

Remote Document Vault

Operational Concept Document

Rashmesh Radhakrishnan

SUID: 246911788

CIS 681- Software Modeling and Analysis

Project: 3 FALL 2013

10/21/2013

CONTENTS

1	INTRODUCTION	1
1.1	EXECUTIVE SUMMARY.....	1
1.2	REMOTE DOCUMENT VAULT FUNCTIONALITY/ REQUIREMENTS.....	2
1.3	USE CASE	3
1.4	USERS	5
1.5	CATEGORIES	6
2	VIEWS.....	7
2.1	VIEW CATEGORIES.....	7
2.2	FILE INFORMATION METADATA AND TEXT - FILE INFORMATION TAB.....	7
2.3	FILE INFORMATION METADATA AND TEXT - MAP ME TAB.....	8
2.4	FILE INFORMATION METADATA AND TEXT – CONNECTION FLOW	9
2.5	SEARCH TEXT BASED.....	9
2.6	SEARCH METADATA BASED.....	10
2.7	UPLOAD TO VAULT.....	10
2.8	UPDATE METADATA AND ADD CATEGORIES	12
3	STRUCTURE	14
3.1	HIGH LEVEL ARCHITECTURE.....	14
3.1.1	CLIENT.....	14
3.1.2	SERVER	16
3.1.3	RELATIONSHIP TOOL.....	18
4	OPERATIONS.....	19
4.1	SERVER STARTUP.....	19
4.2	FILE UPLOAD	19
4.3	TEXT SEARCH.....	21
4.4	METADATA SEARCH	21
4.5	POPULATE PARENT AND CHILDREN RELATIONSHIP.....	22

4.6	DISPLAY METADATA AND FILE CONTENT	23
4.7	VAULT METADATA EDIT AND CATEGORY CREATION.....	24
5	CRITICAL ISSUES.....	25
5.1	MANAGING FILE CONTENTION.....	25
5.2	FILE TRANSFER WITH CHUNKING	25
5.3	OWNERSHIP POLICY	26
5.4	WORKING WITH MULTIPLE DOCUMENTS	26
5.5	BUILDING FLEXIBLE VAULT FRAMEWORK THAT ACCOMMODATES SPECIALIZATION	27
5.6	ARCHIVING ISSUES	27
5.7	ADDITION OF NEW CATEGORIES.....	27
5.8	INSERTION OF THE FILE WITH CHILD DEPENDENCIES.....	27
6	CONCLUSION	28
7	REFERENCES	28

1 Introduction

1.1 EXECUTIVE SUMMARY

“Remote Document Vault” tool is a data repository that is a partial file based data management system. The major features revolve around insertion of files into the remote vault and queries on these files. Queries are classified into two parts a) based on the text contained in the file b) based on the MetaData of the file. Remote Document Vault (RDV) can be used to create a specific MetaData for each file which also maps a particular category and can be used to build a hierarchical relationship between the files.

The vault provides simple and informative view of the dependencies. It also helps the user to understand the various groups a file belongs to. Categories are used here for grouping the files. It is built on a client-server based approach and thus can be used on different devices by the same user and can also be used for document storage and retrieval.

It is intended to be used by:

- Content Providers
- Software Application Developers
- Software Architects
- Quality Analysts
- Code Review Team
- Project Manager
- Customers
- Students

The critical issues involved are listed below which will be discussed at the end of the document

- Managing file contention
- File Transfer with chunking
- Ownership policy
- Working with multiple documents
- Building flexible Vault framework that accommodates specialization
- Archiving issues
- Addition of New Categories
- Insertion of the file with child dependencies

1.2 REMOTE DOCUMENT VAULT FUNCTIONALITY/ REQUIREMENTS

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. Child dependency relationships and file properties (including categories) are entered by the user when a MetaData file is created. The tool sequences through each MetaData file and for every "child", it records the current file as a "parent" in its child's MetaData. This information is stored in a map used for navigation.

The client side application should be able to upload a file to the server and read the file from the server. The server should be able to create a MetaData file on every insertion of a file into the vault. A separate MetaData tool exists which provides a mapping of current files in the vault. The client view consists of a) Category list and the file view pane - corresponding to each selected category b) Parents and children related to selected text file c) MetaData file info and the content of the file for the selected file. The views (b) and (c) get updated when the current selected file is changed. d) A view to upload and edit the MetaData values generated for a local file.

RDV should be able search concurrently across files by accepting a) one or more categories, b) one or more text strings to seek and c) a switch setting indicating whether any or all query strings are required to be matched. The result gives a list box of files. On selecting any file from here it updates all the views considering this file as the current file. It accepts MetaData query messages specifying one or more categories and MetaData tags and returns the content of each element of the MetaData for every file in the file set. A MetaData query will always return its file reference and categories which are stored as text element in the view. A view also exists for editing any existing MetaData file.

1.3 USE CASE

To understand the complete outline the system can be considered to be divided into three major users.

1. Thin client user
2. Administrator
3. Server System

There are various uses that can be seen with respect to each User. These can be used to understand the system better:

1. Thin client user

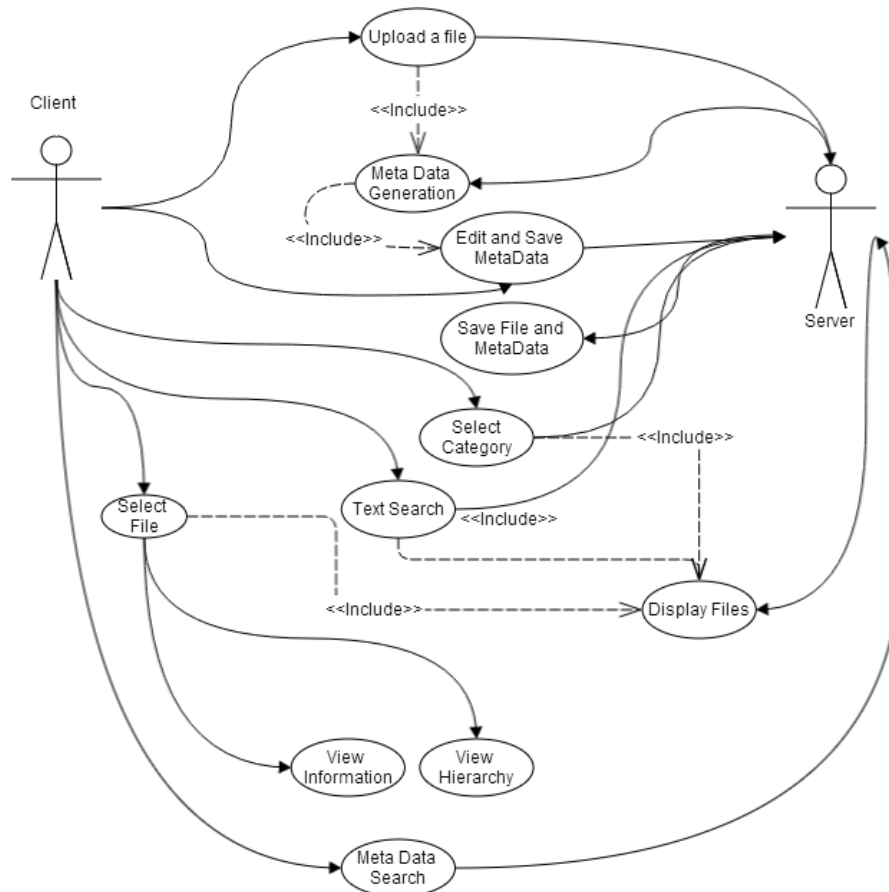


Fig 1.1: Use case diagram for the Thin Client User

- Upload:** The client uploads a text file into the server. This can be done by selecting a local file from the client's disk space.
- MetaData Generation:** The uploaded file is read by the server and the server returns certain attributes of the MetaData information (Size, Last Modified etc.).
- Edit and Save MetaData:** The user can edit the generated MetaData to his need and he can decide to complete the upload by click on save or restart the process by a cancel action.
- Select Category:** The client gets to select one more categories from a populated list of categories. This triggers a roundtrip to the server where it requests for all the files that belong to the requested category/ies if the local cache doesn't contain the category list.
- Text Search:** It accepts text query and requests for the fully qualified name of every file containing that text from the server.

- f) **Display Files:** In the case of category search or a text search the server returns all the files that can be used for further processing. This acts as an input for the use case where the user selects the current file.
- g) **Select Current File:** Once the result of files is displayed, user has an option to select one of these files by double clicking the file from the list view. Once the file is changed then there are two more views that get changed simultaneously.
 - a. **File Information:** This view is used to display the MetaData and the contents of the file (if the user has permission to view the file). The user can see this by selecting the file information tab.
 - b. **Hierarchy:** As per the requirement the view displays the parents and children associated with the selected file. There could also be additional provision to illustrate a pictorial tree which helps explore the entire hierarchy keeping the selected file as the starting point of search.
- h) **MetaData Search:** accepts MetaData query messages specifying one or more categories and MetaData tags and returns the contents of each element of the MetaData for every file in the file set. A MetaData query will always return its file reference and categories.

2. Administrator

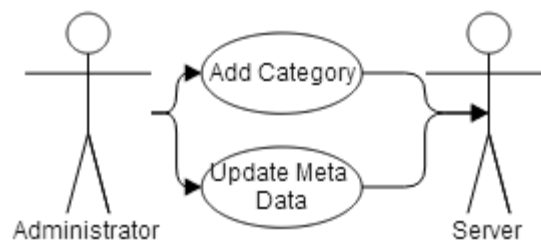


Fig 1.2: Use case diagram for the Administrator

- a) **Add Category:** The various Categories need to be decided by a user who knows about the systems use overall. For example, in the case of a software repository, an architect would be the person who would be able to tell which categories are required and which are not.
- b) **Update MetaData:** Supports editing MetaData files by supplying a set of tag names and values. If the tag exists in the MetaData file being edited then the new 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 the new value.

