share their information more and more convenient. One innovative idea coming out is that we can create a remote storage from which we can upload, extract and query information whenever and wherever. In this project, we want to create a software including the robust remote server and friend client user interface, called Remote Document Vault (RDV).

The purpose of the RDV is to enable insertion and extraction of text files to a remote location and to display information about their properties and relationships. The client side have a menu to choose function of service provided by the server side. On the client side, it uses Graphical User Interface (GUI) to display information about its file categories and files of server. Then clients can click a file to take a look at contents of this file and its corresponding metadata in another windows form. Besides, RDV can provide clients a view about the dependency relationships of a current file, including its parents and children.

Except extraction information from the RDV, the clients are also able to upload files into the RDV. In the meanwhile, the client needs to provide some properties about this file and the server will automatically generate a metadata in the RDV. What is more, the RDV supports that text query and metadata query for the documents in the server side. And the clients can modify the existed metadata in the RDV.

So we can see, the functionalities RDV are really useful and powerful for many fields, like developers, QA, editors, teaching system, network society and on so. We will discuss this case in details in the following sections “Use and Use case”.

To fully implement these functionalities and make a robust software, there are many existing critical issues needed to be talked about before we start to develop the programs. And then we can find a most reasonable solution for that issue. Like authenticated check-in, versioning, large files, concurrent issues, error handling and managing files and so on. And these critical issues will also be discussed in details in the following part.

1.2 Executive Summary

The Remote Document Vault is very useful tool which could be used in many fields, however, the user interface is convenient and friendly for people to use. The Remote Document Vault is in an environment with User (for input/output), Remote Document Vault Service and Network Resources (for communication). The modular architecture of this software is designed with concept of layers: Presentation, Communications, Processing and Data Storage.

The tool is developed as 'Client' Module and 'Server' Module and most major part of both these modules are actually a reusable component which including the following parts:

Client:

- Communications: transmits data from sender to receiver
- CommandLineProc: interprets the extracted information and do action upon different messages.
- ClientFileManagement: browse the local files and slice them into blocks to be sent

Server:

- Navigation: navigate files in categories on the server
- ServerFileManagement: save the file from clients and read the file on the server
- Metadata Tool: generate metadata file on the server
- Metadata Query: search element on each metadata file in a specified path on the server
- Text Query: make exact and fuzzy text query on the server
- Relationship Tool: separate tool for generating map of files relationship, which includes parents relationship and children relationship

Principal users of RDV will be software professionals, includes developers, QA and so on, but not excluding non-software-technical end users like editors, academic users, network society, Blog and daily users only sharing documents with people. Because it provides very friend user interface, the clients don't need to be worried about the service on the background and they can only follow the instruction on the client view.

1.3 Specifications

The Remote Document Vault is required to support the following two main parts, the client side and the server side. Both of these need to be implemented as much of their functionality as practical in separately compiled components, like a dynamic link library with interface and object factory.

The RDV client side should be use of Graphical User implementing by Windows Presentation Foundation (WPF) including the following requirements:

1. Provide a management view that supports local file browsing to upload with metadata and insert into the RDV's specified category folders, which needs to be implemented by Windows Communication Foundation (WCF).
2. Show a list of categories in the RDV, when click one of these categories. There is a view to show all the documents in this specified category.
3. Another view needs to be used for showing the content of a file that has just been clicked. And in this view, the corresponding metadata of that file should also be display near the content view. Besides, it needs to provide a way to go back to category navigation view.
4. Provide text query function. Client can input the searching strings and choose exact or fuzzy querying. When the service is done, this Text Query UI could should the fully qualified name of the matching files. When clicking one file of returning result list, it will come to a view that shows this file content and its metadata.
5. Provide Metadata query function. Client can input one or more categories and metadata tags to search the contents of each element of the metadata for every file in the file set. When clicking one file reference in result list, it will come to a view that shows this file content and its metadata.
6. Provide a metadata edit view, which allow clients to modify a metadata file by supplying a set of tag names and values. If the tag exists in the metadata file being edited the value replaces the old value. If the tag does not exist in the edited file a new element is created with that tag name and value.

The RDV server side should be implemented package by package, so that the server could be more flexible for other application, and it's easier and efficient to build up a robust server. It includes the following requirements:

1. Separate relationship tool that can search the metadata file to build a map of parents for each file in the Vault. It needs to update relationship each time when the server starts and some files were inserted into the Vault.
2. Provide a File management that could create category depending on the name of categories sent from the client side. And it could also implement function to insert files into that created categories.
3. Separate metadata tool will automatically generate a metadata when a file is inserting into the Vault. And the content of that metadata should be depended on the content send from the client side.
4. Separate Text query tool can handle different querying requirements from the client, which already implemented in the project2.

5. Separate Metadata query tool is dealing with different querying requirement from the client, which supports multiple categories and multiple tags metadata search. As for querying result, it always returns file reference and categories.

2. Architecture

The Remote Document Vault will depend on the service provided on the remote server side; however, the discussion here revolves around context in which it is used by a user. The system integrates all packages that run the independent function into two sides, one is the server and the other is client.

2.1 Context Diagram

The context diagram below represents the interaction between client side of Remote Document Vault and its server. The clients have different views to manipulate with server, like extracting documents, checking the relationship, uploading documents and doing querying. According to different action from the clients, communication part will take care of sending messages, which will be encoded with different contents to represent exact one action. And after the messages are sent to the server, the service will do running according to that message's contents. Finally, after the service is done, it will save some records on the server side or just send back the extraction or querying result to the requesting clients.

Figure 1 below depicts context of RDV.

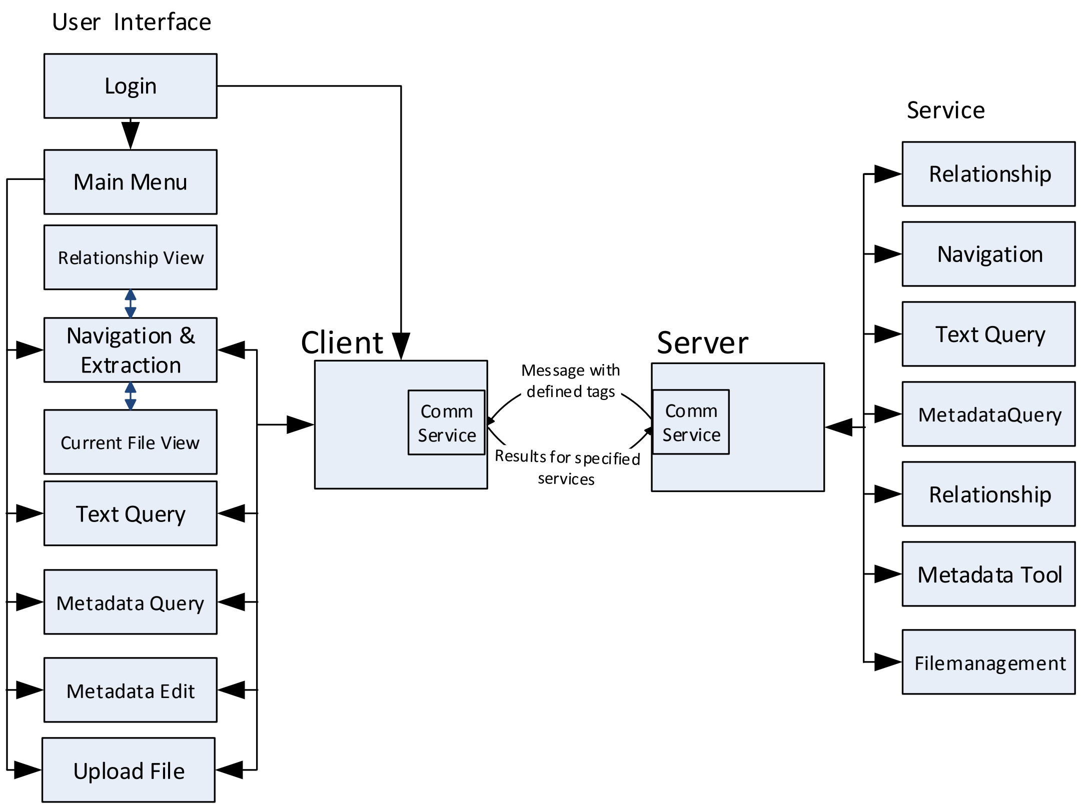


Figure 1 Context Diagram

2.2 Uses and Use Cases

2.2.1 Uses

When talking about the uses, we need to think about what the service in the server side we can utilize. The stuffs in the client side are just user interfaces, it almost do nothing service about the vault on the server. And the main functions in the client side are just friend view for clients to interact with the server. In this project, there is File management service on server can help users to save files on the server side. The Navigation service would help the user have a better understanding of the file system on the RDV. The metadata tool could be used to generate a metadata file on server to store the additional information about this file. The Query service including metadata query and text query, which could help clients extract the information he/she cares about. The separate dependency relationship tool could be used to check the file and get a view about the parents and children file of a specified file.

