CMSC424
MMDA System
Phase 1

Eric Yang
Emily Yue Wang

Table of Contents

1 Environment and Requirement Analysis

1.1 Project Introduction:

The purpose of this project is to create a Multi-Media Data Aggregator system that allows users to organize variety of files, executables, and HTML documents, into DAGRs (Data-Aggregates). This document will outline the overall function of the system, as well as analysis on specific tasks and system requirements, and specifications.

1.2 Project Scope

The project will be divided into three phases. The first phase involves identifying the main goals, tasks, limitations, and potential problems in order to provide a high-level overview of the system in general. In this step limitations, such as identifying which file-formats for videos, images, audio-files, etc. will be accepted, as well as determining important metadata for each of these file formats, and other problems will be considered and solutions will be proposed before any modeling or tasks are started. Tasks and subtasks will also be modeled during this phase so implementation of said tasks will be smooth in latter phases.

The second phase of this project is to establish a DBMS for the MMDA system, which involves modeling entities and relations for multi-media files and DAGRs. During this phase, tasks and subtasks modeled in Phase I will be emulated before being implemented in Phase III. During this phase the webserver will be developed graphically, and initial connections to the database system will be made.

During this phase tasks and subtasks will be fully implemented and operational. The webserver will be fully developed so that the MMDA system can be demonstrated.

1.3 Assumptions

1. The user will have access to the internet and can use the appropriate web browser to access the website
2. The user has sufficient knowledge of English to read instructions and operate the web site's functionalities
3. The files uploaded to the database will not be *outrageously* large files

4. The database and webserver can handle and store all files uploaded
5. The files uploaded have basic meta-data that can be stored

1.4 Technical Problems and Solutions

1. PROBLEM: Lack of full understanding about what elements of ‘metadata’ are important for each kind of file-type that will be used.
   SOLUTION: Research into what elements define specific file types is required.
2. PROBLEM: Which file types should be supported? How will their metadata be extracted?
   SOLUTION: For Microsoft Office Files whose file formats end in ‘x’ (docx, pptx,...) XML unzipping can be used to find metadata. For audio mp3 files ID3 python library can be used to scrape metadata. The hachoir-metadata python library can be used to extract meta-data from many sources, but does not support meta-data for MP4 files.
   http://stackoverflow.com/questions/7021141/how-to-retrieve-author-of-a-office-file-in-python
   http://id3-py.sourceforge.net/
   https://pypi.python.org/pypi/hachoir-metadata

3. PROBLEM: How will GUID’s be generated?
   SOLUTION: An online API will be used to generate GUIDs.
   https://www.uuidgenerator.net/api
4. PROBLEM: Single file uploads are easy with Django, how will multiple file uploads be handled?
   SOLUTION: This solution can be used to upload multiple files with Django
   https://www.youtube.com/watch?v=C9MDtQHwGYM
5. PROBLEM: Should a chrome extension or a firefox extension be made?
   SOLUTION: After research, chrome extensions seem to be easier to develop and require less development experience. Chrome is also a more popular browser, so a chrome extension will be made.
6. PROBLEM: A HTML document has many, many links in a chain to a large amount of HTML documents. Ex (URL1 -> URL2 -> URL3 -> ….. -> URL99999999999)
   SOLUTION: Create a list of sites and prompt the user if they want to add the sites to the database. If the user says no, have an attribute for HTML documents for “next site”. This next site will not be added to the DAGR, but if the user queries for this DAGR and they will be given the option to add this “next site” to the DAGR

2 Systems Analysis and Specifications

2.1 Description of Procedures

The whole Multi-Media Data Aggregator system will be hosted on a remote web server. It can be accessed from the user's perspective via web browser. The user can insert or delete documents or select different queries to explore the whole system.

2.2 Software / Resource Specifications

Django is a Python web-framework and as a result the whole Django application will be hosted on PythonAnywhere for its simplicity in hosting Python based web-frameworks. This is not scalable, because PythonAnywhere has limited disk space and website hits however will be fine for a demonstration of the MMDA system. Most parts of the application will utilize Django's robust functionality and database integration and thus will primarily be written in Python and utilize Python libraries and other Python APIs. The webpage will utilize HTML, CSS, and JavaScript aspects to meld both usability and visual appeal; this is very easy to do using Django's templating system. MySQL database will be used to store database information because Cloud hosting for this database is available on PythonAnywhere and the corresponding DDL and DML will be used to process user's DAGR insertions, queries and requests. Tika-python or hachoir will be used to parse and return metadata of the documents. Tika-python is a Python port of the apache tika library is a powerful way to parse and return the metadata of the documents, however the hachoir library may be a easier library to use because it is natively a Python library. The specific library used to parse metadata will be tested and determined through Phase 2 when a working model is created for the MMDA system.