3. Server System:

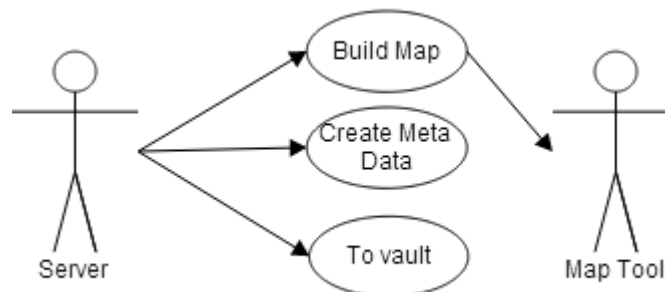


Fig 1.3: Use case diagram for the Server System

- c) **Build Map:** It invokes a separate tool which builds the map with the help of the MetaData files. This tool returns a graph based result (adjacency list) which basically is used for building the hierarchy.
- d) **Create MetaData:** Once the file is uploaded with the MetaData attributes, the server must make a unique copy of the text file and one to one mapping based MetaData file.

- e) **To Vault:** This functions checks for possible issues in relation with the file and then makes a unique file name based copy and saves it on the server's disk drive.

1.4 USERS

- **STUDENTS**

This vault can be used to store books (text files) based on categories. If he/she needs to search for a particular category, say "Mathematics". He/she can search for keywords like "graphs", "game" and get a result of files that contain these words. With the relevant relationship, a Student can decide which files he/she must read in order to go ahead with the routine. With the MetaData search of the category files we can also try to find a hierarchical approach to cover a subject.

- **CONTENT PROVIDERS**

Content providers are firms which provide text, graphics of articles based on interviews, developments, stories etc. that can be used to make a site or publication useful for readers. They always deal with large set of articles. This tool will definitely be useful to content providers to search for required file using the search by text functionality. Users can enter text which is required to be searched, file set, file patterns and they will be provided with list of files along with full path which has the occurrence of text in it. It can also be used to draw a graphical pattern among the files and then predict next stories or find similarities between stories.

- **SOFTWARE APPLICATION DEVELOPERS**

Developers in a software project work with large number of source code files. This tool can be used by developers to create MetaData for the source code files. The MetaData file has version number which will help developers for versioning of the files. Using specific keyword/text they can search the required file from large working set. This tool will provide developers details about the dependencies on file. They can fetch the all the required files and the files they are dependent on to form a project and start working on it. The source code can have a hierarchy and can be used to identify dependent files and isolate and later modularize the code files required for which functional unit.

- **SOFTWARE ARCHITECTS**

Software architects who work on multiple projects can use this to create and version their files. They can login from different systems and also use the vault as a data repository. They can store each dependent unit as a category. Each file that depends on a category can be traced. The coupling and cohesion of each file can be studied with the help of its hierarchical structure. This can also be used to make changes to a file based on how dense a file dependency is. If the file is found to have many dependent files, what all would be effected by making a change can be found clearly.

- **QUALITY ANALYSTS**

QA can use this tool to get various versions of the file that have been changed. They can compare two versions and see how the files were modified. In many years of use, the tools can become a large repository and with a bit of data intelligence of the Meta Tag search and text search can be used for Data Mining.

- **PROGRAM REVIEW TEAM**

Program review teams are generally involved in reviewing the code of the project against set of metrics which are followed in a product development industry. Software project consist of thousands of code files, this tool will help code review teams to get the file which needs to be utilised based on the hierarchical map and the text based search results. They can also use MetaData search functionality to get details of tags which belong to a particular category. A new category can be built for a completely new functionality based research. All people can collaborate and the vault can act as a central repository for individual modules.

- **PROJECT MANAGER**

Project Manager can use MetaData search functionality to get the list of versions of a project management file and then download required version. Manager will also be able to get the details about the user who created a file. He/ She can also track the file versions using the MetaData search functionality. File information can be seen graphically and can help track the progress and dependencies based on the file connections.

- **CUSTOMERS**

Customers can use this file repository online to store all their old and new transaction files online. This can be plugged with an online analytical tool where one can perform search based on all the versions and the related files. The dependencies generated from the RDV can help them sort and eliminate or prioritize needs based on dense dependencies. Files can be searched based on categories. MetaData relevant to files of given set of categories can help in understanding if there is a particular pattern followed. These can be retrieved and used at a later point of time. The older files can be used to perform a similar business transaction with a new company based on the ideas gathered from similar category based files.

1.5 CATEGORIES

A specifically defined division in a system of classification; a class is what a category is. Categories are intended to group together pages on similar subjects. Categories help the users to find, and navigate around, a subject area, to see pages sorted by title, and to thus find article relationships. The term category does refer to both the title of a category page—the category name—and the category itself.

Categorization is the process in which ideas and objects are recognized, differentiated, and understood. Categorization implies that objects are grouped into categories, usually for some specific purpose. Ideally, a category illuminates a relationship between the subjects and objects of knowledge. Categorization is fundamental in language, prediction, inference, decision making and in all kinds of environmental interaction. It is indicated that categorization plays a major role in computer programming.

There are many categorization theories and techniques. In a broader historical view, however, three general approaches to categorization may be identified:

- Classical categorization : grouping objects based on their similar properties
 - According to the classical view, categories should be clearly defined, mutually exclusive and collectively exhaustive. This way, any entity of the given classification universe belongs unequivocally to one, and only one, of the proposed categories.
- Conceptual clustering : attempts to explain how knowledge is represented
 - The task of clustering involves recognizing inherent structure in a data set and grouping objects together by similarity into classes. It is thus a process of generating a classification structure.
- Prototype theory : process of grouping things based on prototypes
 - Categories form part of a hierarchical structure when applied to such subjects as taxonomy in biological classification: higher level: life-form level, middle level: generic or genus level, and lower level: the species level.

In this document we do not mention how the categories are being created. The software gives this discretion to the user of the system to choose. So further along the document description we generally refer to category as a system of classification.

2 Views

The remote document vault implements the Graphical User Interface in C# using the facilities of the .Net Framework Class Library (FCL), version 4.5, as provided in the EECS clusters. Use of GUI is demonstrated, and will be implemented using Windows Presentation Foundation (WPF).

2.1 View Categories

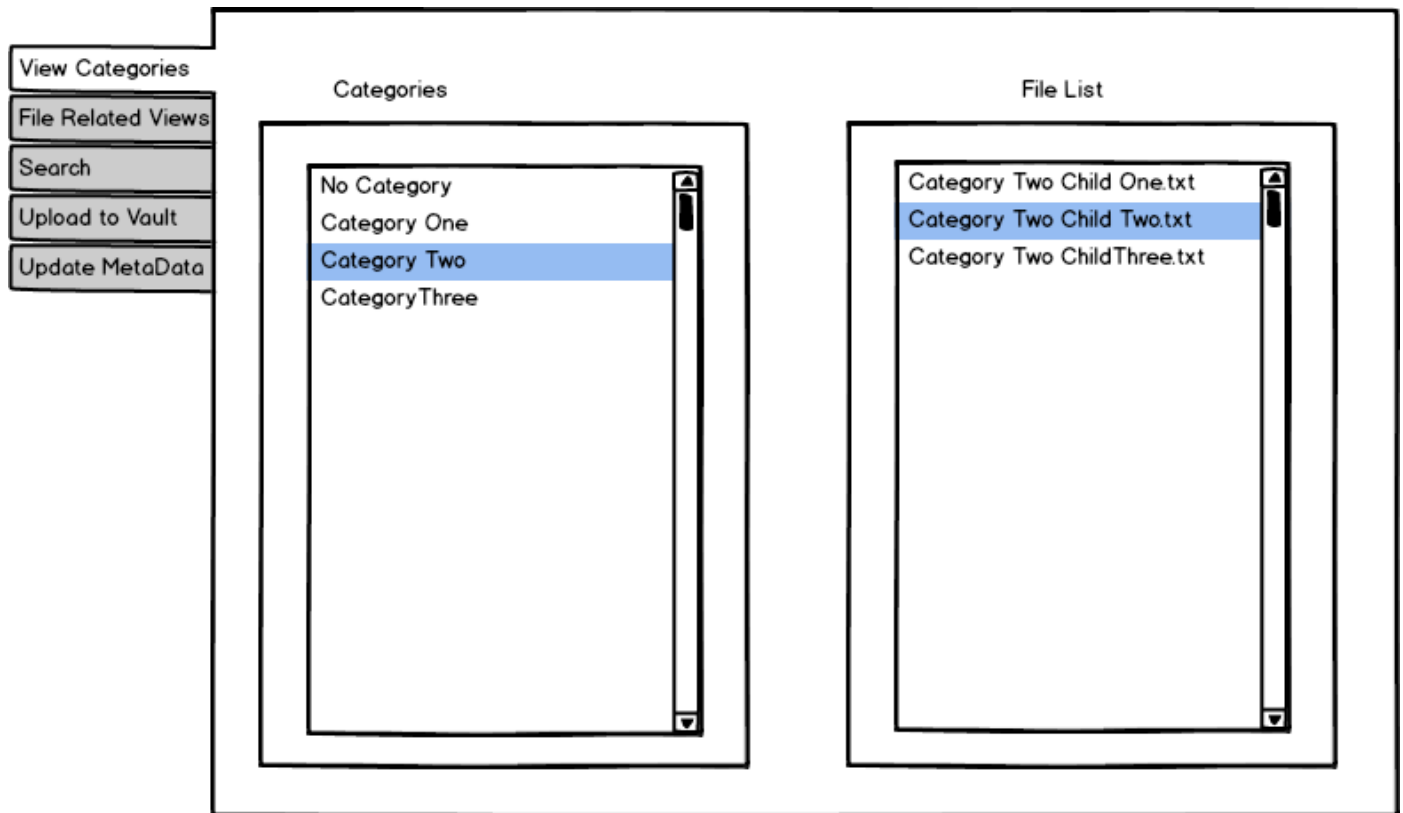


Fig 2.1: GUI for view categories

This Vault Client GUI supplies a category view that displays all the categories in the Vault. Clicking on a category displays all the files in that category. Clicking on one of the category files makes that file the current file, file related views (explained later) get updated treating the selected file as current file. The Vault client GUI tab for view categories can be accessed at any time. The MetaData for each file that has no file parents shall identifies its categories as parents.

2.2 File Information MetaData and Text - File Information Tab

The Vault Client is a WPF application that provides an information view with two information panes, one for MetaData and one for file text of the currently selected file. This view has a file name and the two panes as shown below. This view gets updated every time the current file is changed. The file content is shown based on permissions and we assume this check is taken care by the operating system and the system is not required to look into these details.