2.2.2 Use Cases

This section attempts to identify principle users of the *Remote Document Vault*, ways in which they will interact with the tool and what features will be incorporated to suffice needs of all users.

2.2.2.1 Principal Users

Due to the RDV's friend and straightforward user interfaces and its robust and targeted service, this product could be used in many fields. For example, software industry, news agency, education teaching system, networking society, normal documents sharing users and so on.

Here I categorize the principal users into three groups:

- Software professionals
- Documents users
- Academic and teaching users

In the sections to follow is described who all belong to each of above category of users, what they expect from RDV and how they could utilize our product.

2.2.2.2 Software Professionals

This category contains the people who are getting most benefit using this product. It means that the RDV will be used most frequently than other categories.

- Software Architects

Software Architects are responsible to get the information from the company's clients, and category all their requirements and then turn them into project specification. After that, the software architects need to decide the implementation of technical parts. So when they lay down the technology, then they can start to divide the project into simpler modules which has complete logic and could be reusable in the following tasks. After they make project architect diagram, they can dispatch these single task to the developer and create a category folder in the server side with this tool. In case of a very large project, there will be many different modules need to be developed and test single by single. So using RDV could be convenient for they work together, and it's efficient way to keep the files with categories.

Expectations: Remote Document Vault will prove to be beneficial to an Architect:

a) Efficiently manage separation of a big project by creating category to manage file for developer, tester and so on.
b) With metadata tool, it can provide additional information of files for all of people in the developing process.
c) The dependency tool could also be helpful for architect to design a whole picture for the product.

Conclusion:

The architect uses this project can efficiently separate their project into single task with category folder for the developers and QA tester, and assign additional information about the document in the metadata file. Because the documents will be store in the RDV server side, it not only means that other project workmates can extraction information from RDV but also the file will be updated for them when architect change the document.

- Software developers

Software developers need to make code to transform the logic, requirements into reality. After the architect finished the separation of project, the developers will be able to extraction information from the RDV according to their tasks. And then when they separately developed a task, they can insert the files and provide information in the metadata file to the according category folder. Besides, the developers are also can utilize the dependency tool to take look at the process of developing.

Expectations: Remote Document Vault will prove to be beneficial to developers:
a) Can efficiently navigate the file system in RDV, and need to have a separate category folder to insert related files together.
b) Can extract the task information from architecture through remote server.
c) Can efficiently for developers to work on their single task and manage the files easier.
d) Can support query text file and metadata query when they need it.
e) Check the file relationship and make the file dependency complete

Conclusion:

The developer uses this project can efficiently work on their single task. They can insert their task into the RDV reasonably. They can also extract the additional information from the document's metadata. Besides, text query, metadata query and dependency check are also convenient for them to implement their task and have a whole picture about the file relationship.

- Quality Assurance team

Quality Assurance team deals with the performance of product and its satisfactory under all kinds of circumstances and all environment. It is crucial part in a software company, cause the person in developer team cannot always think about all these stuffs and what he implement might deviate from the trend of the market or other new features. So in this way the QA team is another chance for a product to redesign better. And it will submit their feedback to developer teams and let them to fix that bugs or add some features. After that the products will also need to be pass to QA team again. So it is a circle from developers to QA team until a product has the best performance.

Expectation: RDV performs some of the final tasks before product deliverance to the customer and thus RDV could also be useful in this time:
a) Extract the information from category and check the content of the document
b) Send the feedback to the RDV by modify document's metadata or just insert feedback documents
c) Check the correctness of the relationship about the parent and children files
d) Can make a text query or metadata query when they need it

Conclusion:

The QA team can use RDV to extract information of document and its metadata file that could store the specification of that file efficiently. And it's easy to send feedback conveniently back to the RDV. So that's two main work of QA team. Besides, they can also take look at relationship of the documents and check the correctness of the file and its parent and children files. Make queries when it's necessary.

2.2.2.2 Documents users

The normal documents users could be anyone who needs a server repository to store their documents. And they want to write, edit, modify files in anywhere then they can insert these file in to a place called remote document vault. Besides, it is a good way to store the personal file and they can also add some additional information to that file. Except personal use, this vault could be used as a shared platform. Using the RDV, families, friends, and classmates could extract the information of the documents. Also this tool could be a part of a large network society or a part of Blog, space on the Internet.

Expectation: RDV performs document vault for these people, because they need a way to store files in a remote server, and then share with others:
a) Upload file to server and there is a metadata to store the extra information attached with that file
b) Extract the information from category and check the content of the document
c) Extract the information of metadata

d) Navigate the documents on the server

Conclusion:
The RDV provides many service related with documents. Here as the normal document user, it provide the core function for users. In most case, these users just need to follow the friend UI and finish the uploading file and its related metadata value. Next time when he or she logins in the system, he or she can utilize the navigation view find categories and files and then can extract the information they uploaded and take look at the additional information saved in the metadata.

2.2.2.3 Academic and teaching users

These users are academic and education related stuff like instructor, teaching assistant, research assistant, professors and so on. For example, the university provides professor several education use server for teaching system. The professor can run our server application on its server and provide his or her students the client side of our product. So the students will be able to send assignment files to the server according to his or her name which will be created as a category folder in the server. Of course the file could be attached with metadata file. When the professors, TA and grader are going to judge their assignment, they can also use client application to loin in as their position and then they can extract the documents the students uploaded and grade them into metadata files. So next time, after the students login in, they can extract the modified metadata to see their grades.

Expectation: RDV performs document vault for these people they can login in as different position to have different right to manipulate the documents:
a) Upload file to server and there is a metadata to store the extra information attached with that file
b) Extract the information from category and check the content of the document
c) Extract the information of metadata
d) Navigate the documents on the server
e) Modify the metadata on the server
f) Metadata search for specified search, like keyword, deadline and so on

Conclusion:
The RDV provides the document uploading, extraction, metadata creation, metadata query and metadata modify. All these function will be very convenient for these education stuffs to communicate with their students by files and metadata. Besides, the clients have a good UIs and the server is also managed by categorized folder, so RDV is very efficient to them in this case.

2.3 Activities

A Top-level activity of RDV could be:

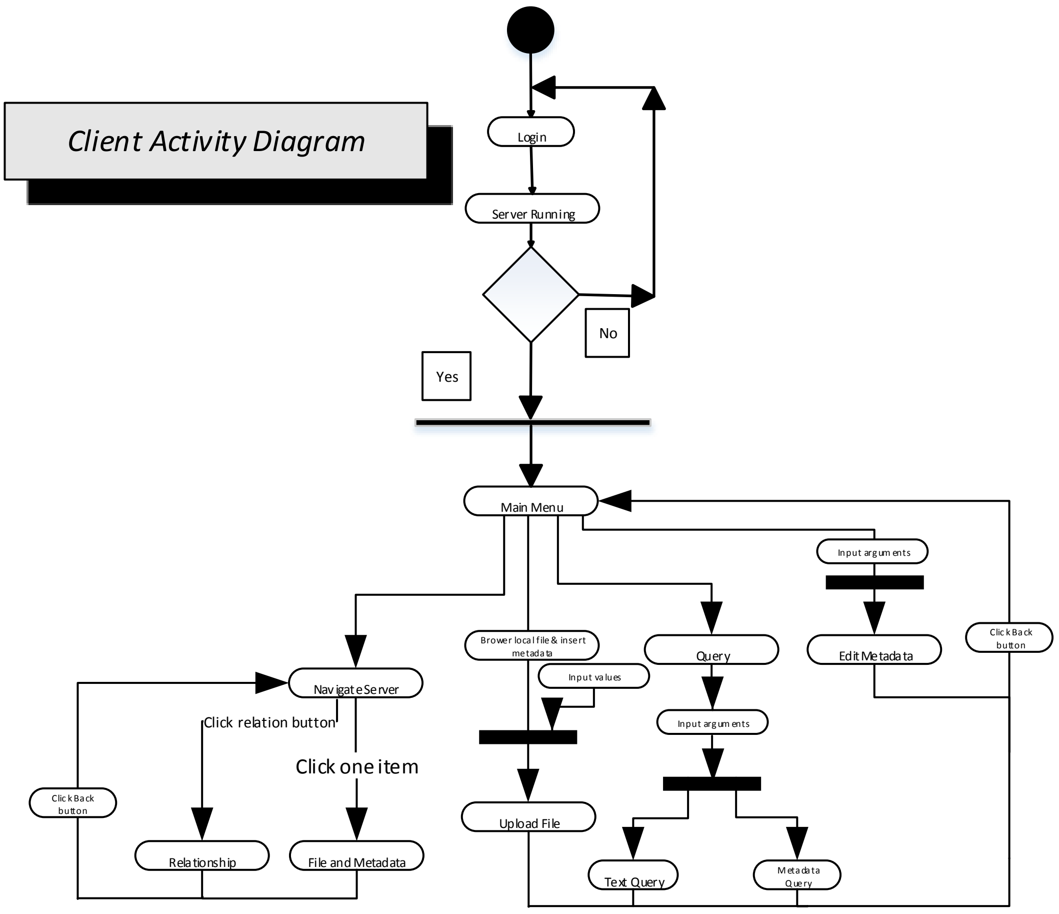
Clients:

1. Login in activity request (we don't implement this feature in this project)
2. Navigation activity request
3. File Extraction & Metadata activity request
4. Relationship activity request
5. Upload file activity request
6. Text query activity request
7. Metadata query activity request
8. Edit metadata activity request

Server:

1. Navigate root to get the categories
2. Navigate a category to show all the documents in that category
3. Open file and metadata to extract information to clients
4. Create relationship map
5. Save files and update relationship map
6. Generate a metadata file
7. Do text query and return the result
8. Do metadata query and return the result

Following is the client activity diagram and server diagram



Figure#2.1 Client Activity Diagram
* We don’t need to implement check-in service in this project

2.3.1 Navigate Server & extract content

When client click Navigate server on the main menu, the client will send a message tagged by “Navigation of server”. After the service of navigation on server, it will send back the navigation results shown on the navigation view of client side. Clients can check all files in a category by clicking one category anytime. And they can also select one file name as current file, click relationship button or extract button to see the relationship or content of that file.

2.3.2 Brower local file & insert metadata

Clients can upload a local file to server. After clients choose a local file, they also need to input the metadata value to help the server to automatically generate metadata for this file. When the clients push the upload button, the local file will be chucking up into blocks and then through WCF send to the server. A file can have multiple related categories and it need to have at least one category to match.

2.3.3 Text Query

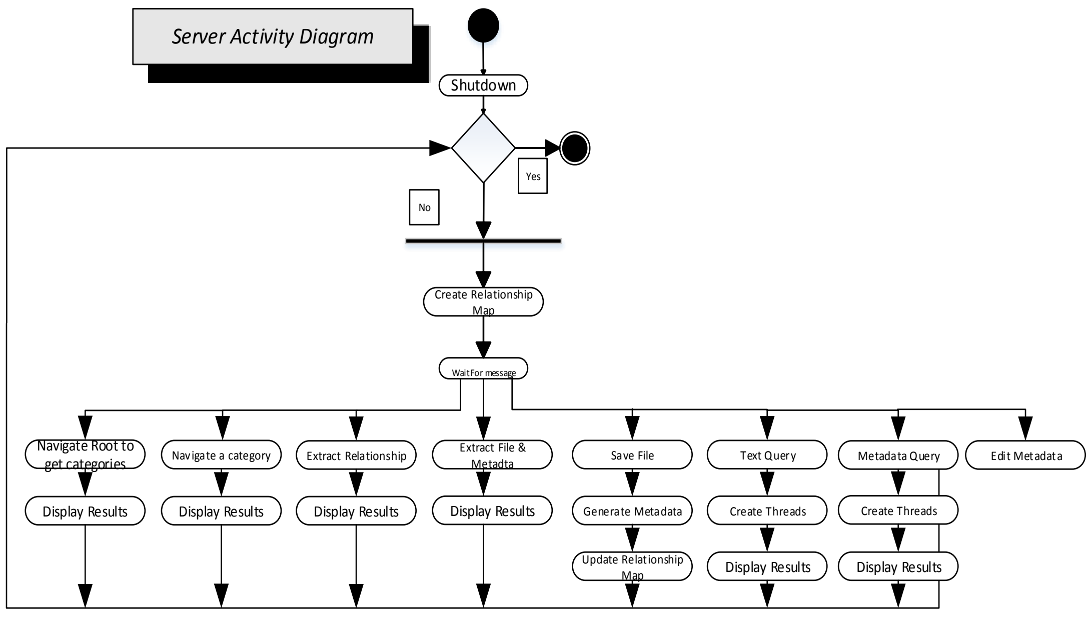
In the client side, the text query means the user will have a view for input and output of text query. The clients can have multiple strings to look for, can look for strings in multiple categories and can choose exact searching, which means that file should contain all matching strings or choose fuzzy searching, which means that file should contain at least one matching string. And then after the server finishes the query, it will send the finding results back to clients.

2.3.4 Metadata Query

In the client side, the metadata query means the user will have a view for inputting the name of searching tags, categories. When the clients push the query button, it will send a message tagged “Metadata Query” with all the values just input to the server. And then the server will utilize this information do the metadata query.

2.3.5 Edit Metadata

It is a view that the client can choose one of metadata file from the list box of metadata files on the server. The values needed to insert in the metadata part are just like the way when uploading file to server. But this time, the server is not to create a new metadata file, is to re-write an existed metadata file with current new value.



Figure#2.2 Server Activity Diagram

2.3.6 Create Relationship Map

Each time the server starts, the relationship tool will run. It will check the current relationship of each file and then store all its corresponding parent and children files name to a searching index xml file on the root of server.

2.3.7 Navigate

When the server get the message tagged with “Navigate root”, the service will use this package to extract categories names in the searching index file in the root of server. And send these categories information back to client. When the server get the message tagged with “Navigate category”, it will get the useful information from message, like category name, and then do the navigation on the server with that category name. When finishing the navigation, it returns the files set back to the clients.

2.3.8 Extract Relationship

When the server receives a message tagged “Extract relationship”, the relationship package will run and get the file name to be checked from the message. And then it can find the parent and children files name from the searching index xml file on the root of server. This searching index xml file is generated each time the server starts, and it will be updated each time when a file is inserted to the server.

2.3.9 Extract File & metadata

When the server gets a message tagged “Extract file & metadata”, the server will try to get file name of current file, the category name of that file and its related metadata file name. And then extraction service will be able to get the content of current file and current metadata file. Finally sending back to clients to display.

2.3.10 Save file and metadata

When the server receives the message tagged “Save File”, the server will check the existence of the category of that file. If the category doesn’t exist, it will create a new directory of that category name on the server. And then put the file blocks together and save the file on that directory. Then use the metadata tool to generate a metadata for that file. The value of metadata tags can be extracted from message. After that, it needs to update the relationship map, where all node of files needed to be checked, because the new file maybe the new children or new parent files of an existed file.

2.3.11 Text Query

When the server receives the message tagged “Text Query”, the Text Query will use the searching strings, categories and the exact matching or fuzzy searching option from the client side. If the text query needs to search string in multiple categories, the server can create multiple threads to handle text query in different category concurrently.

2.3.12 Metadata Query

When the server receives the message tagged “Metadata Query”, the Metadata Query will use the searching tags, categories from the client side. If the metadata query needs to search tags in multiple categories, the server can create multiple threads to handle text query in different category concurrently.