2.3 Documents

2.3.1 Top Level Information Diagram

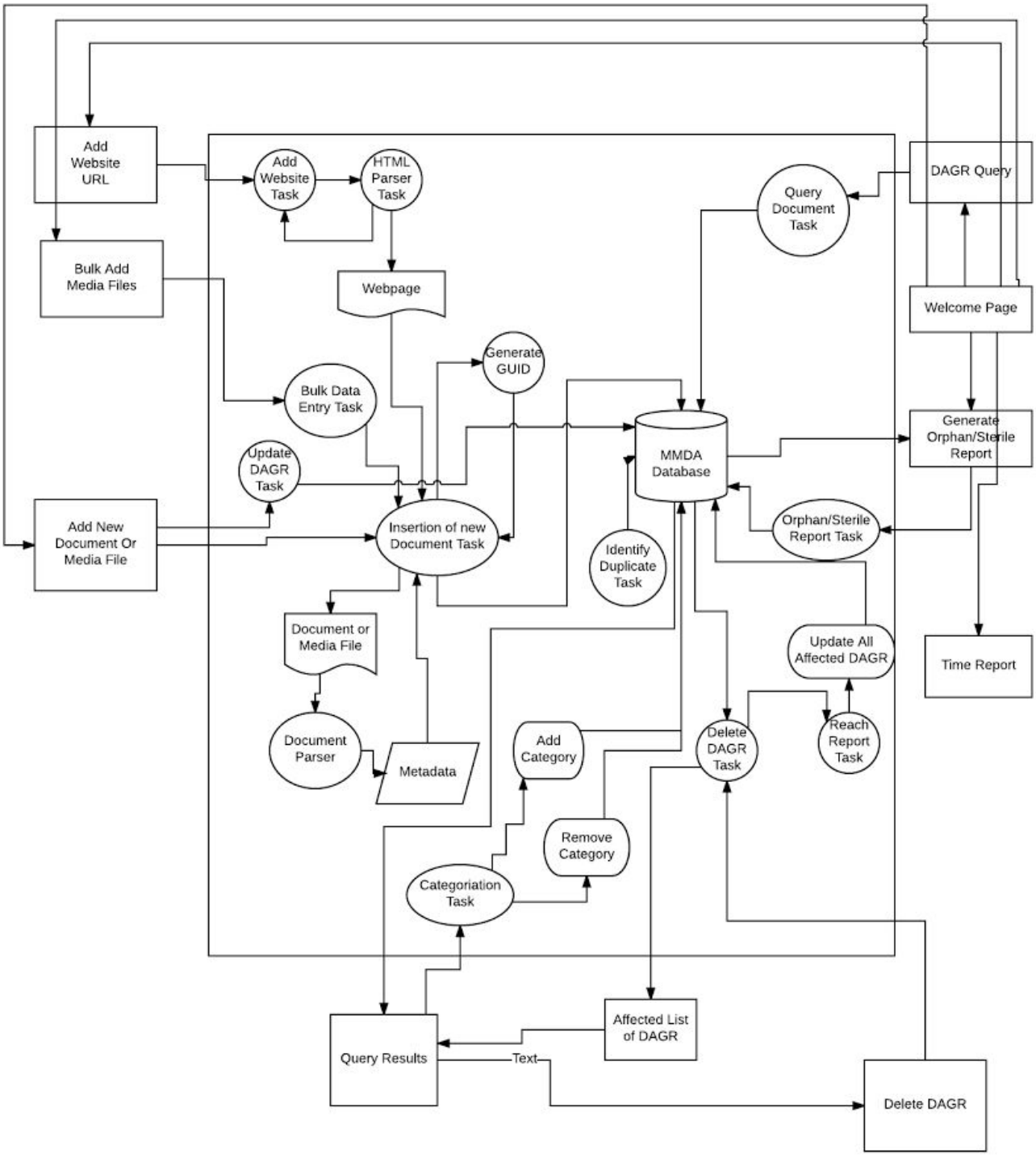

Figure 1.1

2.3.2 Tasks and Subtasks

Generate GUID

TASK NAME	Generate GUID
PERFORMER	Python
PURPOSE	Get a new Unique Identifier for the DAGR
ENABLING CONDITION:	User is inputting a new file or URL into the MMDA system
DESCRIPTION	This tasks uses an API call to generate a GUID
FREQUENCY	Every time a new document is added
DURATION	Minimal, only requires HTTP response from API call
IMPORTANCE	Critical
INPUT	None
OUTPUT	A new, unique GUID is generated
SUBTASKS	None
ERROR CONDITION	HTTP response fails

DAGR Insertion

TASK NAME	Insertion of a new document in the DAGR database
PERFORMER	Python/SQL
PURPOSE	Assign GUID to document and enter its metadata in the DAGR database
ENABLING CONDITION:	A user requests to add a new file from local file system
DESCRIPTION	1) This task Assigns GUID to document 2) Adds a document's metadata to the database. 3) The user can change the name which comes from the path and file names of document
FREQUENCY	Once per document
DURATION	Varies, depends on metadata parsing time as well as the document upload time.
IMPORTANCE	Critical
INPUT	Documents provided by the user
OUTPUT	Document metadata information that is inserted into the database
SUBTASKS	Generate GUID, Document Parsing/HTML Parsing
ERROR CONDITION	Required fields are not filled out / the file is not readable

DAGR Update

TASK NAME	Update an existing document in the DAGR database
PERFORMER	Python/SQL
PURPOSE	Allow users to replace existing files in the DAGR database with new updated parameters
ENABLING CONDITION:	A user requests to update a DAGR
DESCRIPTION	This task parses the document metadata once more and updates all fields that have changed in the DAGR database.
FREQUENCY	Once per update
DURATION	Varies, depends on metadata parsing time as well as the document upload time.
