TimeIn

A temporal visualization for file access

Jeffrey Borden
jborden@gmail.com

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in the Department of
Computer Science, New York University

December 28, 2006

Approved: _____
Dennis Shasha, Research Advisor

Acknowledgements

I would like to thank Professor Dennis Shasha for his continued support and guidance. I am also grateful to my two co-researchers Chris Harrison and Stacey Kuznetsov.

Research and Development

TimeIn is an extension of the Kronosphere Project developed over the course of Spring 2006 by Jeffrey Borden, Chris Harrison, Stacey Kuznetsov and Dennis Shasha.

©
Jeffrey Borden
All Rights Reserved, 2006

Contents

1. Introduction **5**

2. Background **5**

2.1 Hierarchical File Systems 5

2.2 Web Based Image and Document Management Systems . 6

2.3. Desktop Search 7

3. Related Work **7**

3.1 Yep 7

3.2 Google Picasa 9

4. TimeIn Concepts **12**

5. TimeIn Feature Overview **13**

5.1 Overview 13

5.2 Indexers 15

5.3. Timeline Browser 18

5.4 Cluster Browser 25

6. Major Algorithms **28**

6.1 Keyword Extraction 28

6.2 Distance Metric 30

6.3 Clustering Algorithm 31

7. Software Architecture **33**

7.1. Overview 33

7.2. Repository Schema 33

7.3. Indexer Architecture 36

7.4. Timeline Browser Architecture 40

7.5 Cluster Browser Architecture 55

8. User Studies 56

9. Conclusion and Future Work 58

10. References 59

Appendix A. User Study Questionnaire 61

1. Introduction

Recent advances in storage technologies and network bandwidth have led to an explosive growth in the number of and complexity of file objects encountered by users. Existing technologies for browsing, searching and managing this content look increasingly inadequate to meet this growth in complexity, as they require too much people time. In this paper we present TimeIn, a platform-independent system for browsing and (self-) organizing large sets of unorganized file content.

2. Background

2.1 Hierarchical File Systems

Hierarchical file systems are by far the most widely used mechanism for managing and searching content are hierarchical file systems. These systems are engrained in the everyday experience of most users. If users invest time and effort into organizing these file systems, they can potentially provide an intuitive and useful mechanism for managing and browsing thousands of files.

Unfortunately, as most users will report, this system breaks down when users fail to maintain organization or if so much content arrives that they have no time to do so. Concretely, imagine that your friend Bob sends you the 1,000 best photos of a photographer who has been traveling several months over many continents. Putting them all into a directory "Bob's photos" doesn't help much, particularly if the photos themselves have names like 2034.jpg. We seek a way to search and potentially structure data that emerges from properties of the files: when they were created, their linguistic

content, and their image properties. Each such property may give an overlapping subset of the files, e.g. beach scenes may be present at many different time points. Hierarchical file systems encourage a partitioning of files (mitigated only by copying or linking), so do not offer this flexibility.

2.2 Web Based Image and Document Management Systems

The past few years have seen the emergence of web based sites for storing and sharing file objects. Sites such as flickr.com and youtube.com have gained mainstream acceptance and are increasingly used as users' primary system for storing and organizing content.

While these sites provide promising new classes of features, migration away from traditional local file systems has introduced a variety of issues.

Users migrating content piecemeal will have segmented (or duplicated) their file objects across file systems and possibly across a variety of sites. No real mechanism exists to provide an overview of content across all storage mediums.

Users are constrained by the features provided by these web sites for browsing and searching. Even worse is the fact that users utilizing a variety of sites may now have to adopt different new techniques for organizing the content on each of these sites. There are no real standards for browsing and searching content that have been adopted by the majority of sites.

All of the issues introduced by user supplied content organization inherent in local hierarchical file systems are experienced even more acutely in these web based systems.

By offering more mechanisms for describing content, users are even more prone to adopt organizational structures that are unintuitive to other users.

2.3. Desktop Search

Local file system search applications such as Google Desktop Search[1] and Spotlight[2] provide advanced mechanisms for content based search of text based file objects.

While these systems have proved quite useful for querying sets of file objects, they provide little support for browsing content. Consequentially users must be aware of specific topics of interest for the applications to be of any use.

In providing search results, these systems typically provide a limited view of results, often a simple flat list of files. This simple view of search results makes it difficult to illustrate the relationships between the files returned from the search and the other files in the system.

3. Related Work

3.1 Yep

Yep[3] is a Mac OSX application for scanning, managing and browsing PDF files stored on a local file system. It is designed to provide useful mechanisms for content based browsing and also provides an overview of an entire set of file objects as a table of thumbnails. These objects are automatically annotated with the file name and the timestamp of each file. Users can further annotate each object by creating user defined tags to describe each document.

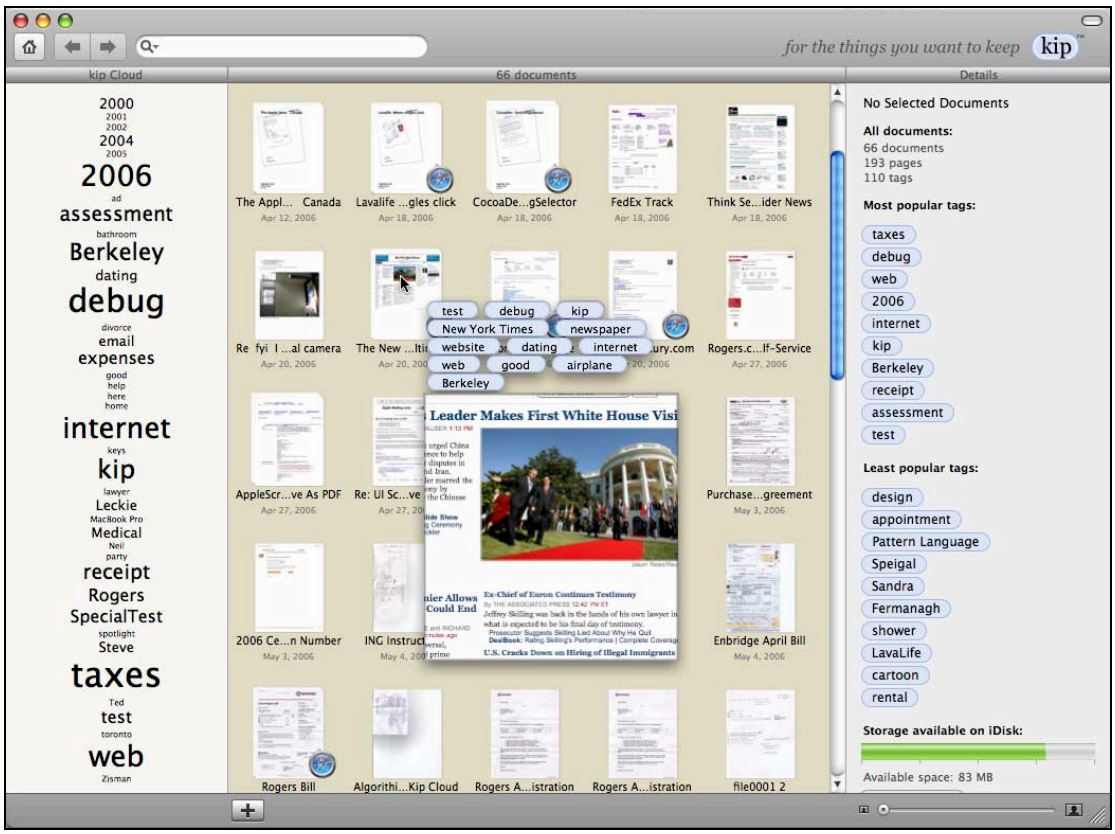

Figure 1. Yep's content based browsing

Yep provides a variety of additional browsing and searching mechanisms that users can use to filter the set of objects presented in the browser. A tag cloud browsing mechanism is provided to filter based on tag annotations. Users can also perform search queries on the tag and title fields of the documents. Yep also provides a mechanism for importing newly scanned documents.