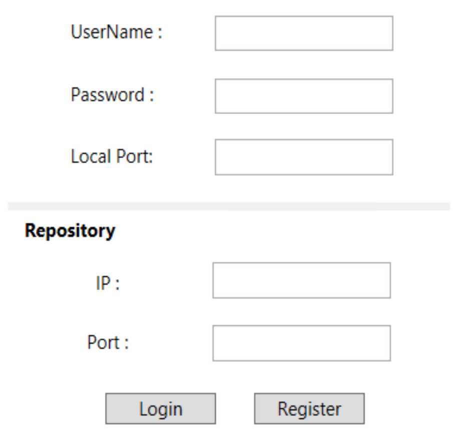
2.3.13 Edit Metadata

When the server gets the message tagged “Edit Metadata”, the Metadata tool in the server side will utilize the value of tags, the metadata file name and the category name to update a metadata file in that category folder.

2.4 User Interface views

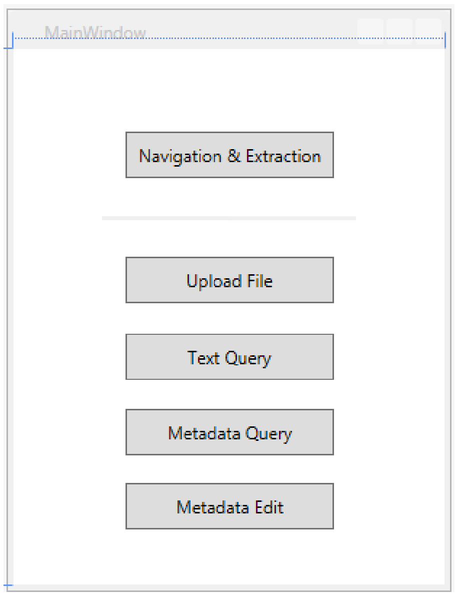
This section describes the graphical interfaces which the clients need to use to communicate with server. These interface will be straightforward and simple for clients to realize their requirements fast. The clients don’t need to the exact the arguments to talk with the server side, the exact function calls on the server side and what kind of data structure have been used on the remote server. What they need to care about is the correct results shown on the client view. Following are screenshots of possible interactions a client may need to use and the description of each view.

Screenshot #1



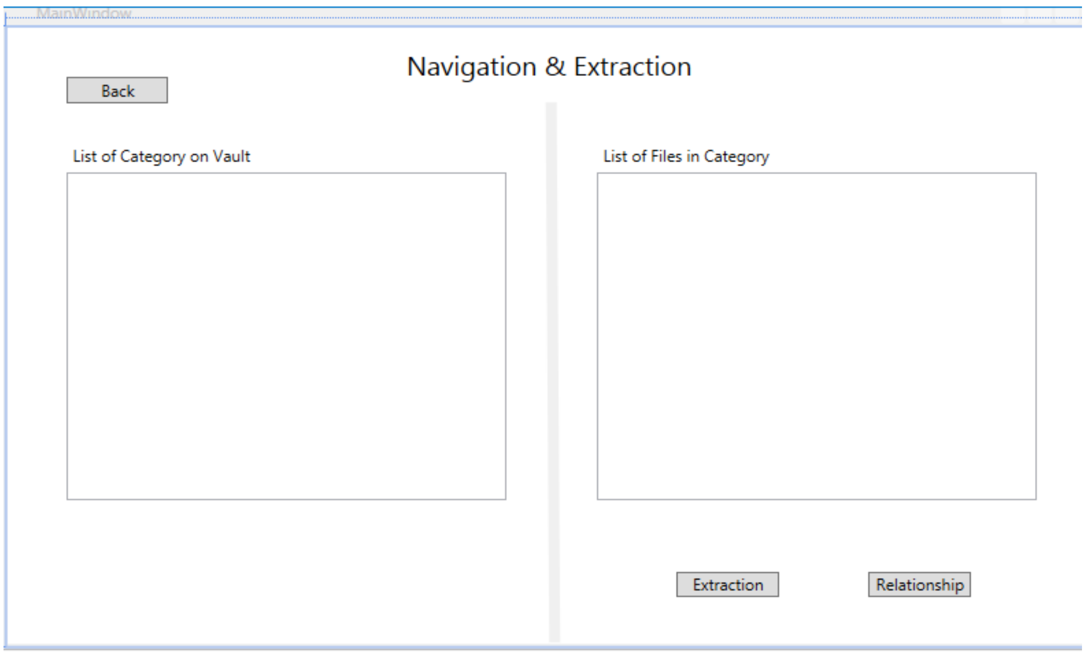
This is the prototype of a login in and register system for a complete product. Due to the available time of project, we don't implement this service.

Screenshot #2



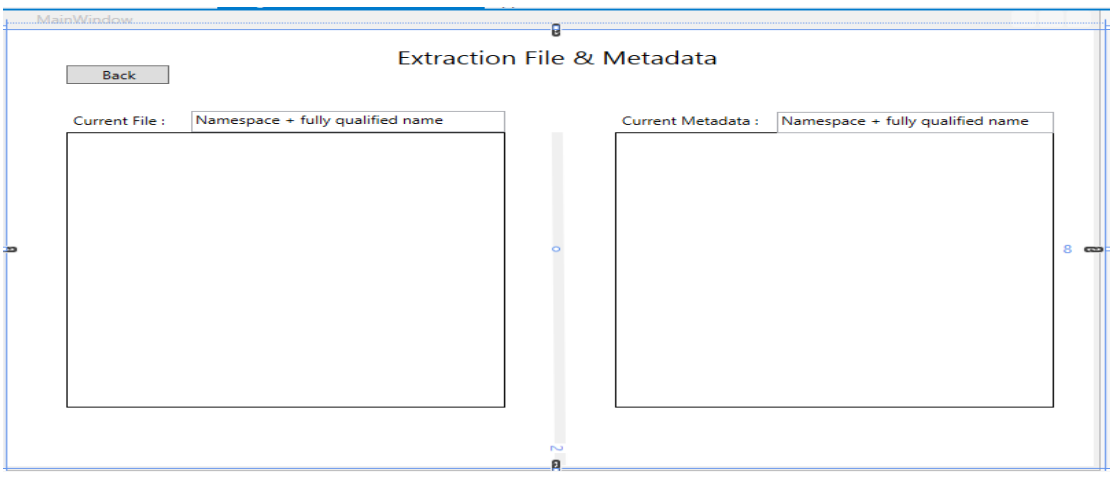
Ignore the first screenshot of the login view. This main menu should be the very start of the application. The clients have several options to choose. When the client click one activity of them, it will send a message to the server. And then the corresponding service on the server will start to run. Except the Upload File activity, other will communicate between client side and server back and forth. It means they can do immediate action from client to server, and then server will pass back the result of service to client.