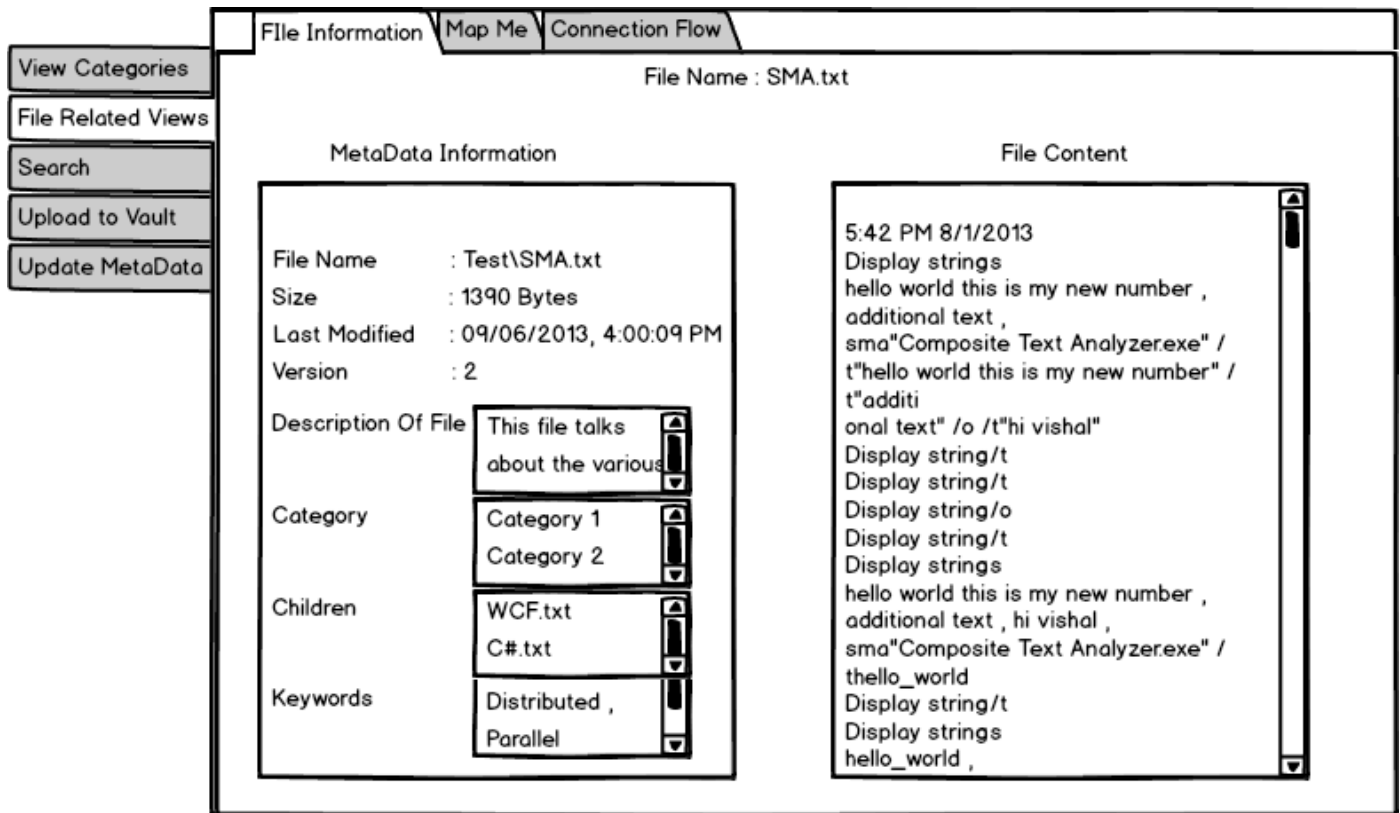


Fig 2.2: GUI for File Information

2.3 File Information MetaData and Text - Map Me Tab

The client GUI also provides two navigation panes, one showing the current file's children and one showing its parents. Clicking on either a child or parent changes the current file to the file represented by the clicked link. Tool tips are provided for clarity of operations.

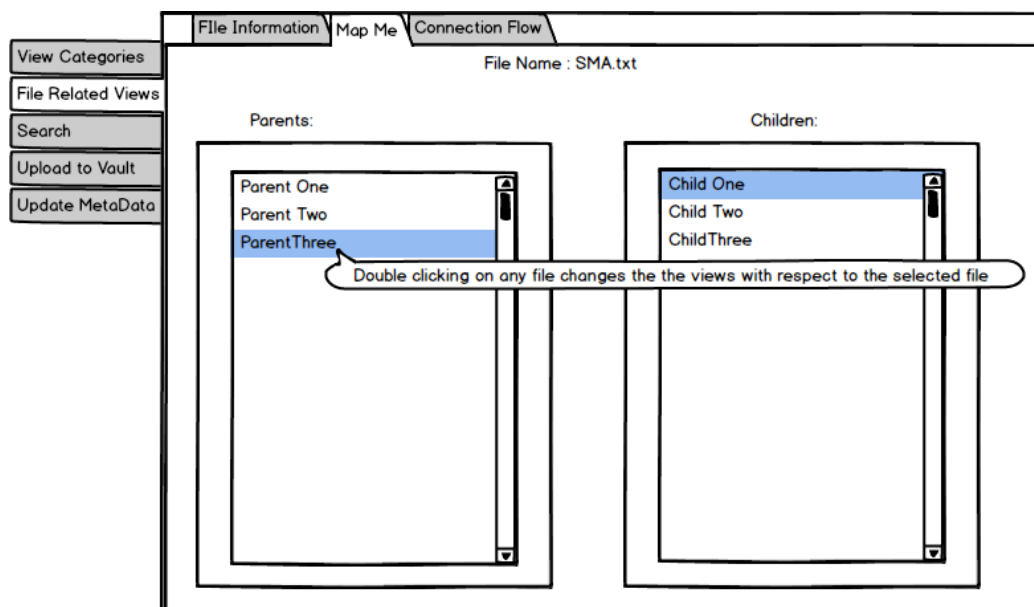


Fig 2.3: GUI for Map Me

2.4 File Information MetaData and Text – Connection Flow

The client GUI also provides an additional feature which maps the parent and children of the current selected file. Clicking on either a child or parent changes the current file to the file represented by the clicked node. This can be used to understand the hierarchy of a given file in the vault.

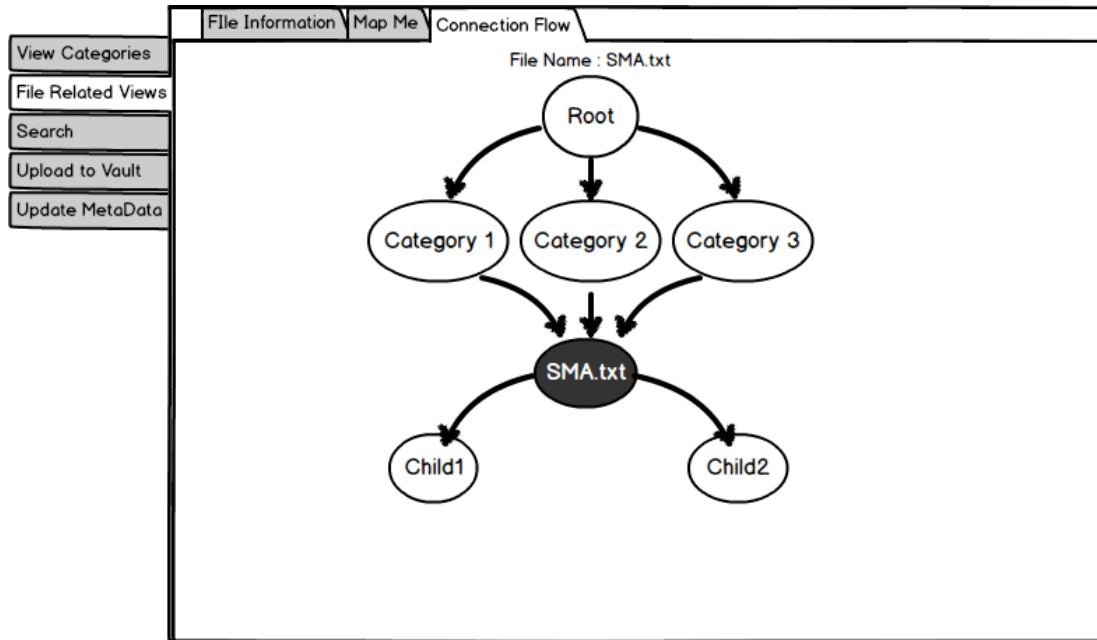


Fig 2.4: GUI for Connection Flow

2.5 Search Text Based

The Vault Client GUI accepts text query messages and returns the fully qualified names of all files containing that text in a clickable list. Text query searches are executed concurrently across files and the search accepts one or more categories, one or more text strings to seek and a switch setting indicating whether matching any query string or all query strings is required.

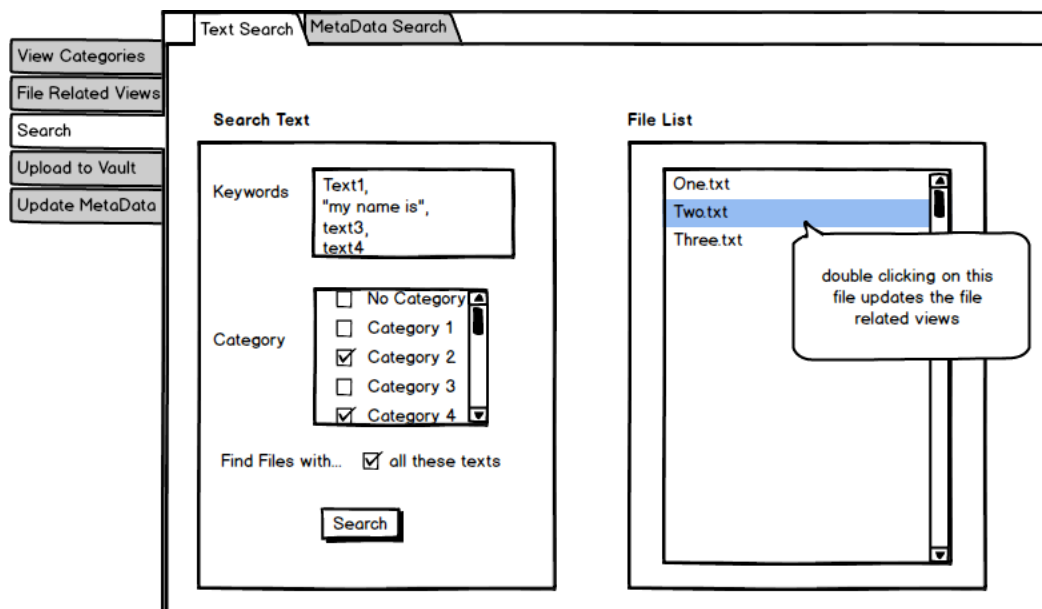


Fig 2.5: GUI for Text Search

2.6 Search MetaData Based

The Vault Client GUI accepts MetaData query messages specifying one or more categories and MetaData tags and returns the content of each element of the MetaData for every file in the file set. A MetaData query will always return its file reference and categories.