As illustrated below in **Table 1**, while Yep provides a variety of useful features, it is lacking a few that would be found in an ideal system. Though Yep provides an overview of all items in its repository, it lacks the ability to group items by content in this view. While Yep provides users with the ability to annotate content with tags, there is no mechanism for automatic tag annotation. There are no features to allow users to organize

content into user defined sets. Yep is also constrained to be used on a local file system, there is no mechanism provided to use web based file sharing sites.

3.2. Google Picasa

Google's Picasa[4] is a Windows based application for managing, searching and browsing image files. It can import images both from the local file system as well as the web based Picasa album site. The application is fully integrated with the Picasa album site and can be used to export local content to web based albums.

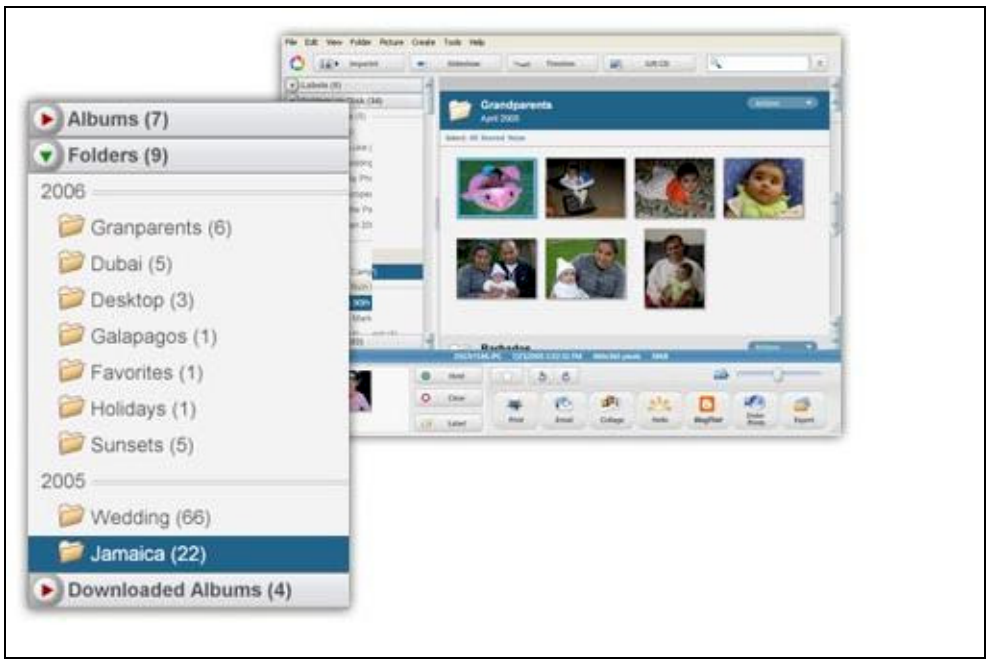

Figure 2. Picasa's table based content browsing

Picasa also provides a variety of features for building slick, polished presentations of images. While the majority of Picasa's features seem be geared towards building these presentations, it does provide a number of content based browsing features.

The browsing system is based on a table based view of image thumbnails. While this view provides an overview of all indexed content, content is grouped into the parent folders of images stored on the local file system and user defined Picasa albums.

A timeline based browsing system is also implemented. The timeline will show the set of file system folders and Picasa albums annotated and organized by timestamp. Users cannot view a timeline of individual image objects.

Images are annotated by filename and either their parent folder from the local file system or user defined albums. Users can perform search queries based on these two fields.

Picasa also provides a rich set of features for editing its indexed images and exporting content to the local file system.

As illustrated below in **Table 1**, while Picasa provides a variety of useful features, it lacks several useful ones. Most notably, there is no mechanism to annotate file objects other than creating user defined albums. Picasa is relies heavily on users to organize file content either on the file system or into user defined albums. Users can only search for files only based on the title of the file or its containing folder. There is no mechanism to browse for objects based on their individual timestamps, only the timestamp of their containing folders. Picasa does provide a variety of unique features for exporting content, however.

Features	Yep	Google Picasa	TimeIn
Provides an overview of objects with content preview thumbnails	X	X	X
-organizes overview of content into groups of related objects		Grouped by file system folders	Grouped by file creation timestamp
Provides a graph based view of image objects grouped by content similarity			X
Browse by file system structure		X	X
Browse items by timestamp	X	Only displays folder timestamps	X
Allows users to annotate objects with tags	X		X
Provides automatic generation of tags based on file content			X
Provides a text cloud browsing mechanism for tag annotations	X		X
Provides users the ability to search file titles		X	X
Provides users the ability to search tag annotations	X		X
Provides users the ability to search files by text content	X		X
Provides users the ability to search files by image content			X
Provides users with a mechanism to organize objects into albums		X	X
Provides users with a mechanism to export organized content to the file system		X	
Provides users with a mechanism to export organized content to a web based album site		X	
Provides users with the ability to import objects from web based file sharing sites		X	X

Table 1. A survey of features found in file browsing systems

4. TimeIn Concepts

TimeIn provides a single unified browsing experience for users. It seeks to unify the various sets of file objects stored on multiple systems; currently users can import content from a local file system or flickr.com.

In keeping with this concept, TimeIn was designed not to partition file objects into disjoint sets. A central unified view of all available content is initially provided to users. To navigate this content, users are provided with a variety of mechanisms for filtering the set of objects presented.

This central view highlights the relationships among files, allowing users to more easily browse objects by content. TimeIn will use this view to illustrate temporal relationships as well a simple overview of each object's text and image content.

To highlight the temporal relationships between files, the central view uses a timeline to organize objects by their associated timestamps. These objects are also annotated with a set of automatically populated text tags describing the content of the objects. For text based objects, TimeIn will analyze the content of the file and attempt to extract a set of descriptive phrases. For image based objects, TimeIn will annotate the object with the most frequently used colors of the image. Users have the option of augmenting these automatically generated tags by defining their own descriptive tags.