Screenshot #3



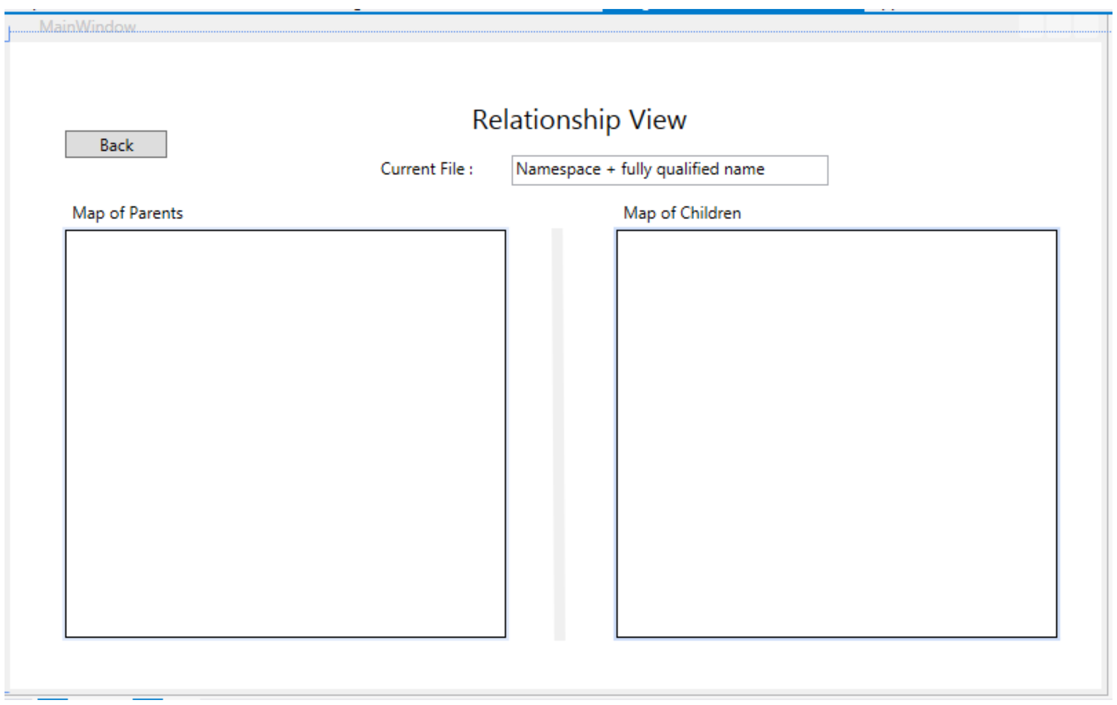
This view is used for navigating the files on the server. When this service has been chosen, the server will grab the categories name from the root searching index xml file. And then show the full path back to the client. When the client clicks one of these categories, another service will be trigger, navigate files of a categories. So the right view will shows the result of navigating files on a category. The client can also choose the item file on the right files list. And he or she can choose the extraction activity or relationship activity, both of them have the corresponding view for this functions. The back button is used for going back to the main menu.

Screenshot #4


MainWindow
Extraction File & Metadata
Back
Current File :
Namespace + fully qualified name
Current Metadata :
Namespace + fully qualified name

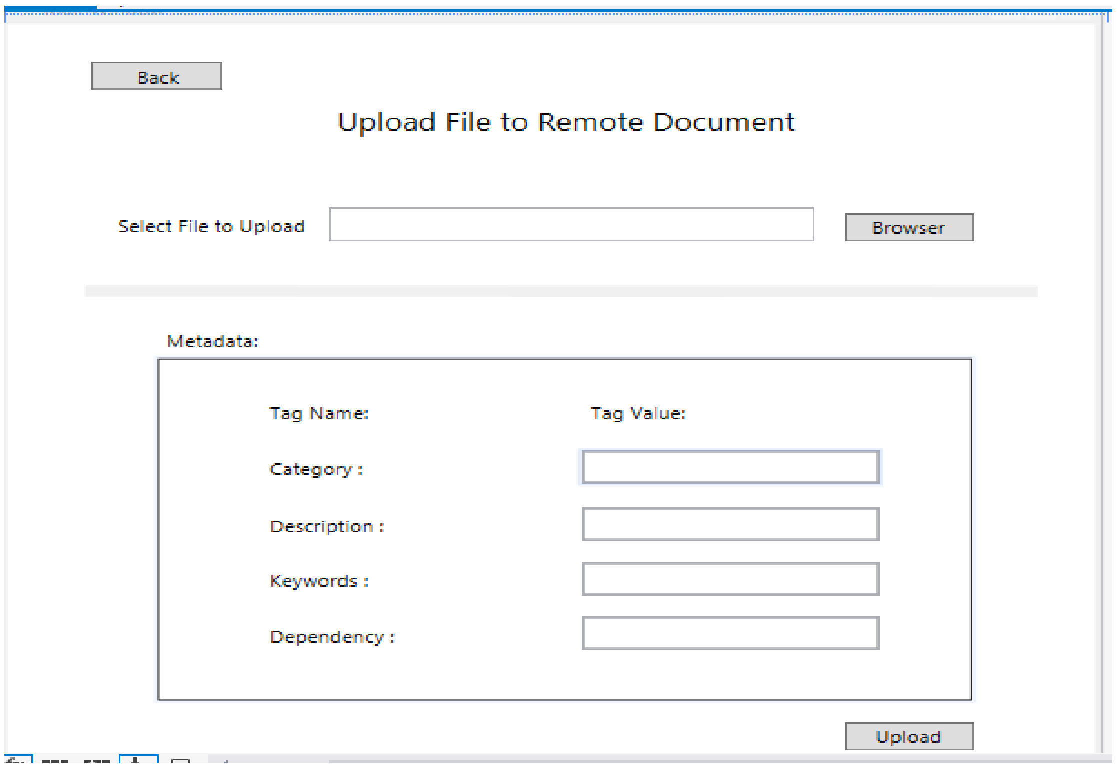
This view is used for showing the extraction information of current chosen file. The left part of view is showing the content of current file and the name of current file. And the right view will show the name of metadata file and the content of this metadata file. The back button on the left-top side is used for going back to navigation view.

Screenshot #5



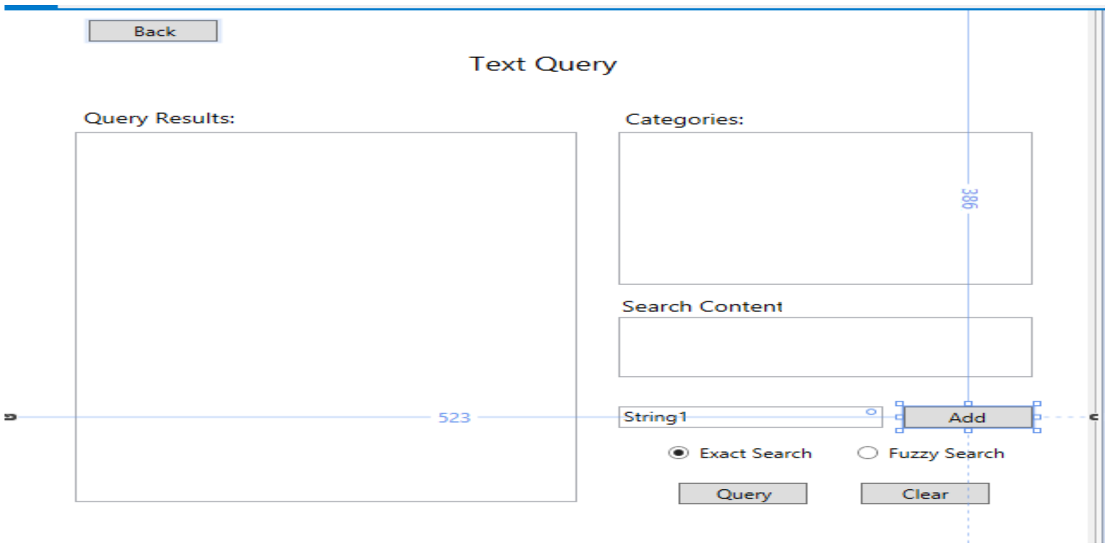
This view is showing the relationship of the current chosen file. The left part of this view is used for showing the parents of the current file. And the right part will show the children files of current file. All these service will be triggered and return result when the client chooses a current file and clicks the relationship button on the previous navigation view. The back button on the left-top side is used for going back to navigation view.