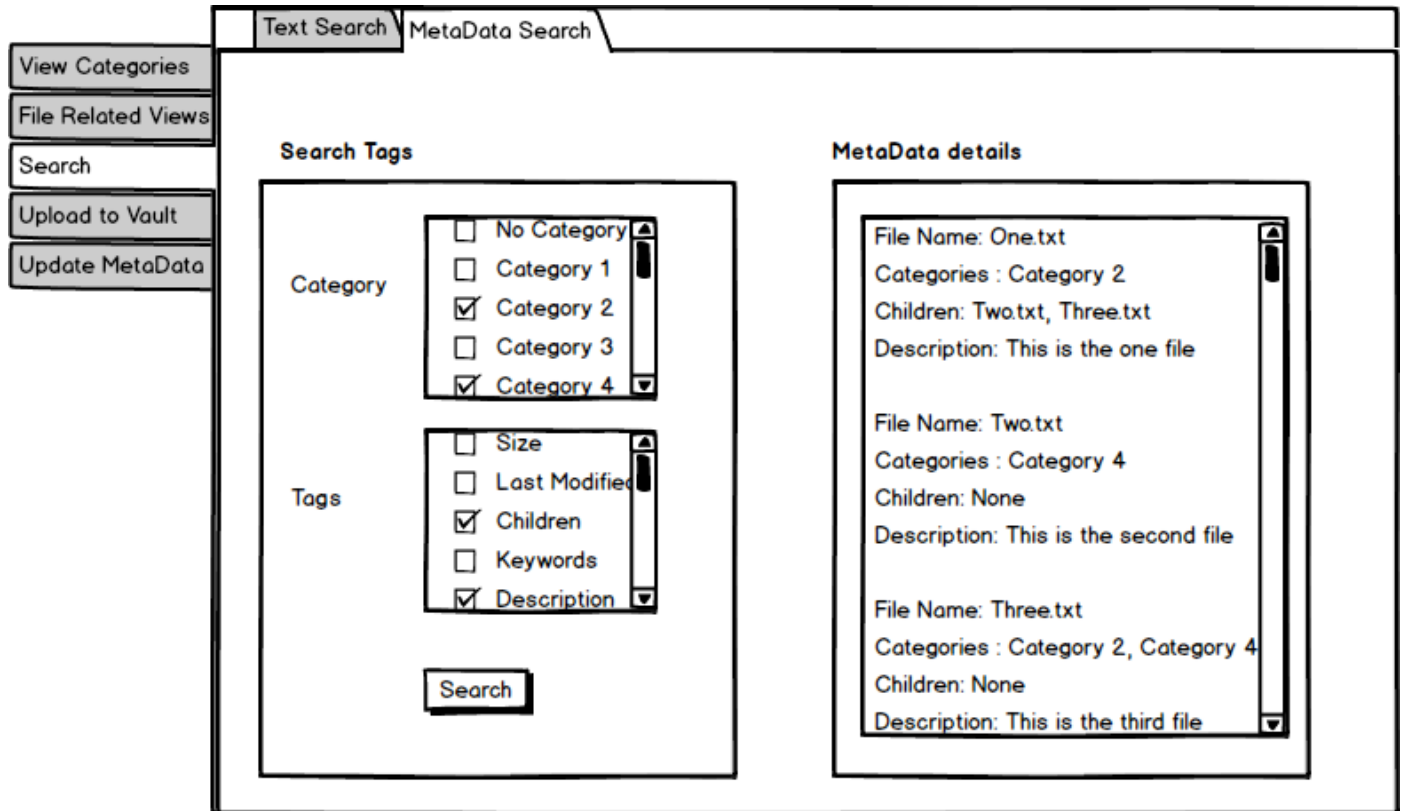


Fig 2.6: GUI for MetaData Search

2.7 Upload to vault

The Vault Client GUI provides a second file management view that supports local browsing for files to insert into the remote Repository and means for the user to provide description text and lists of child dependencies and keywords.

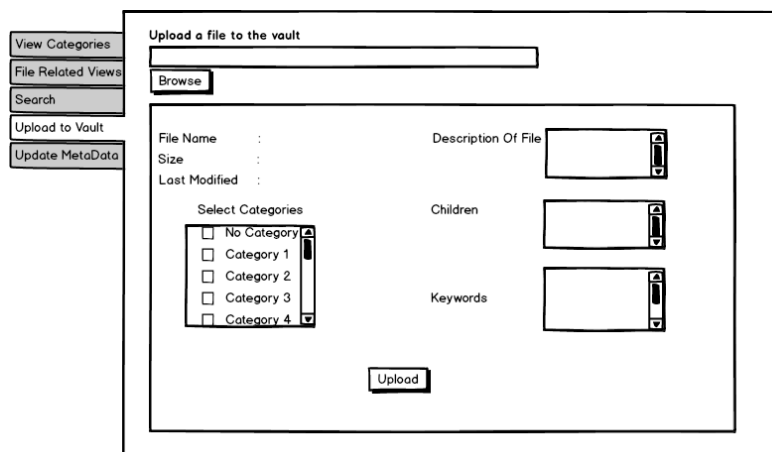


Fig 2.7: GUI for Upload

Once a file is selected for upload, the MetaData tool shall be used to automatically build a MetaData file from the information supplied here. This view shall also support editing MetaData information for the current file. This is as shown below.

The screenshot shows a web-based GUI for uploading a file to a vault. On the left is a vertical sidebar with buttons: 'View Categories', 'File Related Views', 'Search', 'Upload to Vault', and 'Update MetaData'. The main content area is titled 'Upload a file to the vault'. At the top, a text box shows the file path 'D:\My documents\CIS 681 SMA\Project 3\GUI\SMA_Again.txt' with a 'Browse' button next to it. Below this, file details are displayed: 'File Name : Test\SMA_Again.txt', 'Size : 1390 Bytes', and 'Last Modified : 09/06/2013, 4:00:09 PM'. To the right of these details is a 'Description Of File' text area containing 'This file talks about the various'. Below the file details is a 'Select Categories' section with a list of checkboxes: 'No Category', 'Category 1' (checked), 'Category 2', 'Category 3' (checked), and 'Category 4'. To the right of the categories are two list boxes: 'Children' containing 'WCF.txt' and 'C#.txt', and 'Keywords' containing 'Threads,', 'Distributed,', and 'Parallel'. At the bottom center of the main area is an 'Upload' button.

Fig 2.8: GUI for Upload – After File selection

On a successful upload it informs the user of this action with the help of an information message.

This screenshot shows the same GUI as Figure 2.8, but with a modal dialog box in the center. The dialog is titled 'MetaData Creation:' and contains the text 'File Successfully Uploaded' above an 'OK' button. The background elements of the GUI, including the sidebar, file path, file details, categories, children, keywords, and the 'Upload' button, are still visible but partially obscured by the dialog.

Fig 2.9: GUI for Upload – Message display

2.8 Update MetaData and add Categories

The Vault Client GUI provides editing MetaData files 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. This can be seen in the screens below.

Fig 2.10: GUI for Update MetaData and add Categories

Here on the GUI there is an option to select an existing MetaData file and also editable combo box for categories. So we can add a new category and click on save changes for the MetaData changes to be reflected.

Fig 2.11: GUI for Update MetaData and add Categories – MetaData File selected

As can be seen above the values of the current MetaData file is populated. If any of the tags are not present in the Initial MetaData file it creates a new Tag and appends to the existing MetaData file. The above UI also shows the addition of a new category into the MetaData file.

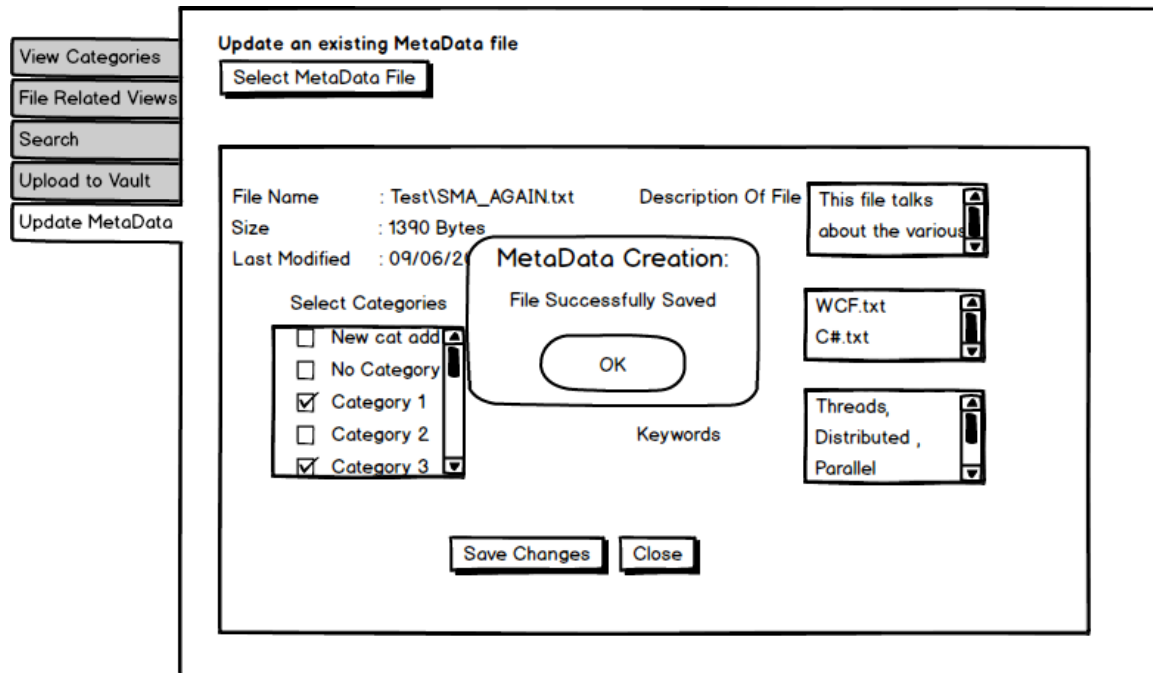


Fig 2.12: GUI for Update MetaData and add Categories – MetaData File changed

This process mainly involves an update of the versioning and checks from the server. Once it identifies that the process was a success. The GUI informs the user by using an information message to display the successful save of the new MetaData file.