While providing novel features for browsing and searching content, TimeIn retains many of the original organizational features of existing systems. When content is imported from a hierarchical file system, users can still browse by the original hierarchy structure.

TimeIn also retains the PhotoSet structures associated with content imported from flickr.com. Just as TimeIn provides users with the ability to search objects by text content, it also provides users a mechanism to search image objects based on color content. Users can also organize content into user-defined "albums" of objects. These albums can then be used to filter the set of objects on the timeline.

5. TimeIn Feature Overview

5.1 Overview

TimeIn is composed of essentially three distinct sub-applications: the Timeline Browser, the Object Indexers (File and Flickr,) and the Cluster Browser. While the only application explicitly started by the user is typically the Timeline Browser, all three can be invoked independently of each other.

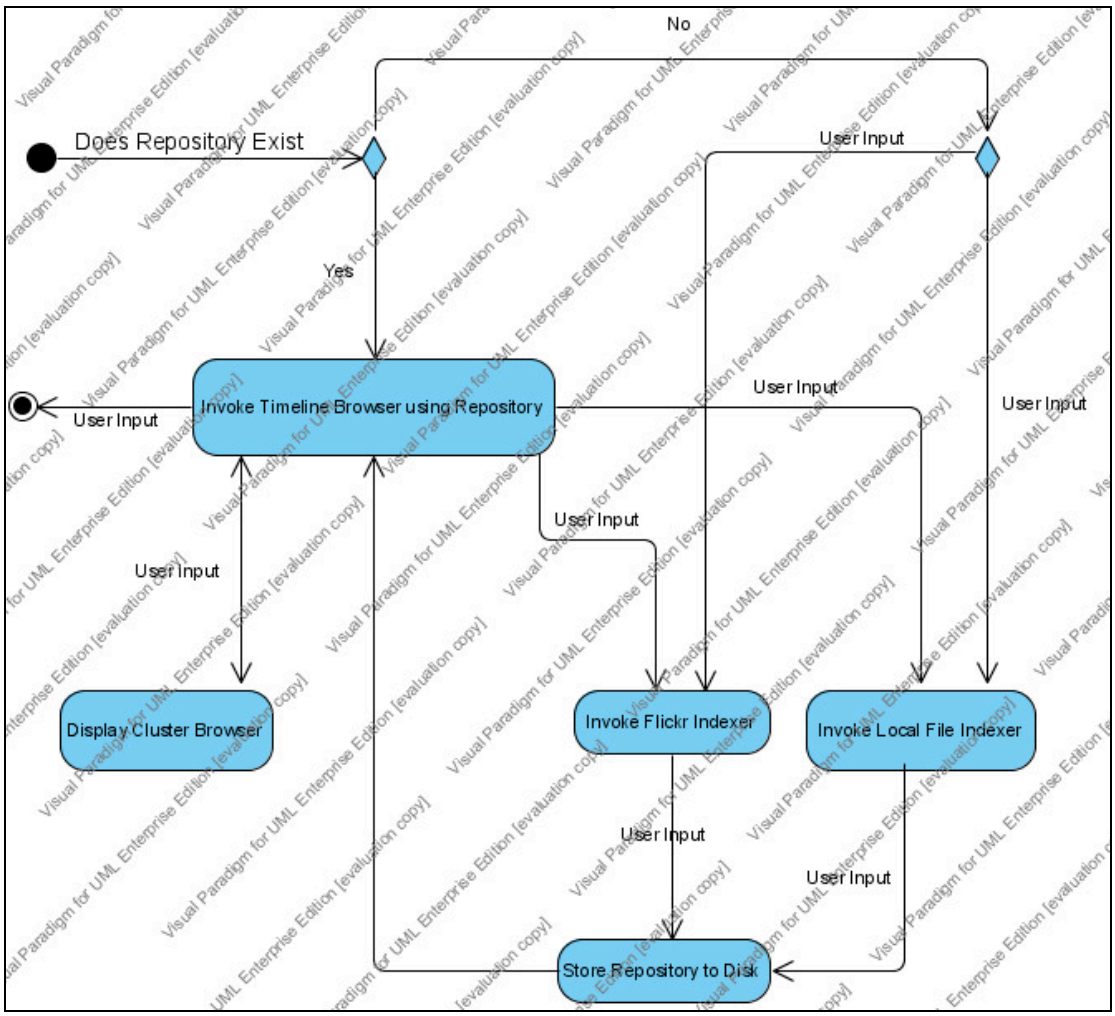

Figure 3. TimeIn Activity Diagram

Users initially invoke the Timeline Browser which can then in turn invoke the other two sub-applications. The relationship between these applications is illustrated in **Figure 3**.

5.2 Indexers

Before using the Timeline Browser sub-application, users need to import content that can be browsed. There are currently two mechanisms that can be used to import files into the browser.

5.2.1 Flickr Browser

Using the Flickr Browser, large sets of images can be imported from the Flickr[5] using the searches performed by the Flickr API[13] and browsed using TimeIn.

There are three types of searches currently implemented.

- **Search by Username** - In this mode, the Flickr Browser will fetch all images belonging to a user. User names are typically found in Flickr by looking at the url of an image, http://flickr.com/photos/username/imageId When looking at a user's homepage, his/her user name is typically displayed in the page title as username's photos.
- **Search by Tags** - In this mode, a comma separated list of Tags can be used fetch a set of images from Flickr. All images which contain each tag supplied will be retrieved.
- **Search by Group** - In this mode, the Flickr Browser will retrieve all images belonging to a Flickr Group, simply supply the group url. A directory of these can be found at http://flickr.com/groups/ An example of a valid group url is: http://flickr.com/groups/groupname/

Search Queries can be further filtered by limiting the number of images retrieved and setting a date range. As of this writing, the Flickr API will fetch a maximum of 5000 images per query

Upon fetching an image from Flickr, the Browser will annotate the image with a variety of metadata associated with the image including: tags, description and notes.

5.2.2 File Browser

Using the File Browser, local files can be used to populate the TimeIn Browser. Unlike the Flickr Browser, the File Browser does not need to overwrite existing repository data and can be used incrementally to index new content.

Upon fetching a file from the local filesystem, the Browser will annotate the file with automatically generated tags. For text documents (.txt, .pdf, .doc) these tags will be generated from the "relevant phrases" within the document. For image files these tags will be generated from the dominant colors of the image.

Figure 4. Timeline Overview

5.3. Timeline Browser

The Timeline Browser sub-application offers an alternative to the hierarchical file system by providing a timeline-based navigational interface with an emphasis on searching.

5.3.1 Timeline UI Overview

As illustrated in **Figure 4,** the basic components of the timeline are icons representing file or Flickr objects. Above each icon is the title or filename of the object represented, below is the timestamp.

If two or more icons would overlap in a current timeline view they are represented as a "compound object." Next to each object is a list of keyword tags associated.

Below the Timeline pane will initially be a histogram legend of all the files contained in the repository as well as a bar representing the current position and zoom level.

By pulling either end of the Ranged Slider, using the mouse scroll wheel, the scroll wheel, or hitting one of the zoom buttons, users can change the zoom level of the timeline. By dragging the Ranged Slider, users can pan the timeline.