IMPORTANCE	Important
INPUT	Documents provided by the user
OUTPUT	Document metadata information is updated in the database
SUBTASKS	Document Parsing/HTML Parsing
ERROR CONDITION	Required fields are not filled out / the file is not readable

Bulk DAGR Insertions □

TASK NAME	Bulk DAGR Insertions
PERFORMER	SQL/Python/JavaScript
PURPOSE	Assign GUID to multiple documents and insert the metadata of many documents to system. Implement a bulk data entry utility.
ENABLING CONDITION:	A user requests to add multiple files from local file system at once.
DESCRIPTION	This task iteratively assigns GUIDs and adds multiple document's metadata into database
FREQUENCY	Every time documents' metadata needs to be added or updated
DURATION	Varies , depends on the number of the documents to be added into the database, upload time (file size), and parsing time
IMPORTANCE	Critical
INPUT	User selects a directory or multiple files to upload
OUTPUT	Data relations to be inserted into the database
SUBTASKS	Generate GUID, DAGR Insertion
ERROR CONDITION	Required fields are not filled out / the file is not readable

Document Parser

TASK NAME	Document Parser
PERFORMER	Python
PURPOSE	Extract all the metadata of a non-HTML component.
ENABLING CONDITION:	When user chooses to add document to the DAGR database
DESCRIPTION	Extract all the metadata from a document, video, mp3, etc… and then return corresponding metadata to the corresponding DAGR Insertion or Update tasks for database insertion.
FREQUENCY	Every new document or update
DURATION	Varies depending on parse time of different file types. (Using hachoir or Python Apache Tika wrapper)
IMPORTANCE	Critical
INPUT	User provided document
OUTPUT	JSON corresponding to the metadata of the file.
SUBTASKS	None
ERROR CONDITION	Document unreadable/Parsing fails