Screenshot #6



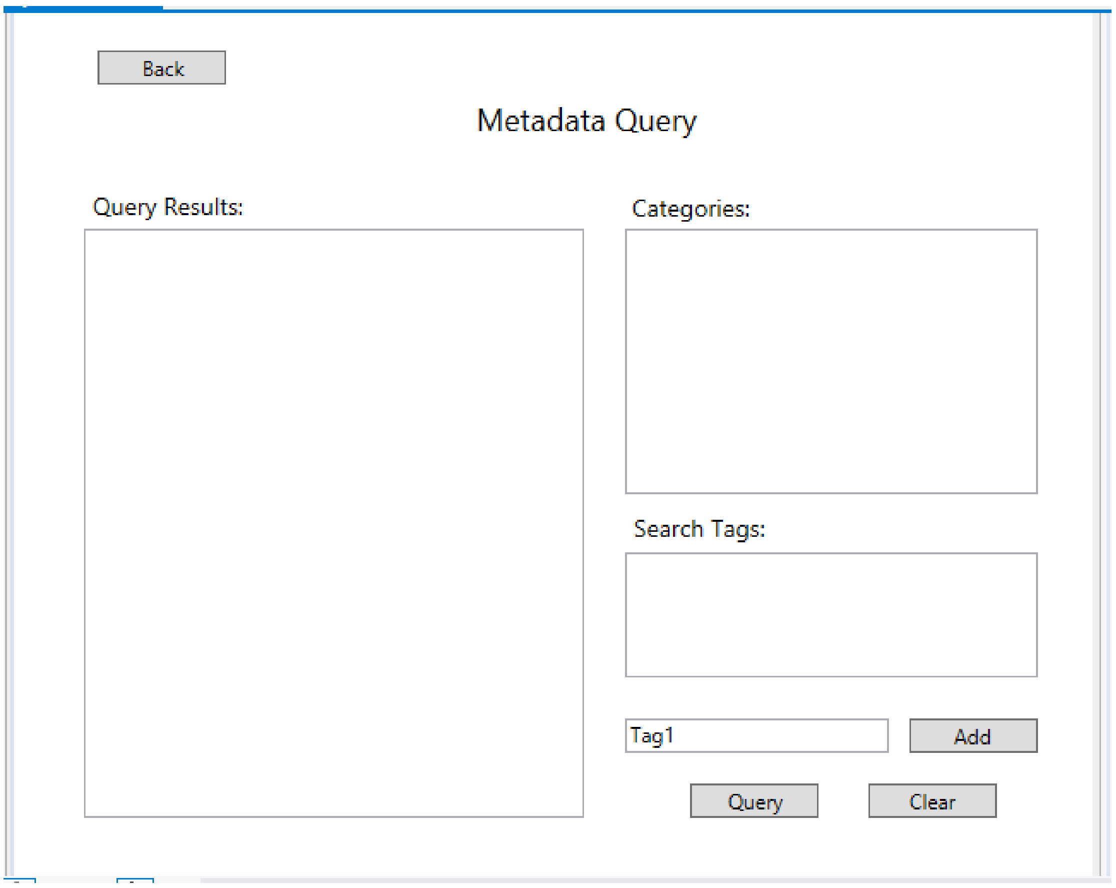
This is the view for clients to insert file to RDV. The clients can use the browser button to choose one of file on the local side, and the file should be the document file to be sent. On the bottom of the view is the metadata part, which could be used for descript this file, clients should write at least one category on the value text block. Other tags are optional. When the client pushes the upload button, the file chosen and the metadata value have been input will be sent to the corresponding function call on the server. The back button on the left-top side is used for going back to navigation view.

Screenshot #7



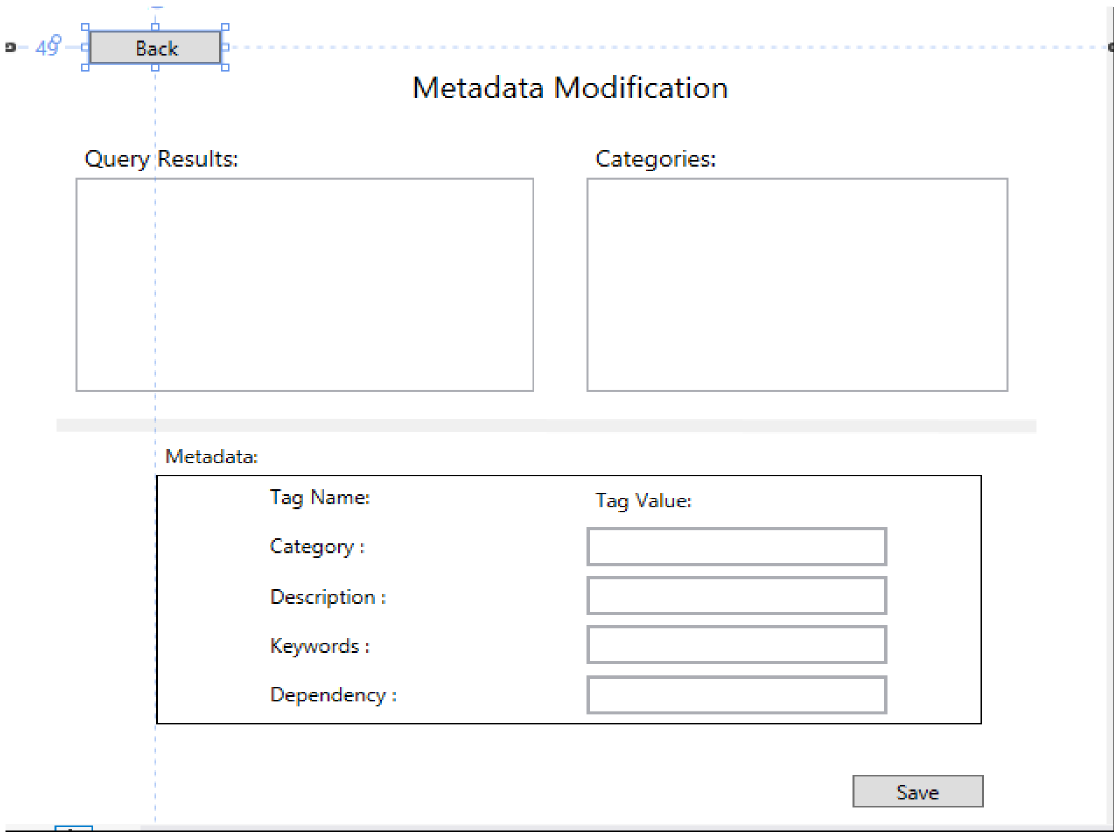
This view is used for client to do text query. This view will be popped up when the client chooses Text Query on the main menu. Here the client can choose multiple categories on the categories list. And the client can also add multiple searching string on the following text block. By push the add button, the current string will be added on the above search content list. Finally the client can choose the exact search or fuzzy search based on the searching string on the search content list. When the server finishes the Text query on the server, the finding result will be shown on the left side of client view. Finally, the back button on the left-top side is used for going back to main menu view.

Screenshot #8



This view is used for client to do Metadata query. This view will be popped up when the client chooses Metadata Query on the main menu. Here the client can choose multiple categories on the categories list. And the client can also add multiple searching tags on the following text block. By pushing the add button, the current string tag will be added on the above searching tags list. When client pushes the query button, the query message will be sent to server when server finishes the metadata query, the finding result will be shown on the left side of client view. Finally, the back button on the left-top side is used for going back to main menu view.

Screenshot #9



This view is used for client to do Metadata modification. This view will be popped up when the client chooses Edit Metadata file on the main menu. Here the clients can choose one of metadata from the Query Results. And the Query Results are the metadata files in the category which has been chosen in the right categories list. The bottom are the values of this metadata, here we need to input at least one category value, and all these values will be rewritten to the previous one. Finally, the back button on the left-top side is used for going back to main menu view.

2.5 Summary

The top-level architecture of the Remote Document Vault is composed of two main parts. One is client side, which is designed for nice and friend User Interfaces. And these UIs should be efficient to talk with the server and fulfill all the requirements. The other side is the RDV server. It is an application running on the server. It runs the core functions to handle the different activities, which are tagged in the message sent from the client. After finishing service, it sends the results back to the clients. The following is the main part of each module's function on both client and server side and how they can get interact with each other.

3 Details Design

When having a high-level understanding of RDV 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 bottom up coding modules to realize the whole application gradually.