3 Structure

3.1 High Level Architecture

The following diagram represents the partitioning of the Remote document Vault in different packages. There are three main packages in total with Executive driver for each as the main controller package.

Brief description of packages is discussed after the diagram.

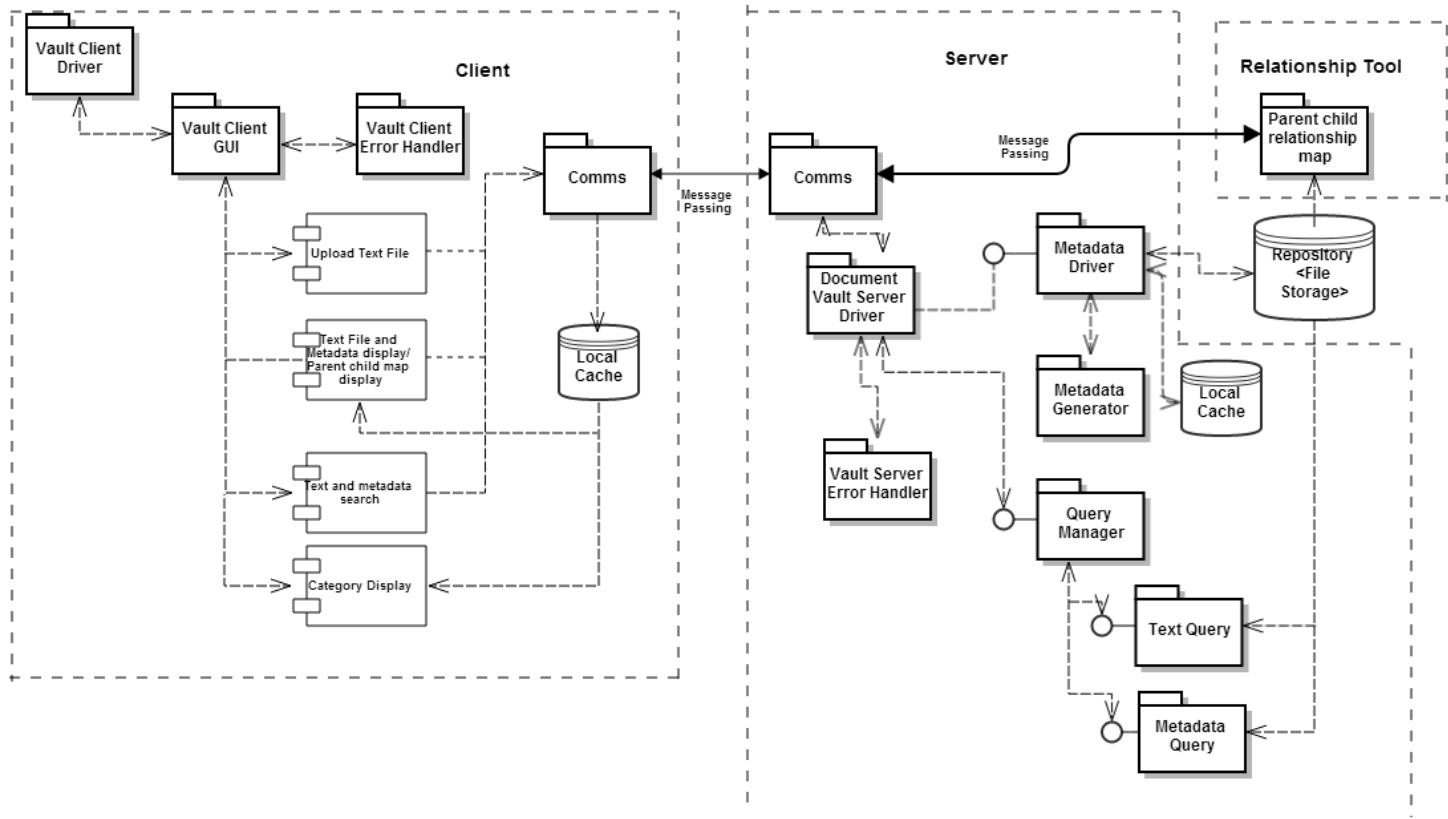


Fig 3.1 Package diagram for Remote Document Vault

3.1.1 Client

The client has several logical partitions each of them is explained in short here:

- Vault Client Driver*: This performs the role of a driver and initiates a call to the vault client GUI. It also acts as an entry point for the tool from the client side. It is responsible for below operations:
 - To invoke methods from other packages of the tool depending on the operation requested by the user.
 - Provide inputs to the invoked methods.
 - Store output of these methods and display it to user using the GUI package.
 - Use output of one package to instantiate methods of other packages.

- b) *Vault Client GUI*: This is used to instantiate the various components involved. It acts like a façade. A façade is an object that provides a simplified interface to a larger body of code, such as a class library. Vault Client GUI can:
- Make the library more readable
 - Reduce dependencies of outside code on the inner workings of a library, since most code uses the façade, thus allowing more flexibility in developing the system
 - Upload Text File: This provides a file management view that supports local browsing for files to insert into the remote Repository and means for the user to provide description text and lists of child dependencies and keywords. It also has a client side script which generates fundamental MetaData like last modified, size, File Name etc.
 - Text File and MetaData display/ Parent child map display:
 - Provides for an information view with two information panes, one for MetaData and one for file text of the currently selected file. The contents for this are first searched from the client's cache. If the contents are not found in the client's cache then it retrieves them from the Server.
 - Provides two navigation panes, one showing the current file's children and one showing its parents. Clicking on either a child or parent changes the current file to the file represented by the clicked link. These views are views of contents in the Document Vault.
 - The Plan to provide a graphical representation of the file in the vault. It shows a tree based structure where it creates a relation with the help of the map. Clicking on either a child or parent changes the current file to the file represented by the clicked link. These views are views of contents in the Document Vault.
 - Text and MetaData Search:
 - Accepts text query messages and returns the fully qualified name of every file containing that text in a clickable list box. Text query searches are executed concurrently across files by passing this to the server for processing. The text search accepts one or more categories, one or more text strings to seek and a switch setting indicating whether matching any or all query strings is required.
 - Accepts MetaData query messages specifying one or more categories and MetaData tags and return the contents of each element of the MetaData for every file in the file set. A MetaData query will always return the file reference and categories associated with the MetaData file.
 - Category Display: Category view displays all the categories in the Vault. Clicking on a category displays all the files in that category. Clicking on one of the category files makes that file the current file, "Text File and MetaData display/ Parent child map display" gets changed because of this. The category view can be accessed at any time as its one of the main tabs in the RDV.
- c) *Vault Client Error Handler*: This package is responsible to handle all the possible exceptions which will be thrown by all the components. It handles the exception and returns proper error message which user is able to interpret. These are raised onto the GUI of each object instance.
- Accepts the errors and tries to decide if the exception can be handled.
 - Non Critical issues: Issues which are raised because of inputs from the user or an incorrect set of instructions can be handled and returned here.
 - Critical Issues: If the system finds an Issue which cannot be handled, like a system crash, then it tries to get the system back to the last stable atomic saved state.
- d) *Vault Client Comms (Communications)*: The various functions of packaging are divided into primary, secondary and tertiary functions. In contrast with the primary functions, which primarily concern the

technical nature of the packaging, secondary functions relate to communications. Primary, secondary and tertiary functions are divided into the following sub-functions:

- Primary functions
 - Protective function : Checksum and integrity checks for the transfer
 - Storage function : The distribution into smaller sub groups and transfer
 - Loading and transport function : Sequencing and distribution
- Secondary functions
 - Service function : The mode of transfer that is used between the client and server
 - Guarantee function : Request for acknowledgments and retransmissions
- Tertiary functions
 - Additional function: Decide on whether a map update is required and communications to GUI requesting permissions for cache update if current file involves updates.

- e) *Local Cache*: A cache stores copies of documents and maps passing through it; subsequent requests may be satisfied from the cache if certain conditions are met. The cache at the Client's end uses.

➤ Freshness

Allows a response to be used without re-checking it on the origin server, and can be controlled by both the server and the client. For example, the Expires response header gives a date when the document becomes stale, and the Cache-Control: max-age directive tells the cache how many seconds the response is fresh for.

➤ Validation

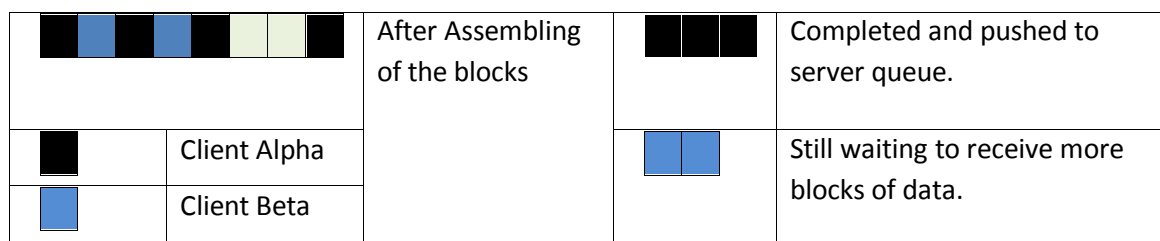
This is used to check whether a cached response is still good after it becomes stale. For example, if the response has a Last-Modified header, a cache can make a conditional request using the If-Modified-Since header to see if it has changed. If there is change in the Last-Modified header a weak validation is done first where a check is done to see if the files in the cache were affected. Strong validation helps to exactly replace the cache with the new updated values.