HTML Parser

TASK NAME	HTML Parser
PERFORMER	Python
PURPOSE	extract all the metadata of an HTML component.
ENABLING CONDITION:	When user chooses to add an HTML website using the chrome extension
DESCRIPTION	Extract all the metadata of a HTML document and then automatically store all the metadata in the annotation of the corresponding DAGR component, including images and external URLs. For a locally stored file the above tasks will provide the required metadata fields. For an external URL reference, you can harvest some keywords from either the URL text or the text surrounding the URL.
FREQUENCY	Every time a URL is saved as a DAGR
DURATION	Varies, depending on length of document. No uploads needed
IMPORTANCE	Important
INPUT	User provided URL
OUTPUT	The metadata of a document in the annotation of the corresponding DAGR component
SUBTASKS	Generate GUID/Add Website Task
ERROR CONDITION	The webpage is not readable

Add Website Task

TASK NAME	Add Website
PERFORMER	JavaScript
PURPOSE	Add an HTML document to the DAGR database
ENABLING CONDITION:	When user chooses to add an HTML website using the chrome extension
DESCRIPTION	Adds the HTML document to the DAGR database by calling the HTML parser
FREQUENCY	Every time a URL is saved as a DAGR
DURATION	Once for each URL addded
IMPORTANCE	Important
INPUT	User provided URL
OUTPUT	DAGR for HTML document or website is added to DAGR database.
SUBTASKS	Generate GUID/DAGR Insertion/HTML Parser
ERROR CONDITION	The webpage is not readable

Categorization

TASK NAME	Categorization
PERFORMER	SQL/HTML
PURPOSE	Allow user to create a wide variety of categories for DAGRs
ENABLING CONDITION:	User requests to make categories for different DAGRs The interface can enable the categories and subcategories creation and can support the deletion and insertion of DAGRs in categories.
DESCRIPTION	The categories for DAGRs can be created like county or University alerts, breaking news(CNN,WTOP, NYT), coupons (groupon, Amazon), travel promotions (Expedia, VacationToGo,Kayak) or any other user defined category for location oriented content (maps, housing, restaurants), personal document category, etc.
FREQUENCY	When the user requests
DURATION	varies , depend on the number of DAGRs that is inside the databases or will be added into databases.
IMPORTANCE	Critical
INPUT	Various DAGRs
OUTPUT	The HTML webpage of Categories and subcategories for DAGRs
SUBTASKS	None
ERROR CONDITION	Invalid parameter inputted by user

Deletion of a DAGR

TASK NAME	Deletion of a DAGR
PERFORMER	SQL
PURPOSE	Allow users to delete DAGRs that are not needed anymore
ENABLING CONDITION:	The user requests and approve to delete the DAGR and that DAGR is existed
DESCRIPTION	When a DAGR is deleted all appropriate metadata and its links have to be updated to avoid the problem of “dangling references” All metadata of the DAGR will be removed from the database, stored files will not be removed. The deletion of a DAGR should generate a list of affected DAGRs and ask for a user approval before actually deleting it.
FREQUENCY	Whenever the user requests
DURATION	Varies, depend on users’ needs and the number of DAGRs in the databases that the DAGR being deleted is referenced in.
IMPORTANCE	Critical
INPUT	Delete request and approval from user
OUTPUT	A list of affected DAGRs and the updated appropriate metadata and links that is related to the deleted DAGR.
SUBTASKS	Reach Query
ERROR CONDITION	The DAGRs that is supposed to be deleted does not exist. The related appropriate metadata and links is not updated after the deletion which causes the problem of “dangling references”

DAGR Metadata Query

TASK NAME	DAGR Metadata Query □
PERFORMER	SQL, Python, JavaScript
PURPOSE	To report to the user the results from a query as defined by the user.
ENABLING CONDITION:	The queries created by the user through simple interface JavaScript interface
DESCRIPTION	Queries are created from users to find DAGRs based on metadata attributes, such as date, author, type, size, or even based on keywords assuming that have been extracted or specified but the user and stored in the metadatabase.
FREQUENCY	Whenever the user requests
DURATION	varies , depends on the number of DAGRs in the database
IMPORTANCE	Critical
INPUT	Optional parameter of the metabase attribute to place restriction on the output list of DAGRs for example: date, author,type, size, keywords.
OUTPUT	The results of the query formatted in a table
SUBTASKS	Deletion of a DAGR
ERROR CONDITION	Invalid parameter provided by user or the DAGRs that user requests doesn’t exist;