The legend will be dynamically updated to display the position of the current timeline view within the repository.

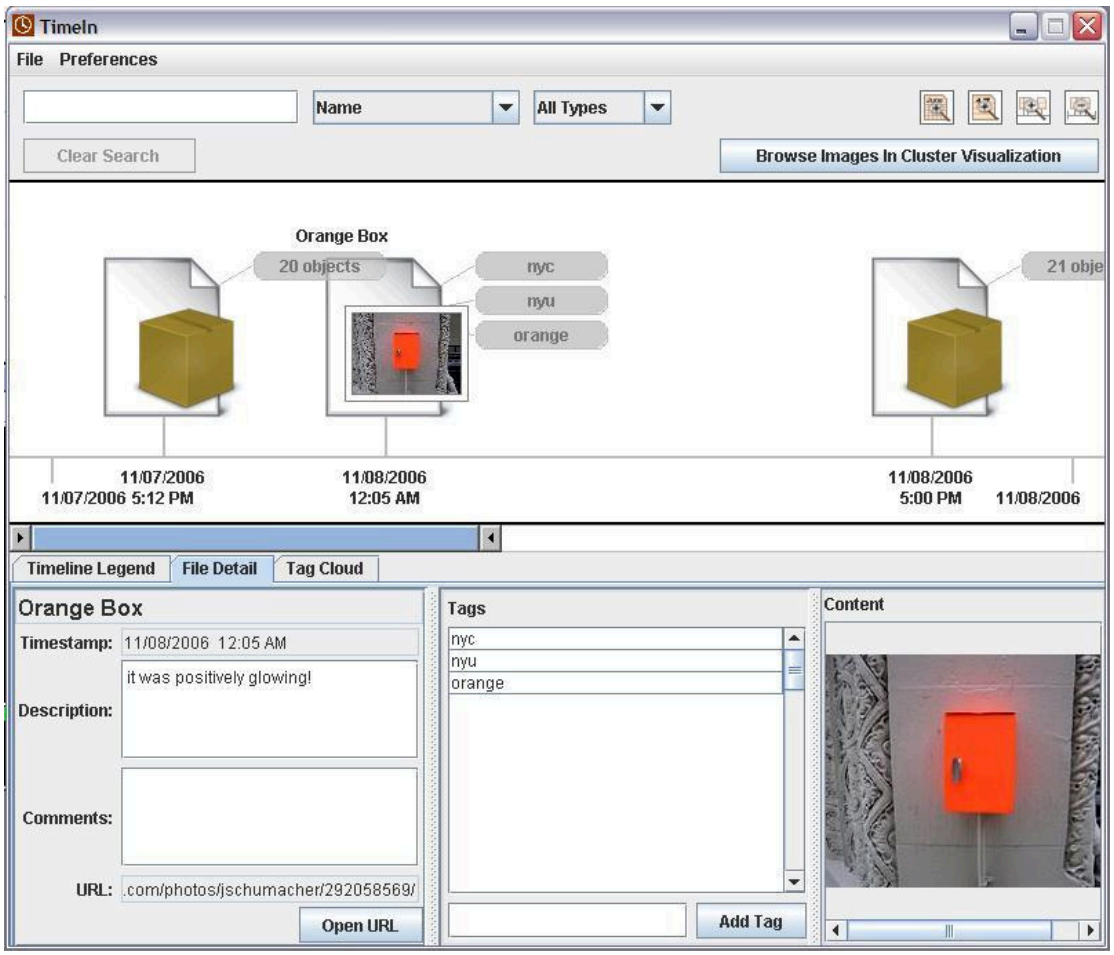

Figure 5. File Detail Panel

The timeline view can also be centered on an arbitrary file by left clicking on its icon. When an item is clicked the File Detail panel will be updated as illustrated in **Figure 5**.

If a file or Flickr object is selected, the metadata associated with the file will be displayed in the File Detail panel. If a compound object is selected, a list of the contained files will be displayed. Items in this list can also be selected and a subset of the associated metadata will be displayed.

Icons in the Timeline can also display a popup menu when right clicked. Using this popup menu, a user can change the zoom level of the timeline, invoke a content based search, or display all files contained within a compound object as displayed in **Figure 6**.