3.1.2 Server

- a) *Vault Server Comms (Communications)*: The various functions of packaging are divided into primary, secondary and tertiary functions. In contrast with the primary functions, which primarily concern the technical nature of the packaging, secondary functions relate to communications. Primary, secondary and tertiary functions are divided into the following sub-functions:

➤ Primary functions

- Protective function : Checksum and integrity checks for the transfer
- Storage function : The distribution into smaller sub groups and transfer
- Loading and transport function: Sequencing and distribution of the client based data. The data can be put in a list for each user connected. So parallel multiple user threads can be used to extract data. Once the data request is complete then the element is pushed into the main queue for processing.



	Client Gamma			Completed and pushed to server queue.
---	-----------------	--	--	--

- Secondary functions
 - Service function : The mode of transfer that is used between the client and server
 - Guarantee function : Request for acknowledgments and retransmissions
 - Tertiary functions
 - Additional function: Decide on whether a map update is required and communications to Map Tool requesting for cache update if current file involves updates.
- b) *Vault Server Driver*: This performs the role of a driver and initiates a call to the vault core services. It also acts as an entry point for the tool from the server side. It is responsible for below operations:
- To invoke methods from other packages of the tool depending on the operation requested by the clients.
 - Provide inputs to the invoked methods.
 - Store output of these methods and send it to the respective client.
 - Use output of one package to instantiate methods of other packages.
 - Decide concurrent execution threads.
- c) *Query Manager Service*:
- Text query messages are processed and Text Search returns the fully qualified names of every file containing that text. Text query searches shall be executed concurrently by spawning a thread for each file subsets which belong to one or more categories. Searches are processed for one or more text strings by the switch setting indicating whether matching any query string or all query strings is required.
 - MetaData query messages specifying one or more categories and MetaData tags and return the contents of each element of the MetaData for every file in the file set. A MetaData query will always return its file reference and categories. This can be done concurrently by spawning threads which can do multiple set based searches.
- d) *Vault Server Error Handler*: This package is responsible to handle all the possible exceptions which will be thrown by all the components. It handles the exception and returns proper error message which client is able to interpret. These are raised onto the object instance.
- Accepts the errors and tries to decide if the exception can be handled.
 - Non Critical issues: Issues which are raised because of inputs from the client or an incorrect set of instructions which can be handled and returned here.
 - Critical Issues: If the system finds an Issue which cannot be handled, like a system crash, then it tries to get the system back to the last stable atomic saved state.
 - Connection issues: with the help of counters estimate if there was a loss of connection or network. Retry and eventually move the entry down in the cache. If the client comes back then move the data back into the cache.
- e) *Meta Data Driver*: This package accepts the file path, file patterns from the client package which acts as file set for the tool. This package uses the capabilities of .Net framework for file handling mechanism.
- MetaData Generator: This package accepts MetaData details from the client and adds them to respective tag elements. It then forwards these tags to XML Processor package to create MetaData file.

- Version Management: When a modified file is checked into the Repository a new version of its MetaData file is created and added to the repository contents along with the modified file. These do not replace the previous versions.

3.1.3 Relationship Tool

Relationship Tool (Separate tool) searches MetaData files to build a map of parents for each file in the Vault. This tool is run each time the Repository starts. When a file is inserted into the Repository its parent relationship with each of its children is recorded in the map.

- Each MetaData file can be seen as a node. The Relationship Map can be built as discussed below.

```
Class MetaDataNode
{
Private:
String Filename;
List<string> Children, Keywords, Categories, Parents;
}
```

The MetaData of each file provides children pointed by the file, categories the node points to and other MetaData relevant to the file. With this update the nodes, for each node the parents are determined. It is done by making the parent of the child as the node itself. Each child node's parent is updated to the current node for each node in the map. This is done recursively.

```
<List<node>> MAP;
```

MAP gets built here recursively and when it finds that there are no parents for a node then the categories are taken as parents.

- The Relationship Map can be also used to build the category list of the files.
- Categories can be a hash key based list.
- When a node is being read from the map we push the node into the dependency hash list.
- After all the nodes are read in the map, we have a list of category based files.

4 Operations

The operations of the Remote Document Vault can be seen as smaller activities each involving the Client, Server and the relationship tool. Each activity diagram mainly focuses on how each operation works.

4.1 Server Startup

Each time the Repository starts the Relationship tool is run.

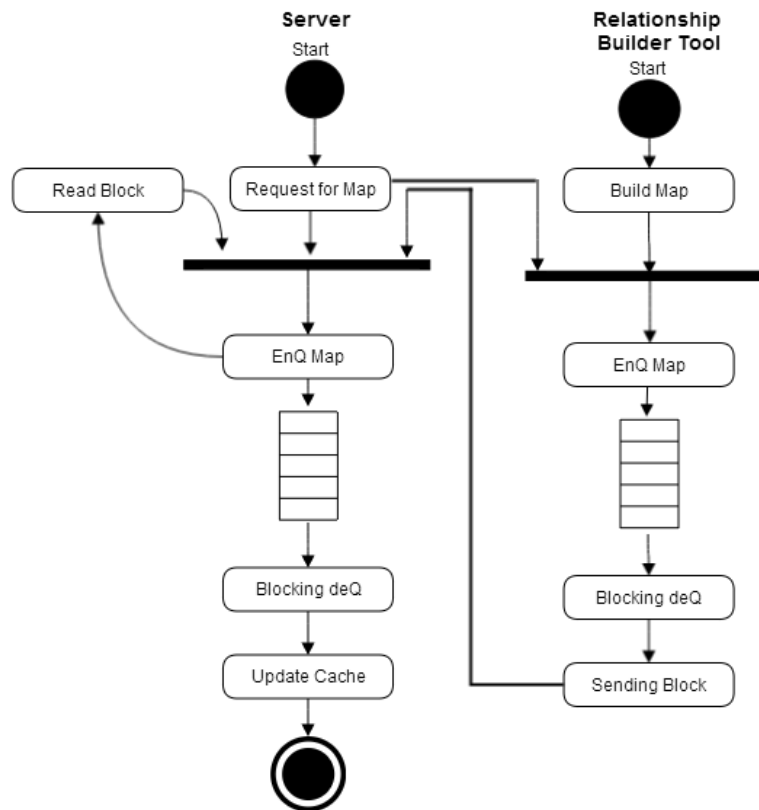


Fig 4.1 activity diagram for server startup

- The server starts up and requests for a map from the Relationship Builder tool.
- Relationship Builder tool builds a MAP based on the MetaData files stored in the Vault.
- Once the MAP is built and the RB tool gets a request message, it splits the MAP into fixed block size and pushes into a queue where disassembling and transmission are done to the server.
- Server receives all the blocks and once the transmission is complete, it updates the local cache copy present.

4.2 File Upload

A local file is selected and then pushed into the vault. This activity is explained here.

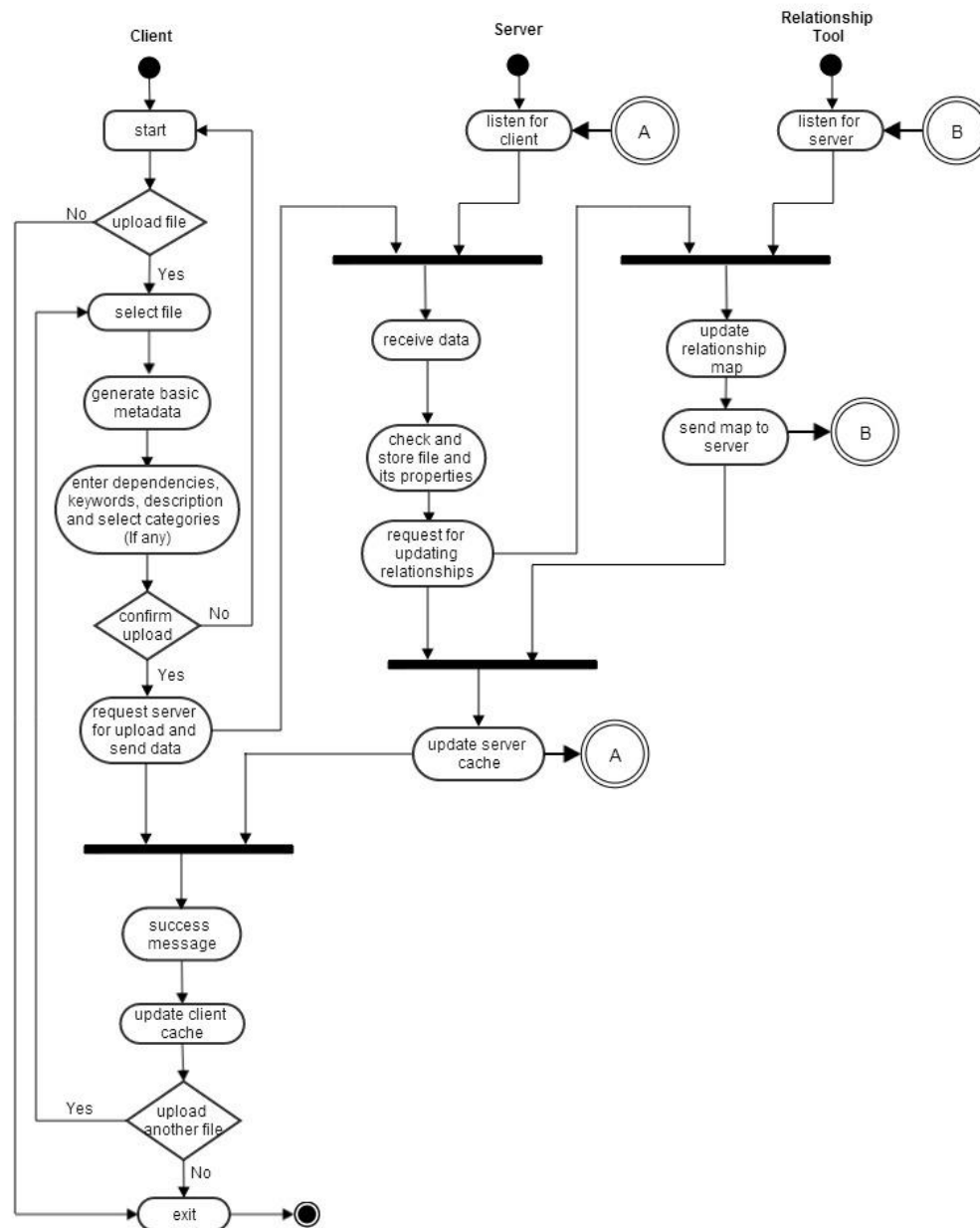


Fig 4.2 activity diagram for File Upload into the Vault

- This activity requires the server and Relationship Tool be running.
- A local copy of the file is selected.
- The user is presented with certain basic information of the file like the size, Last Modified, Categories present generated for the user.
- The user gets to edit and enter new MetaData relevant information.
- Once the customer confirms upload a request for upload to the server is done.
- The server identifies that the task involved is for a new entry into the vault.
- Server performs necessary checks and then stores the Text file and MetaData file onto to the Vault directory.
- As the MAP has been changed server requests the Relationship Tool for a new MAP.