3.1 Critical Processing Modules and their Interactions

RDV client side consists modules:

1. Client Executive Module
2. Display Module
3. Message Module
4. Main Menu Module
5. Navigation Client Module
6. Text Query Client Module
7. Metadata Query Client Module
8. EditMetadata Client Module
9. FileClient Module
10. FileMgr Module
11. Relationship Client Module
12. CommClient Module

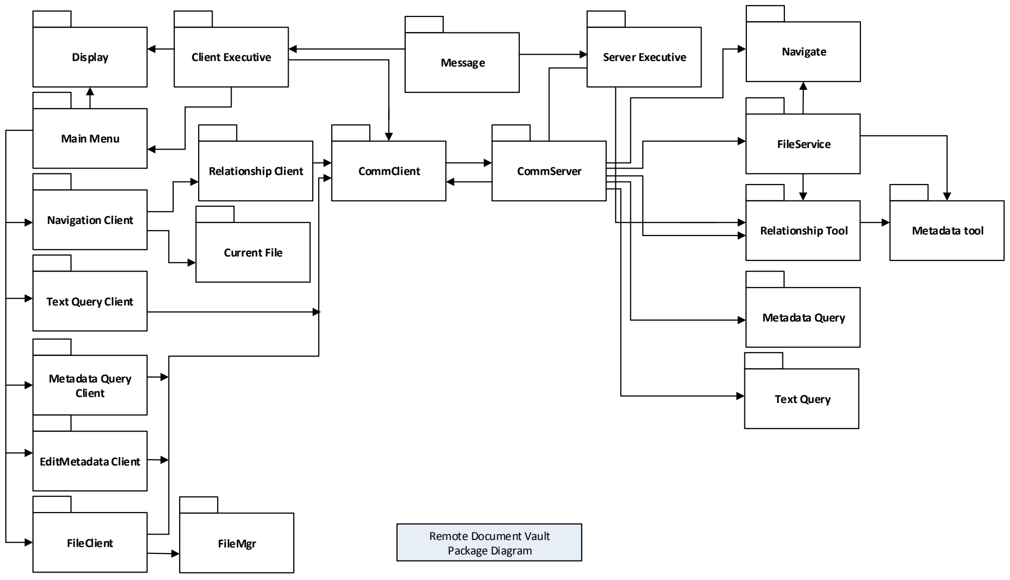
RDV server side consists modules:

1. Server Executive Module
2. NavigateModule
3. Message Module
4. FileService Module
5. Relationship ToolModule
6. Metadata toolModule
7. Metadata QueryModule
8. Text QueryModule
9. CommServer Module

As shown in the module diagram below, Executive Modules are the main application on both client side and server side. The most packages on the client side are User Interfaces. They are in charge of sending useful information to server to do corresponding activities. However, in the server side, each package contains core functions of each service and they do the functions according to the arguments sent from clients.

As for lower level modules like XML Reader Package, XML Proc package which was wrapped in the Metadata tool and FileMgr package. And 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 Remote Document Vault

3.1.1 Client Executive

This is the beginning of the client application. When the client starts to run, it will generate an object of CommClient, which is the main part for client to talk with server. And it also let the application set the current window as main menu window for user to choose the function.

It has the following classes:

- MainWindow: control the current window's view
- CommClient: user for creating a channel with server and do the communication with server.

3.1.2 Main Menu

This is the package designed the UI of Main Menu, which controls the function option for clients. The UI details have been discussed in the previous UI parts.

3.1.3 Navigation Client

This is the package designed the UI of Navigation& extraction. It's transfer from Main Menu. And it has a button to go back to the main menu view. When this view is starting, it will send a message to the server to get the whole category list which will be shown on this view. Besides, when click one of categories, another message will be sent to the server to get the whole files name of that categories. And the result will be shown on this view too.
It has the following classes:

- MainWindow: control the current window's view
- Message: include the message need to send to the server
- CommClient: user for creating a channel with server and do the communication with server.

3.1.4 Relationship Client

This is the package designed the UI of Relationship. It's transfer from Navigation& extraction. And it has a button to go back to the Navigation& extraction view. When this view is starting, it will send a message to the server to get the current file's parent and children relationship. And the return result will be shown on this UI.

It has the following classes:

- MainWindow: control the current window's view
- Message: include the message need to send to the server
- CommClient: user for creating a channel with server and do the communication with server.

3.1.5 Current File

This is the package designed the UI of current file. It's transfer from Navigation& extraction. And it has a button to go back to the Navigation& extraction view. When a file is selected, it will send a message includes the name of that file, and its corresponding category and metadata to the server. When server finishes the service, it will get the content of that file and its metadata back to the client.
It has the following classes:

- MainWindow: control the current window's view

- Message: include the message need to send to the server
- CommClient: user for creating a channel with server and do the communication with server.

3.1.6 Text Query Client

This is the package designed the UI of Text Query. It's transfer from main menu. And it has a button to go back to the main menu view. After the client inputs all information of query, it will send a message to the server to do the text query on the server side. When server finishes the service, the querying result will be sent back to the client and show on this UI.

It has the following classes:

- MainWindow: control the current window's view
- Message: include the message need to send to the server
- CommClient: user for creating a channel with server and do the communication with server.

3.1.7 Metadata Query Client

This is the package designed the UI of Metadata Query. It's transfer from main menu. And it has a button to go back to the main menu view. After the client inputs all information of query, it will send a message to the server to do the Metadata query on the server side. When server finishes the service, the querying result will be sent back to the client and show on this UI.

It has the following classes:

- MainWindow: control the current window's view
- Message: include the message need to send to the server
- CommClient: user for creating a channel with server and do the communication with server.

3.1.8 Edit Metadata Client

This is the package designed the UI of Editing Metadata. It's transfer from main menu. And it has a button to go back to the main menu view. After the client inputs all information of editing metadata, it will send a message to the server to rewrite a metadata file.

It has the following classes:

- Message: include the message need to send to the server
- CommClient: user for creating a channel with server and do the communication with server.

3.1.9 File Client

This package has views of input and core function to browser a local file behind the view. It has the core function to send file, inside the function, it can separate the file into blocks, which could be efficient to send to the server. The basic prototype of this service “File Client” is provided by PhD. Fawcett.

3.1.10 CommClient

This package is used for creating channel with the server side. It needs to use BasicHttpService to connect with server. The basic prototype of this service is provided by PhD. Fawcett.

3.1.11 Message

This is a utility package. In this package, we define a message class and enumeration of tag’s names. This package will be used for transferring information from client side and the server side.

3.2.1 Server Executive

This is the beginning of the server application. When the server starts to run, it will generate an object of CommServer, which will create a channel with client and it is the main part for server to talk with client. At the same time, an object of Relationship tool will be generated, which is control initialization of all documents relationship.

It has the following classes:

- Relationship Tool: Initialize the current relationship of all documents
- CommServer: user for creating a channel with server and do the communication with server.

3.2.2 Navigation tool

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, here when we get the name of category, we can do the category navigation, to get all files in that category.

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.

- CommServer: user for creating a channel with server and do the communication with server.

3.2.3 Metadata tool

This module will get the category name and current file name, and the list of metadata information needed to be added. When the metadata is not existed, the service will create a new metadata file for this file. If the metadata already in this category, the previous metadata file will be rewritten. 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.
- CommServer: user for creating a channel with server and do the communication with server.

3.2.4 Text Query module

This module is already implemented in the project2. And it's separate and reusable. In this project, the Text Query module will handle the same situation, by getting the category name, strings, exact searching strings or fuzzy searching strings.

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.
- CommServer: user for creating a channel with server and do the communication with server.

3.2.5 Metadata Query module

This module is also implemented in the project2. And it's separate and reusable. In this project, the main function is provided for searching the metadata contents in the categories. It accept at least one tag argument, 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.

3.2.6 File Service module

This service is used for saving the file in the category the client provided. It has several main functions, like OpenFileForWrite, WriteBlockFile, Close and so on. And this module prototype is already implemented in “FileService” package, which is provided by Phd. Fawcett.

It has the following classes:

- Metadata Tool: Write the metadata file with the current file
- Relationship Tool: update the relationship when a new file is inserted in the RDV
- CommServer: user for creating a channel with server and do the communication with server.

3.2.7 Relationship Tool module

This Relationship tool is used for store the information of each file’s parents and children. It runs each time when the server starts, the information will be stored in a dictionary data structure and export to a searching xml file in the root. And when a file is inserted into the server, this tool will run again and the information of relationship of each files will be updated. It needs the metadata tool to read and write xml file.

It has the following classes:

- Metadata Tool: read the files’ corresponding metadata file and export the relation result to xml file
- CommServer: user for creating a channel with server and do the communication with server

3.2.8 CommServer

This package is used for creating channel with the client side. It needs to use BasicHttpService to connect with server. The basic prototype of this service is provided by PhD. Fawcett.