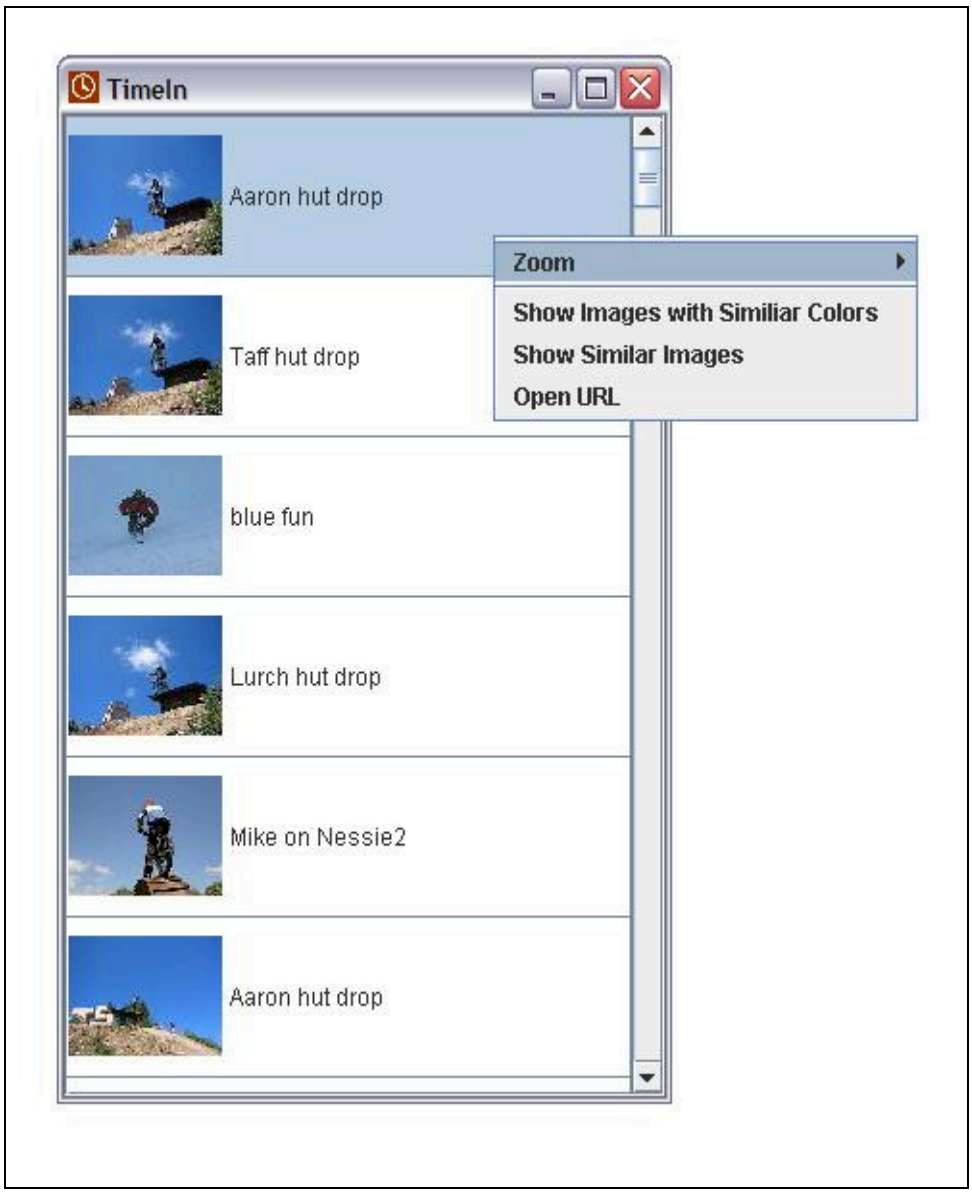

Figure 6. Compound Popup Dialog

A content based search will use the currently selected image as an example of images to search for. Currently, images that contain similar colors and overall similar images can be retrieved.

5.3.2 Search Filters

A variety of search filters can be used to limit the set of file objects displayed in the timeline.

Using the Search Field illustrated in **Figure 4** to input queries, users can search through a variety of text content associated with the files. These content fields are specified using the Search Type pull down menu and include, the title of files, the tags associated with files and text content of files or Flickr Descriptions. Please note that a variety of query types are supported including wildcards (e.g. *image* se?rch) and Boolean operators ((image and search) or flickr)

Types of content displayed in the timeline can be filtered using the Object Type pull down menu. By right clicking on an icon, a content based search can be invoked.

After invoking any kind of search filter, the clear search button can be used to reset the state of the timeline such that all content is displayed.

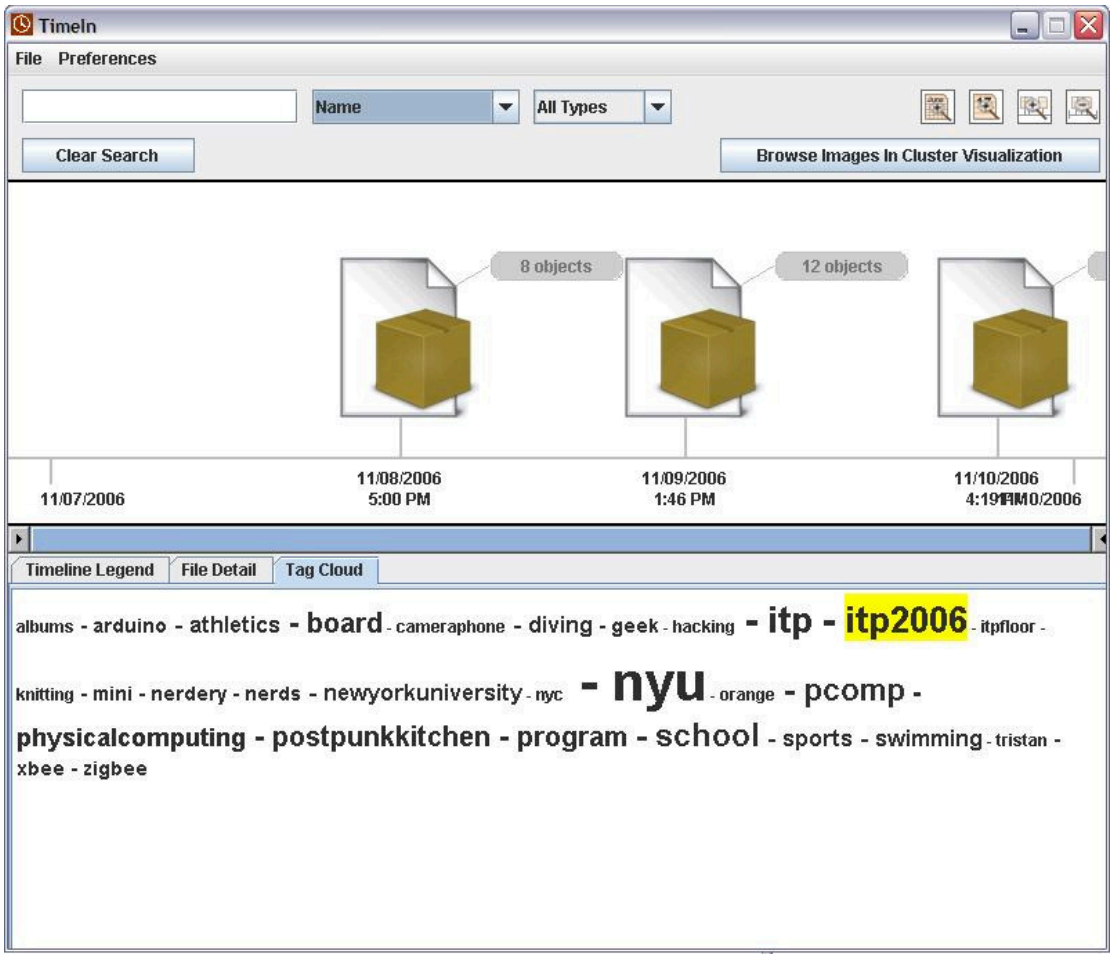

Figure 7. Tag Cloud Browser

5.3.3 Tag Cloud Search Control

As illustrated in **Figure 7**, users can browse the variety of tags contained within the repository using the Tag Cloud Search Control. More common tags are displayed in a larger font.

These tags can be selected by left clicking on any of them one time and deselected by left clicking a second time. Please note that several tags can be selected by simply left

clicking on each. When tags are selected, only objects containing all of the selected tags are displayed in the timeline.