- The new MAP is updated into the server's cache and the server sends a confirmation message for the successful upload.
- The Client gets the new MAP and updates its own cache for its relevant MAP.

4.3 Text Search

Accept text query messages and returns the fully qualified name of every file containing that text in a clickable list. Text query searches are executed concurrently across files and accept one or more categories, one or more text strings to seek and a switch setting indicating whether matching any query string or all query strings is required.

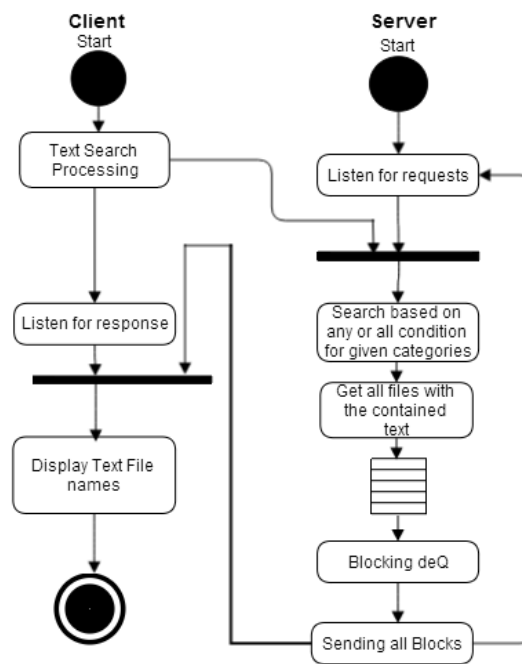


Fig 4.3 activity diagram for Text Search in the Vault

- Client makes a request for a text search with the required arguments.
- Server identifies the type of request as a text search
- Server concurrently searches file sets based on the given text search criteria.
- Server splits the file lists into small blocks and sends these blocks to respective clients.
- Client receives these and represents them in a clickable format.

4.4 MetaData Search

Client accepts MetaData query messages specifying one or more categories and MetaData tags and the client returns the contents of each element of the MetaData for every file in the file set back to the user. A MetaData query will always return its file reference and categories.

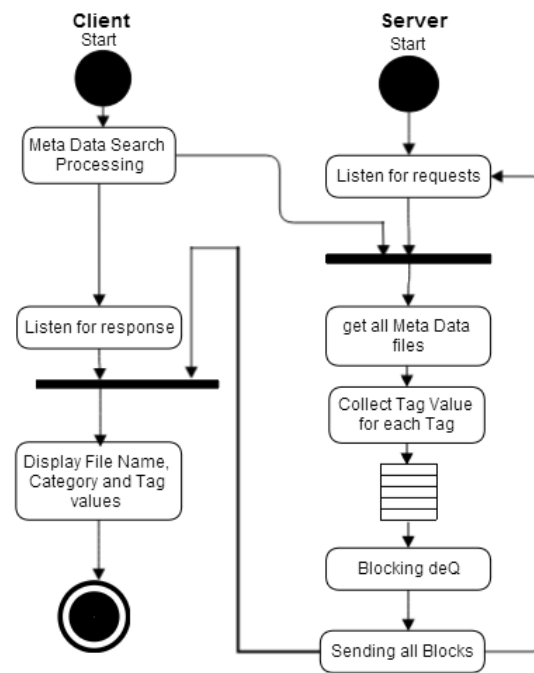


Fig 4.4 activity diagram for MetaData Search in the Vault

- Client makes a request for a MetaData search with the required arguments.
- Server identifies the type of request as a MetaData search
- Server concurrently searches file sets based on the given MetaData search criteria.
- Server splits the Tag values into small blocks and sends these blocks to respective clients.
- Client receives these and represents them in a text format.

4.5 Populate Parent and Children relationship

The client provides two navigation panes, one showing the current file's children and one showing its parents. Clicking on either a child or parent changes the current file to the file represented by the clicked link.

- Client gets the selected file and checks if this was already checked once prior in the current session.
- If the file was already loaded once in the current session then checks if the cache is updated for this relationship.
- In case of Valid Cache it avoids the call to the server and uses the cache to display the result in a two pane window.
- The Client system finds the cache invalid then it updates the cache by requesting a MAP from the server.
- Server gets requests from the client, identifies it as MAP request, and sends its MAP in blocks.
- Client updates the cache and display the result in a two pane window.

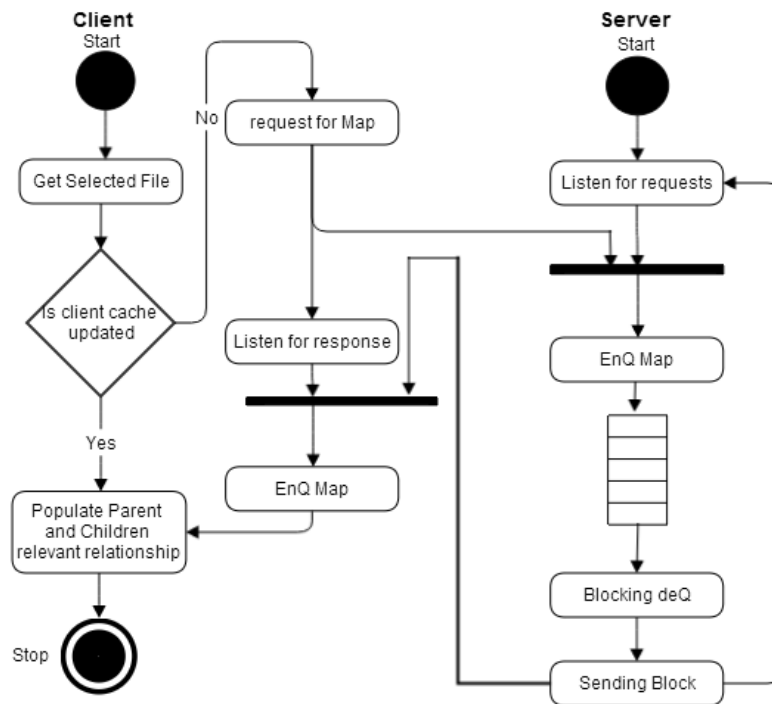


Fig 4.5 activity diagram for Populate Parent and Children relationship

4.6 Display MetaData and File Content

The client provides an information view with two information panes, one for MetaData and one for file text of the currently selected file.

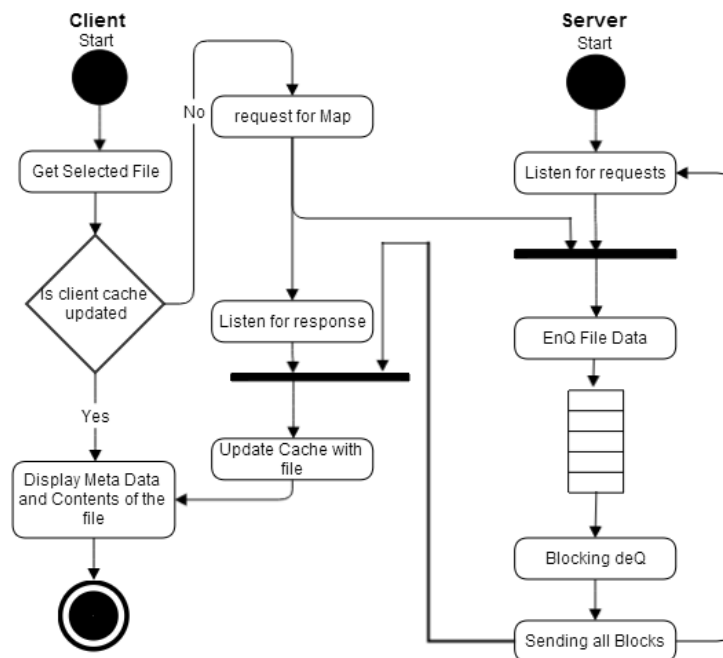


Fig 4.6 activity diagram to Display MetaData and File Content

- Client gets the selected file and checks if this was already checked once prior in the current session.

- If the file was already loaded once in the current session then it checks if the cache is updated for this data.
- In case of Valid Cache it avoids the call to the server and uses the cache to display the result in a two pane window.
- If the Client system finds the cache invalid then it updates the cache by requesting the file and its attached MetaData from the server.
- Server gets requests from the client, identifies it as a File request, and sends its File and MetaData in blocks.
- Client updates the cache and displays the result in a two pane window.

4.7 Vault MetaData Edit and Category Creation

Editing MetaData files by supplying a set of tag names and values requires a higher supervision. So the client here is considered as an administrator. 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. Categories can also be added and pushed into the Vault.