Orphan and Sterile Reports

TASK NAME	Orphan and Sterile Reports
PERFORMER	SQL
PURPOSE	Find and report to user all the DAGRs that are independant which means they are not referenced by a parent DAGR or have no descendant DAGRs.
ENABLING CONDITION:	User's request
DESCRIPTION	Generate a list of all the DAGRs that are not referenced by a parent DAGR or have no descendant DAGRs.
FREQUENCY	Whenever the user requests
DURATION	Varies,depend on the number of DAGR in the database
IMPORTANCE	Important
INPUT	None
OUTPUT	The results of the query formatted in a table and displayed to the user
SUBTASKS	Deletion task
ERROR CONDITION	Invalid parameter from users

Reach Query

TASK NAME	Reach Query □
PERFORMER	SQL
PURPOSE	Find all the DAGR that can be reached by a given DAGR or those that point to it. This part support the further deletion and insertion of DAGRs
ENABLING CONDITION:	User's request
DESCRIPTION	Find the connections between different DAGRs which means all the DAGR that can be reached by a given DAGR or those that point to can be found for further insertion or deletion; The reach query may extend to several levels of indirection. But in this case you have to store appropriate information during the retrieval to avoid looping.
FREQUENCY	Whenever the user requests
DURATION	Varies,depends on the number of DAGRs in the database and number of DAGRs linked to the initial DAGR of the reach query.
IMPORTANCE	Important (necessary for deletion)
INPUT	GUID for the DAGR which the reach query will be applied to (basically start "Node")
OUTPUT	The results of the query formatted in a table for user usability and visability
SUBTASKS	Deletion task
ERROR CONDITION	Invalid parameter entered by users

Time-Range DAGR Report

TASK NAME	Time-Range DAGR Report
PERFORMER	SQL
PURPOSE	Create a report with several pre-selected data items for all the DAGRs created or entered in a given time range for user to review.
ENABLING CONDITION:	A request from the user
DESCRIPTION	The request queries the database for all documents which is limited by the user's defined date and time parameters.
FREQUENCY	Whenever user requests
DURATION	Varies, depends on the number of documents in the database
IMPORTANCE	Important
INPUT	The time range for DAGRs created
OUTPUT	The results of the query formatted in a table for user readability
SUBTASKS	None
ERROR CONDITION	Invalid parameter entered by users

Identify Duplicate Content

TASK NAME	Identify Duplicate Content
PERFORMER	SQL
PURPOSE	Replace the duplicate content in the DAGRs with different GUIDs with just one of the them. Remove the duplicate documents in different GUIDs.
ENABLING CONDITION:	Different GUIDS correspond to the same content
DESCRIPTION	Scanning and differencing the underlying documents. When two distinct GUIDs are referring to the same object, all occurrences of them should be replaced with one of the two.
FREQUENCY	Whenever there are two distinct GUIDs are referring to the same object. Whenever the user requests.
DURATION	Varies, depends on the number of DAGRs with same content.
IMPORTANCE	Important
INPUT	None
OUTPUT	The results of the query formatted in a table
SUBTASKS	Deletion task/Update Task
ERROR CONDITION	Invalid parameter from user

Webserver

A locally hosted, barebones with no functionality, web server implemented using Django has successfully been set up. Development of this web server into a MMDA system will take place moving into Phase 2 and further. During Phase 2, the webserver will be hosted on PythonAnywhere and functionalities creating a working model of the MMDA system will be created. In Phase 3, CSS and JavaScript will be applied to *beautify* the working model and create a more professional web application that is visually appealing to the user.

Phase 2

Purpose of Phase 2

There are several tasks that will be done during this phase to make the planning and requirement analysis of the system in phase 1 be in a more concrete stage:

1. E-R diagram, which help present the entities and the relationship between entities in our database;
2. Relational schema, which differs from the relation schema can help provide the collection of relation schemas and serves as a blueprint of a database that outlines the way data is organized into tables.during this stage. The specific attribute type will be defined to support the file type that the database will include.
3. Pseudo Code: after we are done with all the concept, model and relations analysis. We will dig deeply in this part to create the draft of implementation of each tasks using pseudo code, which provides a detailed description and guideline of how every task can be achieved by code. There are a lot of functions like insert file and delete DAGR file from the database that need a lot of detailed coding achievements.
4. Deploy web server on PythonAnywhere and connect to corresponding database

Problems Encountered:

1. Problem: Apache Tika Python bindings does not work on PythonAnywhere
   Solution: Use hachoir library to get metadata instead. For filetypes that are not supported by hachoir, create separate scripts using other libraries etc such as lmxl, docx,...
2. Problem: Project missing a distinct theme
   Solution: Gear the MMDA database towards utilizing Twitter API and grab metadata relating to standard user media files along with Tweets
3. Problem: Webserver deployment issues (many)
   Solution: Used online resources to solve issues as they arose

E-R Model

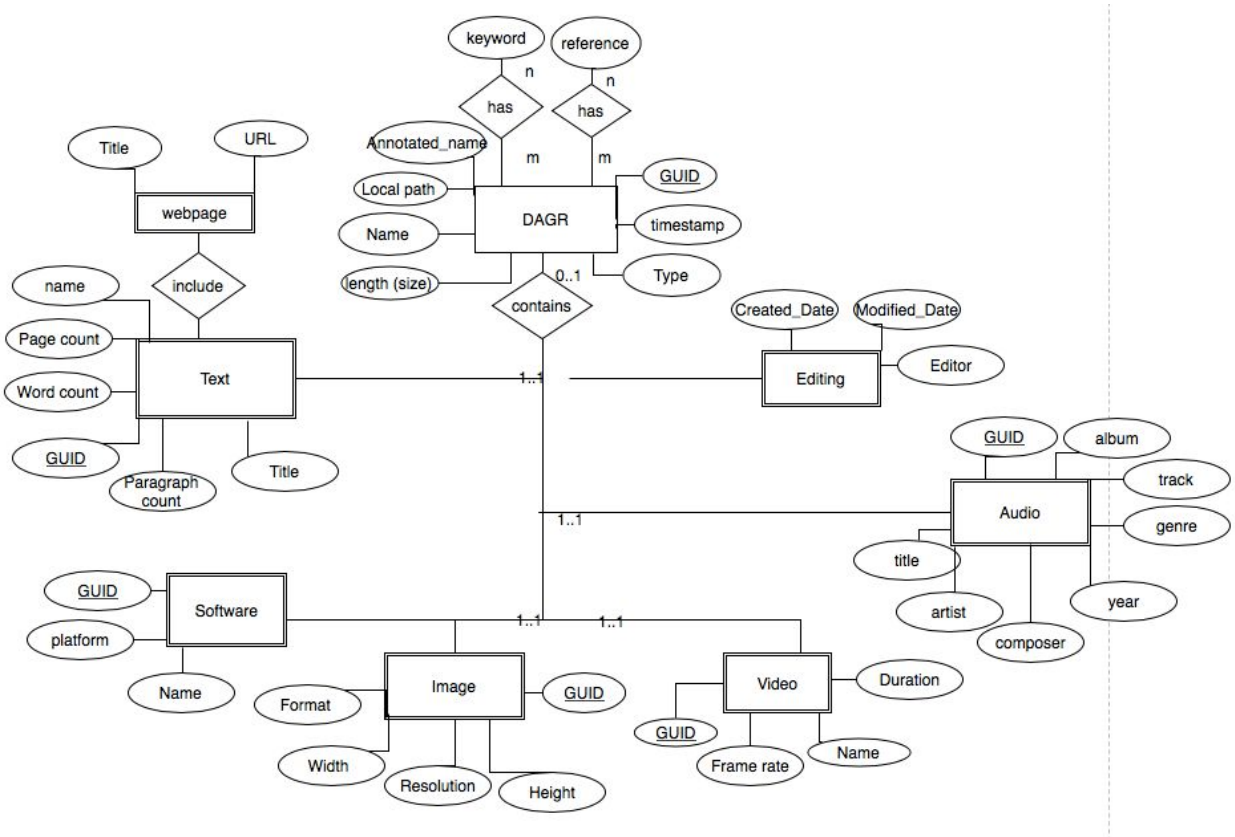

Figure 2.1

Relational Schema

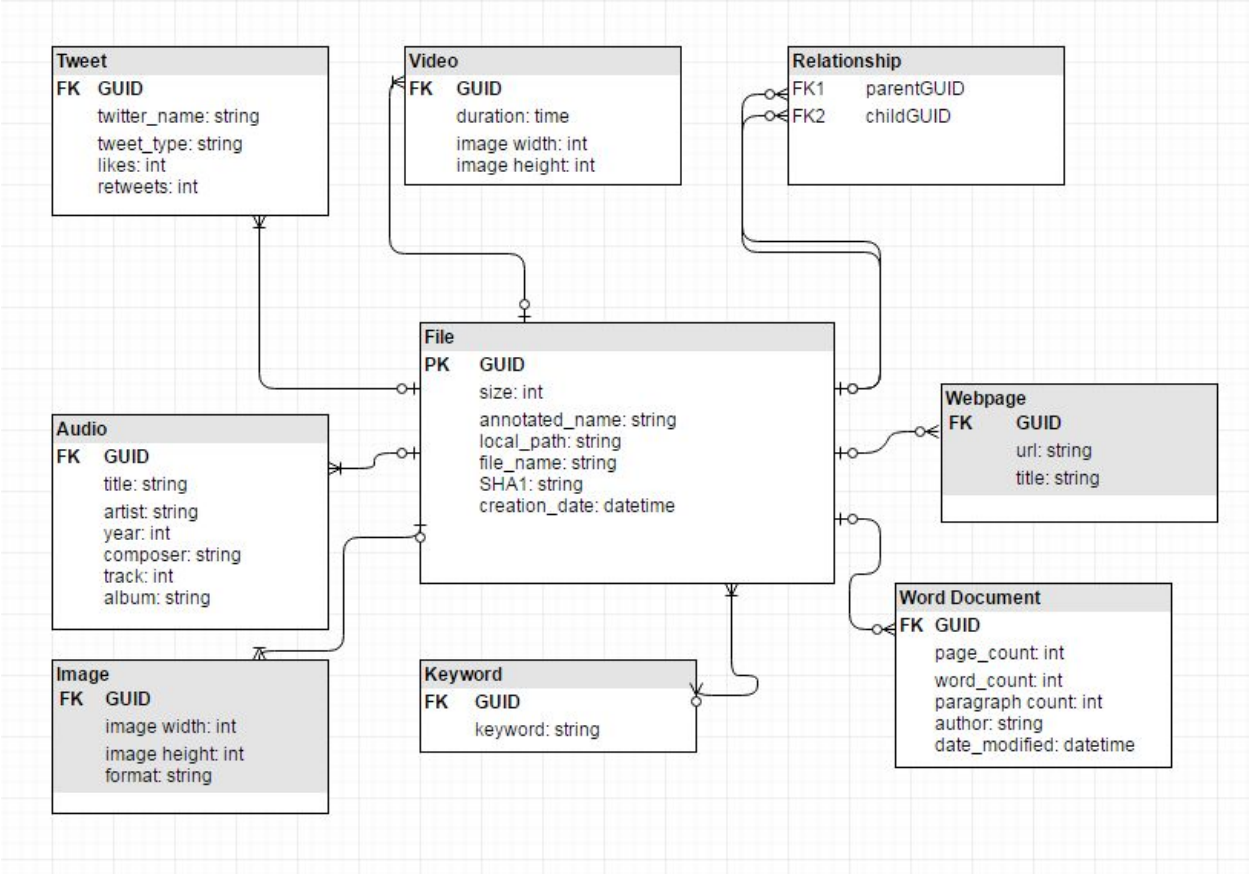

Figure 2.2

Pseudo Code

DAGR Insertion

```
def insertFile(file_1, parentGUID):
    Hashed_file = SHA-1(file_1)
    Named_files = Select * from Files where Filename='file_1'
    ##found the same file name with the same hashcode,then we do nothing
    for file in Named_files:
            If Hashed_file == Hashed_code from file
                Return "the file contained the metadata already exists."
            Else
                GUID=generateGUID(file_1)

                if parentGUID:
                    childGUID = GUID
                Keyword_file1 = input from the user
                Annotated_name = input from the user
                Attributes_file1 = hachoir(file)
                attributes = [Annotated_name, Attributes_file1, parentGUID, childGUID,local_path, GUID]
                Insert into Files values (attributes)
                Insert into keywords values(GUID,Keyword_file1)
                Return GUID
```