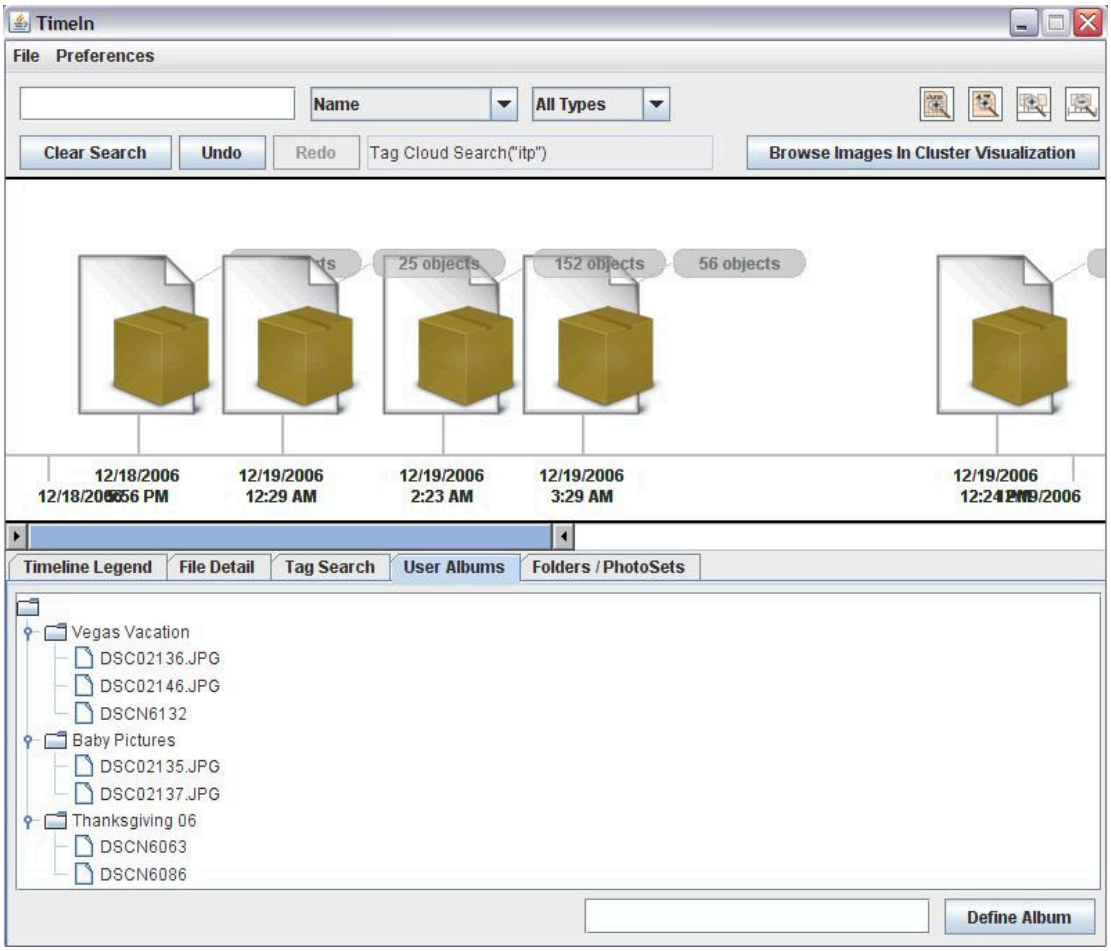

Figure 8. User Defined Albums

5.3.4 User Defined Albums Search Control

As illustrated in **Figure 9** users can define custom "albums" to help them organize objects. These albums are essentially sets of objects defined by the user. The album control provides a tree based mechanism to browse these albums and allows users to filter

the objects displayed in the timeline such that only objects belonging to the selected album are displayed

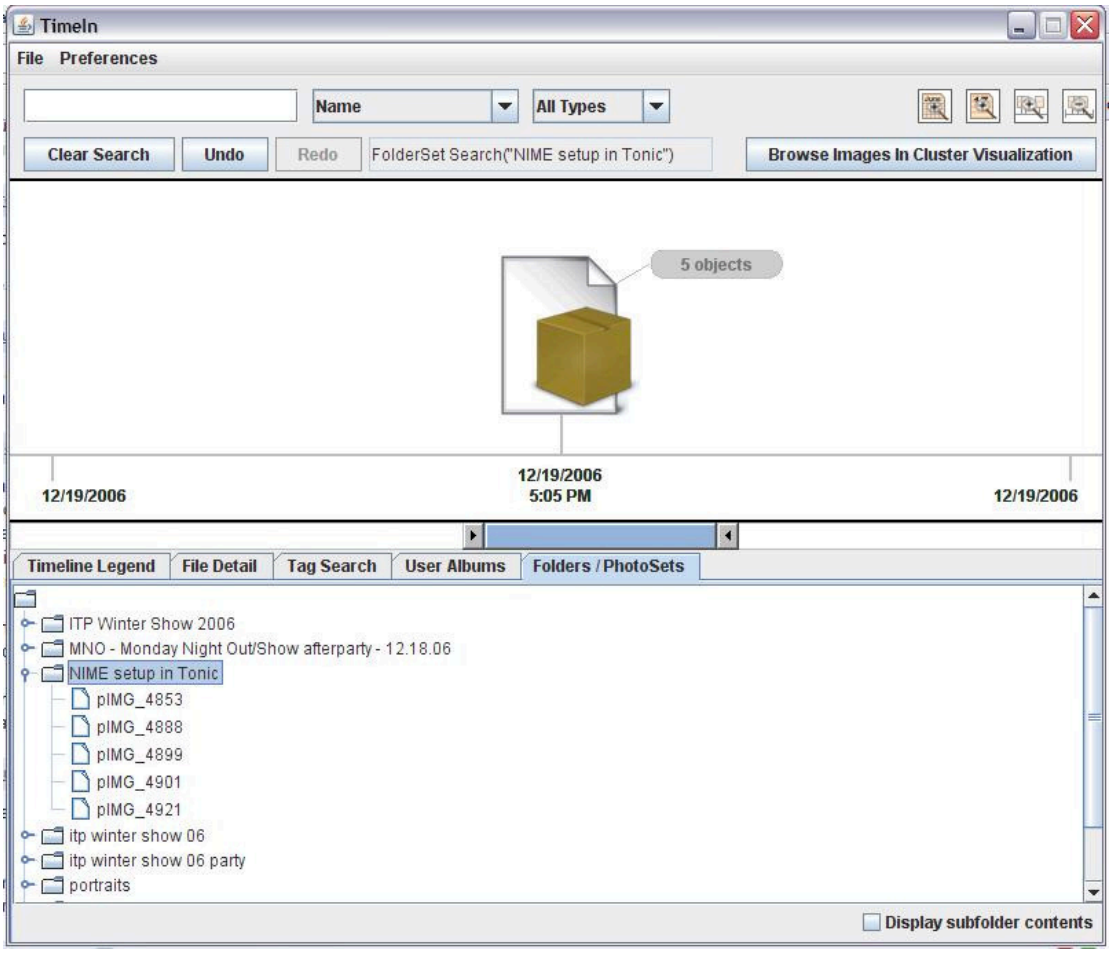

Figure 9. PhotoSet Browser

5.3.5 Flickr PhotoSets and File System Folders Search Control

As illustrated in **Figure 101**, users can browse objects in the repository using either the original file system structure for Local objects or the PhotoSets associated with Flickr objects. The PhotoSet control provides a tree based mechanism to browse these

structuresand allows users to filter the objects displayed in the timeline such that only objects belonging to the selected photoset or path are displayed

5.4 Cluster Browser

The Cluster Browser sub-application provides a quick and simple mechanism to browse a large set of images by organizing them into clusters of images containing similar content.

The Cluster Browser is invoked from the can be invoked from the Timeline Browser by hitting the Browse Images in Cluster Visualization button.

When the button is hit, all image objects present in the current timeline view are used to populate the cluster browser.

As illustrated in **Figure 11**, the clusters of images are displayed in a self organizing graph. To change the zoom level, move the cursor over a white portion of the display, hold the right mouse button and move the mouse up to zoom out and down to zoom in. Alternatively the scroll wheel can be used to change the zoom level

To pan the graph, move the cursor over a white portion of the display, hold the left mouse button and move the mouse.