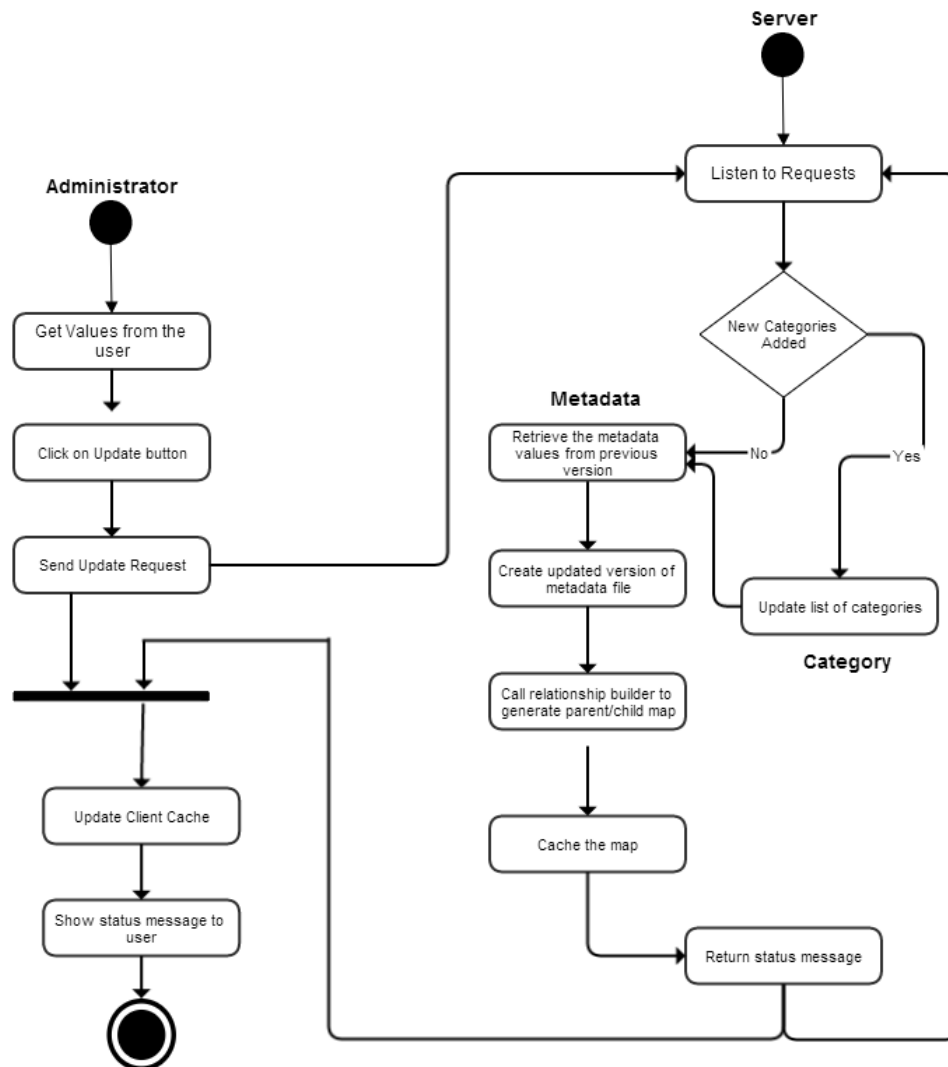


Fig 4.7 activity diagram to Vault MetaData Edit and Category Creation

- Administrator changes the selected MetaData File by changing the MetaData currently associated or adding a new Category.
- Server gets a request for MetaData update.
- Server checks if new categories have been added. If yes then adds them to the existing categories file.
- Server updates the new values of the tags. If there are tags that did not exist, the server adds these tags and updates these new values.
- Server updates the MetaData version to the new version.
- Server requests the Relationship tool to build the new MAP and updates its cache.
- Server returns the successful update message.
- Client (Administrator) updates its local cache.
- Client informs the user about the successful update.

5 Critical issues

5.1 Managing file contention

Competition for resources (File) leads to contention. The term is used especially in networks to describe the situation where two or more nodes attempt to transmit a message across the same wire at the same time. That is, two or more nodes may try to send messages across the network simultaneously. The contention protocol defines what happens when this occurs.

Repository will not allow the insertion of the files with the same name. Repository supported by this tool will not have sub directories hence tool will not support insertion of files with same name into different directories.

When server receives a request to upload a file with same name it sends an error message to the Vault client which will be displayed to the user. The case where an update or upload for the same file simultaneously occurs we have a file contention issue.

Solution planned here is to implement a token based multiple readers and a single writer approach. So a lock is requested on the file during editing and if the server grants this only then can the edit be performed on the file. When a file is being uploaded each client gets a token. If the files have the same name, the file with a higher token value gets permission for storage while the other is denied with an error message.

5.2 File Transfer with chunking

When sending large messages using Windows Communication Foundation (WCF), it is often desirable to limit the amount of memory used to buffer those messages. One possible solution is to stream the message body (assuming the bulk of the data is in the body). However some protocols require buffering of the entire message. Reliable messaging and security are two such examples. Another possible solution is to divide up the large message into smaller messages called chunks, send those chunks one chunk at a time, and reconstitute the large message on the receiving side. The application itself could do this chunking and de-chunking or it could use a custom channel to do it. In the application we assign a Globally Unique ID to file transfers. When file transfer request comes in client has to send a GUID assigned to the request message and the server agrees to this ID. Once each block is sent it attaches a tail element which informs about the order number of the block and end of transmission. The server then forms a separate queue for each GUID and orders the blocks based on the order numbers.

Chunking should always be employed only after the entire message to be sent has been constructed. A chunking channel should always be layered below a security channel and a reliable session channel.

5.3 Ownership policy

Who can edit a MetaData file? Access right for the file uploaded to the repository for users other than the up loader becomes a question of concern. There are multiple solutions possible to this issue. Two possible solutions are discussed here.

1. When two users (who are given access to the same MetaData file) try to save changes that affect the same MetaData, RDV displays the Resolve Conflicts dialog box for one of the users.

- In the Resolve Conflicts dialog box, read the information about each change and the conflicting changes made by others.
- To keep your change or the other person's change and go on to the next conflicting change, click Accept Mine or Accept Other.
- To keep all of your remaining changes or all of the other users' changes, click Accept All Mine or Accept All Others.

2. A more trivial solution could be:

- Only Author of the file or person who uploaded the file will have edit right to the files
- If a user wants to change the contents of a file, he needs to update the vault client and perform a new upload thus maintaining the consistency of the files.
- This might well require an authenticated login to the Vault client to identify the user, for this OCD it's out of scope and can be considered for future releases of the tool.

5.4 Working with multiple documents

As in all data modeling, when developing data models, it is good to consider the inherent properties and requirements of the application objects and the relationships between application objects. RDV models must also reflect:

- how data will grow and change over time, and
- Kinds of queries your application will perform.

These considerations and requirements force developers to make a number of multi-factored decisions when modeling data, including:

- Normalization and de-normalization.
- Indexing strategy.

To handle multiple documents and work with such a large data set we generally use cache. Here we consider the cache to be updated by the client. The server informs the client about the cache change and when the client needs the updated data it requests for a fresh copy. This shifts a lot of processing from the server to the client.

In the server we handle this by using a multithreaded approach using the core of the processor to a better use and making the response time for each transaction with client faster.

5.5 Building flexible Vault framework that accommodates specialization

One reasonable policy for check-in is that dependencies must already exist in the repository. If we use that policy we will need to define an open check-in state that allows changing parts of the check-in without creating a new version. So, for example, if we wish to check-in a File that has children not yet in the Repository we can do an open check-in and add the child packages whenever we are ready. When all of the dependencies are resolved, we close the check-in. Such a flexible vault Framework is not supported. A possible solution could be to use a generic interface and a rule based implementation which can be used for the specialization. The rules can be added at any point of time and would be immediately in effect.

5.6 Archiving issues

Document Vault supports versioning of files; any updates to the existing file will result in a new version to be stored in the repository. Soon the repository will have hundreds of older version of files which are not being used hence document vault has to support archiving of the older version of the files.

Solution could be:

- Archiving of the files will be done programmatically. Server will archive files which are not accessed over a year into the archive repository.
- User initiated archiving: The administrator can archive the files that are no longer accessed and thus reducing the number of files through which the processing is done.
- A check must also make sure that the files that are archived have no dependencies with the current files in the vault. It shouldn't leave any files that are dependent on the file that was archived.

5.7 Addition of New Categories

There can be multiple users wanting categories of their own which may not be of any use to the overall system, thus providing this privilege to users will affect the structured storage and retrieval of file supported by the Remote Document Vault.

- New categories can only be added by an administrator of the system and this functionality will be supported only by update MetaData GUI (Only assigned to the Administrator Client View).
- Users who require a new category should send a request to architect describing the need for a new category.
- Architect reviews the request and sends the request to administrator for adding the category.

5.8 Insertion of the file with child dependencies

When the user provides the metadata information during file upload, if the files provided as part of the dependencies are not present in the repository, building the parent map with such relationships will result in crash of the tool.

- To avoid the incorrect child dependencies, server has to validate the dependencies provided by the user by searching the repository for the metadata files. If files are not present failure message should be sent to vault client.
- If users are uploading multiple files in a top down approach i.e. first uploading the parent then its children, client has to indicate the multiple file upload with help of an indicator in the request message which will make the server to hold a lock until all the files are received and then validate the child dependencies in the cache and then start upload.

- Another solution would be to show advisory message to user perform multiple file insertions in bottom up approach i.e. first uploading the children then the parent.

6 Conclusion

Remote Document Vault" tool is a data repository that is a partial file based data management system. The major features revolve around insertion of files into the remote vault and queries on these files.

- Queries are classified into two parts
 - a) Based on the text contained in the file
 - b) Based on the MetaData of the file.
- Remote Document Vault (RDV) can be used to create a specific MetaData for each file which also maps a particular category.
- Can be used to build a hierarchical relationship between the files. The vault provides simple and informative view of the dependencies.
- It also helps the user to understand the various groups a file belongs to. Categories are used here for grouping the files.
- There are a few critical issues discussed in the document, these must be considered while building the application.
- It is built on a client-server based approach and thus can be used on different devices by the same user and can also be used for document storage and retrieval

7 REFERENCES

1. Jim Fawcett, Study Guide - Operational Concept Document (OCD), <http://www.lcs3.syr.edu/faculty/fawcett/handouts/CSE681/Lectures/StudyGuideOCD.htm>
2. Gliffy.com, Online Diagram Software and Flowchart Software, <http://www.gliffy.com/>
3. Balsamiq, Rapid Wireframing Tool, <http://balsamiq.com>
4. MSDN – the Microsoft Developer Network, <http://msdn.microsoft.com/en-us/library/aa717050.aspx>