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. They can also code from UI to corresponding Service on the server each function by each function with this detail design of package diagram.

4 Critical Issues

4.1 Concurrency issue:

4.1.1 Concurrent uploading

Situation: When there are two or more clients connecting to the RDV, and they want to send two different files at the same time. If the RDV server uses the single thread to deal with transmitting files to the server and it very likely happens that one client submits a large file, other clients may get stuck on their side.

Solutions:
On the sever side, we use the multi-thread to handle different files' transmission for different clients. So in this way, these clients will be able to send their local file to the server. However, the speed of uploading would probably slow down. It depends on the binding and encoding way we choose creating WCF. With the multi-thread on the server side, we could have better performance on the both server side and client side. Also, it increases the throughput.

4.1.2 Unreadable File and File Conflict

Situations:
When RDV comes 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 system API 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 when some client wants to extract information of a specified file and its corresponding metadata, other clients may do writing or modifying to these files. This situation would probably happen when client sends the request to do file extraction, text query, and metadata query and metadata modification.

Solutions:
Before the RDV does service relating with file reading and writing for a client, it needs to check the file handler first each time before the program tries to read or write it. If

the current file cannot be access or is being used, it cannot process operation and do return a warning or error message to the client side. Then the application on server 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.

4.2 Managing file contention

Situations:
When the clients continue to upload the files into the RDV, the repository may be messed up if we don't have a good way to store the files on the server side. In that way, it could not be efficient to make querying or modification. One managing problem is that the RDV could accept to store same files or not. Another one is how to handle a file identified one or more user-defined categories.

Solutions:
1. In order to reasonably manage the RDV and make efficient to query and extract, we can create a directory based on the name of category when user uploads a file with its related metadata. Get the name of category in the metadata file and create a category directory at the first time on the server side, other times we just put the files on those category directory. In the meanwhile, create a whole searching index on the root and update it each time when a file is going to store on the RDV.

2. We don't allow the clients to upload the files with the same name in the RDV. If the RDV can store the same files, we do need more effort to store the absolute path of that file and it will create the problem when the service is going to search the relationship between parents and children. So in this project, the server will ban clients to upload the file with same name that already existed in the RDV. And the client will get warning message information, saying "the RDV already has the same file" on the client side when he tries to upload a file with same name in the server.

3. If a file has two or more categories identities (this situation is different with the previous one), firstly the server will store information about file name and belonged categories into searching index on the root directory. In this way, when client wants to check the RDV, the server just needs to extraction information from this searching index on the root, instead of recursively searching all category directories. And then save the file and its metadata file to its related category folders.

4.3 Security & Ownership policy

Situations:
Security issue, not everyone can login remote document vault. Because it is remote

document vault, which could be used for company security documents, personal information and so on. So it could should not be total open to those people who don't have the authentication. Otherwise, it is valuable to be attacked by improper person. Ownership issue, sometimes clients want to modify a file in the RDV that was not belong to him or her or let's say he have not right to handle others'. Besides, sometimes the owner wants to update its old file. Although we don't need to implement these features in the project 4, it's an import issue of a real tool for companies.

Solutions:
For security, the RDV needs to hold a registration system to keep information about the authenticated clients. Only authenticated person has right to check the files on the server. When an authenticated person submits a file and metadata to the RDV, it should come with the author tag and version of that file in the metadata file. Otherwise, when others want to refer to this file, it may not get the latest version of correct file. We can even keep the old files in the RDV and create a new file and metadata with new version in the RDV.

4.4 Working with many documents

Situations:
When clients continue to upload their files on the RDV, it might reach the max capacity of storage someday. And after that the server will crash and is not reliable for storing any file, which means files might get missing. Another situation is that, some files might be generate randomly by clients or some files are entirely useless and non – visited for a long time. It wastes the valuable capacity of the server.

Solutions:
One way to solve this problem is depend on the stuffs in maintain department, they need to regularly check the server of RDV and upgrade the server when it is necessary. Another way to manage this is that we set the timestamp for each file and store this information in its metadata. The server will run a regular service to check the unvisited files and then may achieve these files or notice stuffs of maintain department to fix it.

4.5 Tags for metadata

Situation:
The clients may want to use their own tag in the metadata file, but it will damage the consistence of RDV.

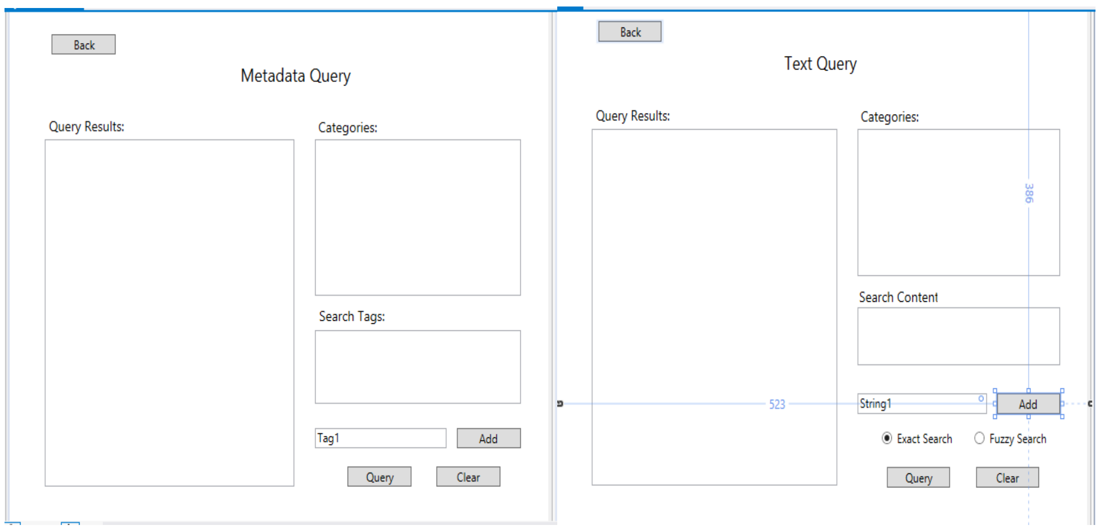
Solutions:
Instead of allowing clients to create their own tags, we provide more common, useful

and consistent tag for them. They just need to input the value of tag in the metadata part in the client side.

4.6 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”. So when the server receive these information, it cannot parse them and do the correct functions. This situation will happened when client tries to add query arguments or modify the metadata file.

Solution:
To solve this problem, we just need to provide good interactive UI interface for client to insert the value, and with a button to generate the correct argument for the service on the server.



4.7 File transfer with chunking

We can use streamed or buffered/chunked way to upload file in WCF. The transfer with chunking is the one that provides reliable data transfer as we can use wsHttpBinding with WS-ReliableMessaging turned on. Reliable messaging cannot be used with streaming as the WS-RM mechanism requires processing the data as a unity to apply checksums, etc.
Besides, processing a large file this way would require a huge buffer and thus a lot of available memory on both client and server; denial-of-service comes to mind. The workaround for this is chunking: splitting the file into e.g. 64KB fragments and using

reliable, ordered messaging to transfer the data.

4.8 Building flexible vault framework

This RDV should be flexible framework to store files for other projects like Project#5.
This RDV has very friendly UI interface by WPF for clients to do insertion and extraction of files and their metadata from a Repository server with WCF.
This RDV has a navigation UI for client to extraction information from the repository.
This RDV provides a separate tool to search metadata files and build a map of parent for each file in the repository.
This RDV provides a management view to support browser local file and upload the files into the repository.
This RDV support text and metadata query through argument input on the client side.

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

Prototype code provided by PhD. Fawcett
WCF Communication between clients and server:
Project: WCF_CommPrototype
ICommServeice.cs
CommService.cs
WSHttpClient.cs

Upload file from clients to server:
Project: FileService
IFileService.cs
FileService.cs
Clients.cs

Brower category and navigate on the server:
Project: WPF_DispatcherDemo

Clients get the result as string from server:
Project: SelfHostedStrings
IStrings.cs
Strings.cs
Clients.cs

Others:
From My Project 2
Following packages are for the server side of the RDV:
MetadataTool.cs
Text_Search.cs

Metadata_Search.cs
FileMgr.cs