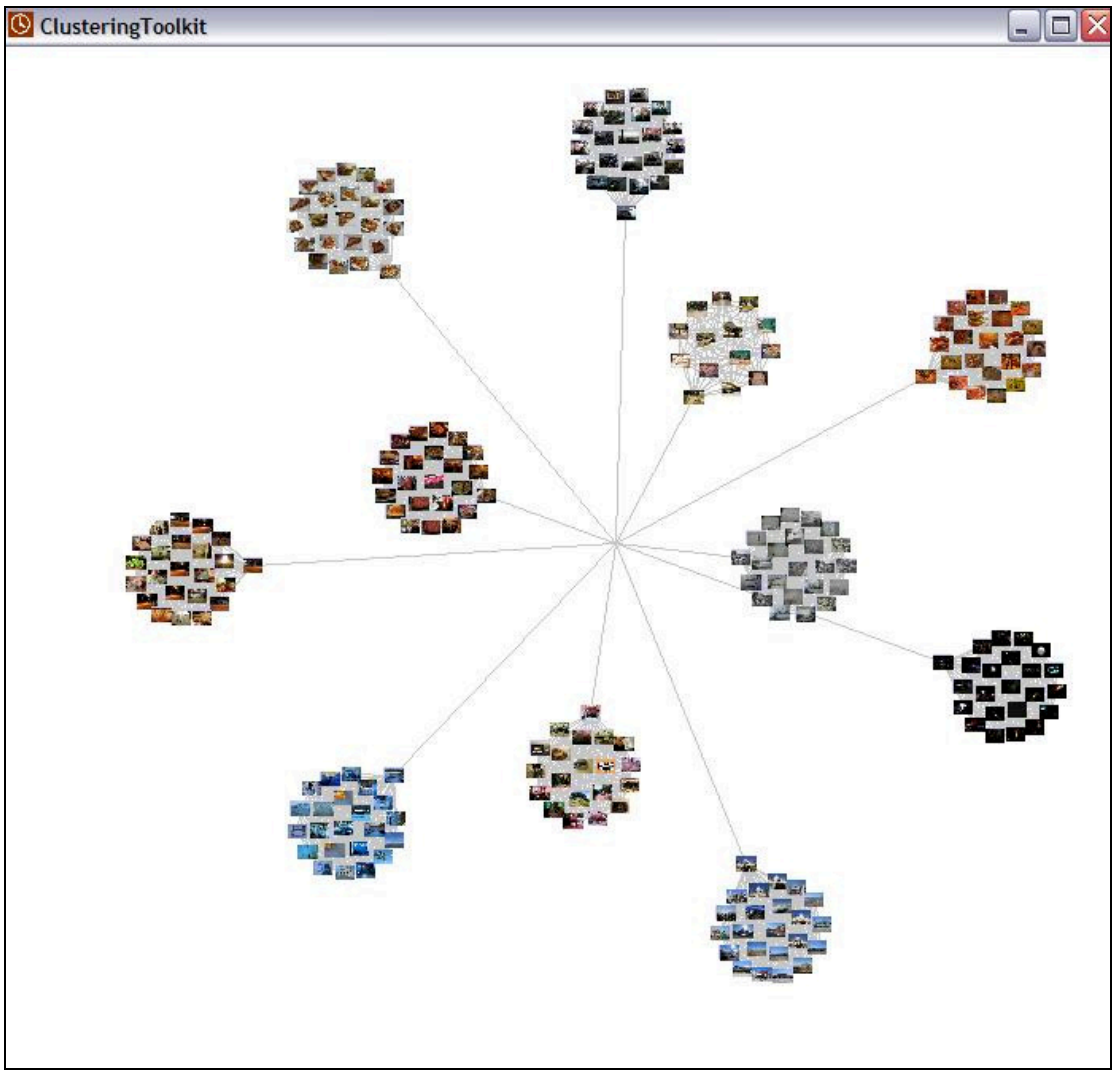

Figure 12. Cluster Browser

The clusters of images can be moved around on screen by moving the cursor over an image, holding the left mouse button and dragging.