DAGR Update

```
def updateFile(GUID_1):
    # Updates the file based upon GUID
    ###two kind of  information to be updated
    new_ annotation = input by the user
    new_keywords = input by the user
    remove_keywords = input by the user
    add_child = GUID selection by the user
    add_parent = GUID selection by the user
    remove_children = GUID selections by the user
    remove_parents = GUID selections by the user

    UPDATE Files
        set old_annotation = new_annotatedname where GUID==GUID_1
    for keyword_1 in remove_keywords:
        DELETE from Keywords where GUID = GUID_1 AND keyword = keyword_1
    for keyword_1 in new_keywords:
        INSERT in Keywords values(GUID_1, keyword_1)

    if add_child:
        create relation between GUID_1 and add_child
    if add_parent:
        create relation between GUID_1 and add_parent
    for parent in remove_parents:
        DELETE from relationship where parentGUID = parent and childGUID = GUID_1
    for child in remove_keywords:
        DELETE from relationship where parentGUID = GUID_1 and childGUID = child
```

Document Parser

```
def parse_document(File):
    get_file_type(File)
    if file_type is in supported_filetypes:
        if file_type == "docx":
            attributes = parse_docx(File)
        else:
            attributes = hachoir(File)
    return attributes
```

HTML Parser

```
def parse_document(File):
    get_file_type(File)
    if file_type is in supported_filetypes:
        if file_type == "docx":
            attributes = parse_docx(File)
        else:
            attributes = hachoir(File)
    return attributes
```

Add Website Task

```
def Add_Website(url_1):
    # check if url already is inside DB
    if SELECT *, COUNT(*) from Webpage where url = url_1 != 0:
        return
    GUID_1 = generateGUID(url_1)
    Keywords = input from the user
    Annotated_name = input from the user
    local_path = None
    name = Use regex to get website name
    attributes = [GUID_1, Keywords, Annotated_name, local_path, name]
    Insert into Files values (attributes)
    Insert into keywords values(GUID, keywords)
    HTML_parser(url_1, GUID_1)
    Return GUID
```

Deletion of a DAGR

```
def DELETE_DAGR(GUID_1):
    boolean response = Prompt_User("Do you want to delete this record")
    if response:
        parents = SELECT UNIQUE parentGUID from Related where childGUID = GUID_1
        children = SELECT UNIQUE childGUID from Related where parentGUID = GUID_1
        affected = set(parents + children)
        alert("These dagrs will be affected" + affected)
        response = prompt("Are you sure you want to delete this DAGR?")

        if response:
            DELETE * from table FILES where GUID=GUID_1
            DELETE * from table relationship where parentGUID = GUID_1
            DELETE * from table relationship where childGUID = GUID_1
            return affected
        else:
            return []
```

DAGR Metadata Query

```
def DAGR_QUERY(keywords):
    query_type = Keyword Search or GUID Search or Annotated Name Search
    if Keyword Search:
        Select All DAGR with keywords[0] or keywords[1].... keywords[n]
    elif GUID Search:
        Select * From Files where GUID = keywords[0]
    elif Annotated Name Search:
        Select all DAGR where TITLE includes keywords[0] or keywords[1]...
    else:
        error
    return Queried DAGRs
```

Orphan and Sterile Reports

```
def DAGR_QUERY(keywords):
    query_type = Keyword Search or GUID Search or Annotated Name Search
    if Keyword Search:
        Select All DAGR with keywords[0] or keywords[1].... keywords[n]
    elif GUID Search:
        Select * From Files where GUID = keywords[0]
    elif Annotated Name Search:
        Select all DAGR where TITLE includes keywords[0] or keywords[1]...
    else:
        error
    return Queried DAGRs
```

Reach Query

```
def Reach_query(GUID_1):
    A = SELECT * from FILES
        WHERE GUID in
            (Select CHILD_GUID from relationship
            where relationship.parentGUID = GUID)
    return A
```

Time-Range DAGR Report

```
def Time_Range_Report(start_datetime, end_datetime):
    A =Select * from Files where Files.creation_date>start_datetime and
        Files.creation_date < end_datetime

    return A
```

Webserver Status:

A running version of the webserver has been established at CMSC424.pythonanywhere.com. It currently does not support any MMDA functionalities, however a demo using the Twitter API has been used to grab image URLs from tweets (CMSC424.pythonanywhere.com/use_twitter/gallery)