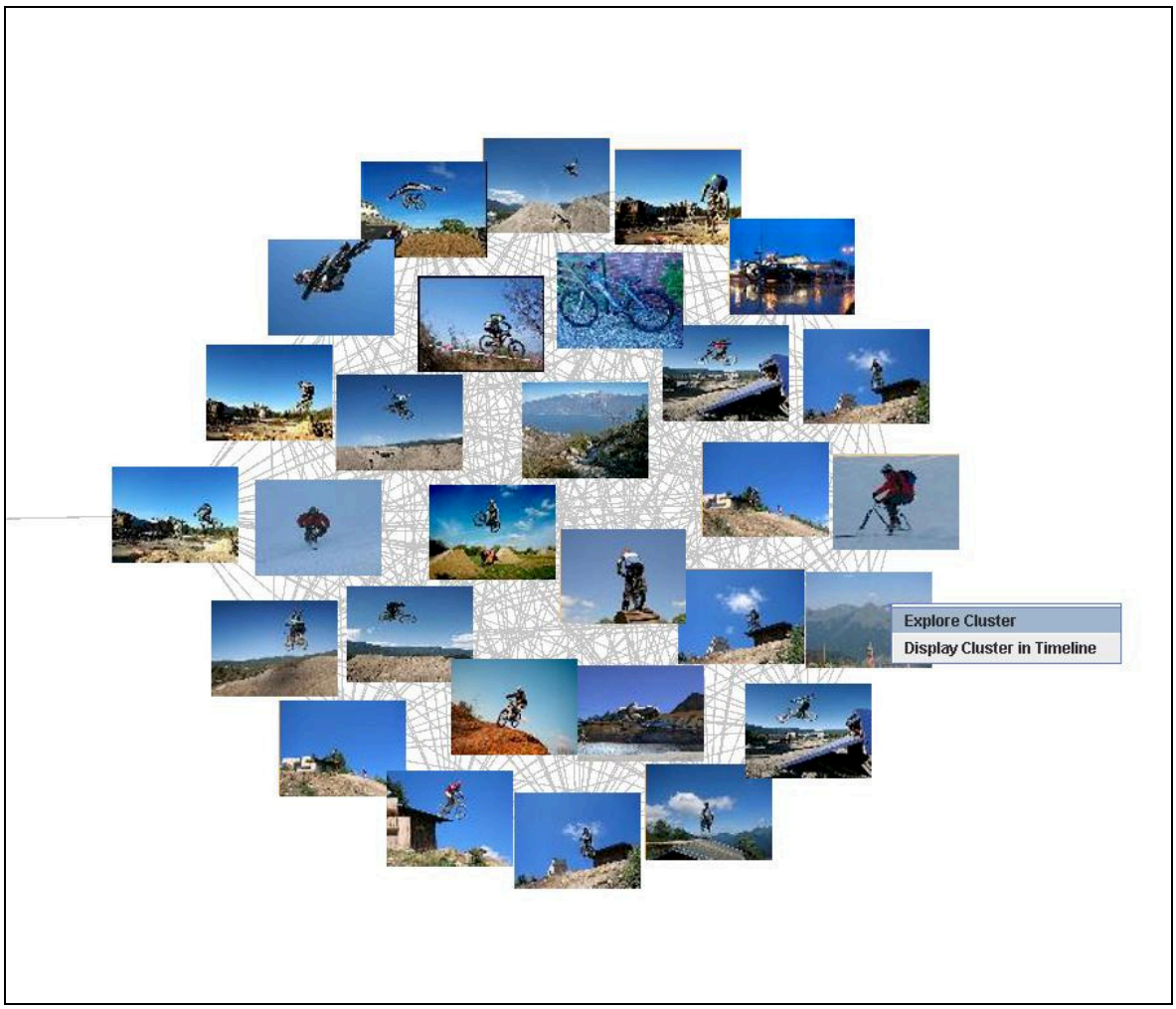

Figure 13. Image Cluster Detail

A popup menu can be displayed by right clicking on any item in a cluster as illustrated in **Figure 145**. Using this menu, all images contained within the cluster (including those not shown in the graph) can be browsed via a popup menu or all images in the cluster can be displayed in the timeline.

6. Major Algorithms

6.1 Keyword Extraction

In an attempt to provide automatic tagging of text content, we've developed a Keyword Extraction algorithm used by the File Indexer. This algorithm will take a set of text documents and for each document attempt to extract the 100 most "relevant" phrases and assign a "relevance" score to each. Phrase relevance is defined in this context to mean phrases that are qualitatively useful to in describing a document relative to the set of documents.

In developing these algorithms we initially investigated the set of phrases returned by traditional TF/IDF algorithms[6]. We found that while the algorithms did identify many phrases that could fit the "relevance" conditions, the algorithms also identified many phrases which did not fit the "relevance" conditions. Worse, we found that the "relevance" ranking often did not match the actual "relevance" of the phrases.

While these algorithms could certainly serve as a basis for the final algorithm there was a clear need for a more effective "filtering" mechanism to apply a more appropriate "relevance" score to each phrase extracted.

6.1.1 Ling Pipe[7], Information Extraction Package

In an attempt to implement this "filtering" mechanism, the Ling Pipe Information Extraction Package was used. Ling Pipe provides Hidden Markov Model[8],[9] based annotation system to annotate various features of documents.

Two annotation mechanisms were used in our algorithm, Part of Speech Annotation and Named Entity Phrase Annotation.

Part of Speech Annotation will tokenize a text corpus and assign a set Part Of Speech tag to each token (e.g. noun, verb, adjective) along with a relative confidence score between 0 and 1.

Named Entity Phrase Annotation will attempt to designate all "Named Entities" from a document and assign a relative confidence score between 0 and 1 to each. A named entity in this context is defined as a phrase that refers to a particular kind of object in the world. For example in the sentence "George Washington crossed the Delaware river." "George Washington" and "Delaware river" would be considered named entities.

Training the models used by these annotators proved to be a challenging issue. The algorithm would use untagged text corpora as well as a large variety of possible corpora. A variety of training sets were evaluated qualitatively. The Brown Model was selected to train the Part of Speech annotation mechanism while the English News: MUC-6 Model was used to train the Named Entity annotation mechanism. These models can be found at: http://www.alias-i.com/lingpipe/web/models.html

6.1.2 Keyword Extraction Algorithm

The first stage of the Keyword Extraction Algorithm used by the File Indexer will use the LingPipe package to annotate each text document in the corpus. The next stage will tokenize each document and assign a TF/IDF score to each token. The algorithm will then weight each token based on the Part of Speech annotation, tokens annotated as nouns are boosted by a weighting factor multiplied by the confidence score assigned by the annotator.

The algorithm will then add each phrase extracted by the Named Entity annotator to the score table. The base score of each multi-token phrase is defined as the average weighted TF/IDF score of each token in the phrase. The next step of the algorithm is to traverse the score table and boost each phrase found in the Named Entity phrase table by a weighting factor multiplied by the confidence score assigned by the annotator.

6.1.3 Evaluation

While a full empirical evaluation of this algorithm would prove quite difficult and beyond the scope of this project, the algorithm was qualitatively evaluated using the "20 Newsgroups" corpus of 20,000 documents[10]. This evaluation was used to tune the boost weights of the algorithm.

6.2 Distance Metric

To provide image clustering as well as the ability to query images by examples, we've developed a lightweight distance metric to compare two images. The algorithm

essentially scales two images to a three dimensional matrix of 9 x 20 x 20, this matrix can be described as 9 20x20 bitmaps representing the presence of one of 9 colors in particular pixel. Distance between two images is the sum of the XOR distance between the two sets of matrices.

A more detailed description of this algorithm can be found in [10.]

6.3 Clustering Algorithm

To implement the graph based cluster browser, we've developed a clustering algorithm based on the before mentioned distance metric. The algorithm will segment a set of images into clusters of "similar" images.

A variety of clustering techniques were evaluated in implementing this algorithm: K-Means[11], Partitioning Around Medoids[12] and Agglomerative Clustering[13]. Given the nature of the Distance Metric, PAM proved to the optimal mechanism in terms of silhouette score and performance.

Although our Distance Metric is fairly cost effective, the complexity of PAM grows quadratically with respect to the number of images. To mitigate this issue we evaluated the CLARA algorithm[14], an approximation algorithm based on PAM. While the PAM algorithm will use each image in a cluster in calculating the mediod, CLARA uses a sample set of the images selected randomly to calculate the medoid. Experimentation

demonstrated that CLARA performed virtually identically to PAM given a large enough sample set. The sample set size is used as a tunable parameter to the algorithm.

Since the algorithm has no prior knowledge of the corpus of images used as input, the optimal number of clusters to segment the images into is unknown. In an attempt to determine the optimal number of clusters to use, the clustering algorithm runs a number of times using a variety of number of clusters. Each iteration, the overall silhouette score of a clustering is calculated. The clustering with the highest silhouette score is presented to the user in the Cluster Browser.

The performance of this algorithm is determined primarily by the number of cluster size evaluations and the sample size used in the CLARA algorithm. Optimal values of these parameters were determined using a variety of sample set corpora. The algorithm as implemented delivers acceptable performance using image sets containing up to around 5000 images.

7. Software Architecture

7.1. Overview

The bulk of the communication between the three sub-applications described in section 3.1 occurs via a disk based repository implemented using the Apache Derby database project[15] and Apache Lucene search engine project[16]. The Timeline Browser and Cluster Browser also communicate via two Java Beans messages. When the Cluster Browser is invoked as a standalone application, this data is supplied via command line argument.

7.2. Repository Schema

7.2.1. Overview

All data created by the Indexers and user annotations made within the Timeline Browser are stored using the Derby Database. To provide better query flexibility and performance, portions of the database are also indexed via a Lucene search engine index.

7.2.2. Derby Tables

As illustrated in **Figure 17** there are essentially two sets of tables in the Derby Database.

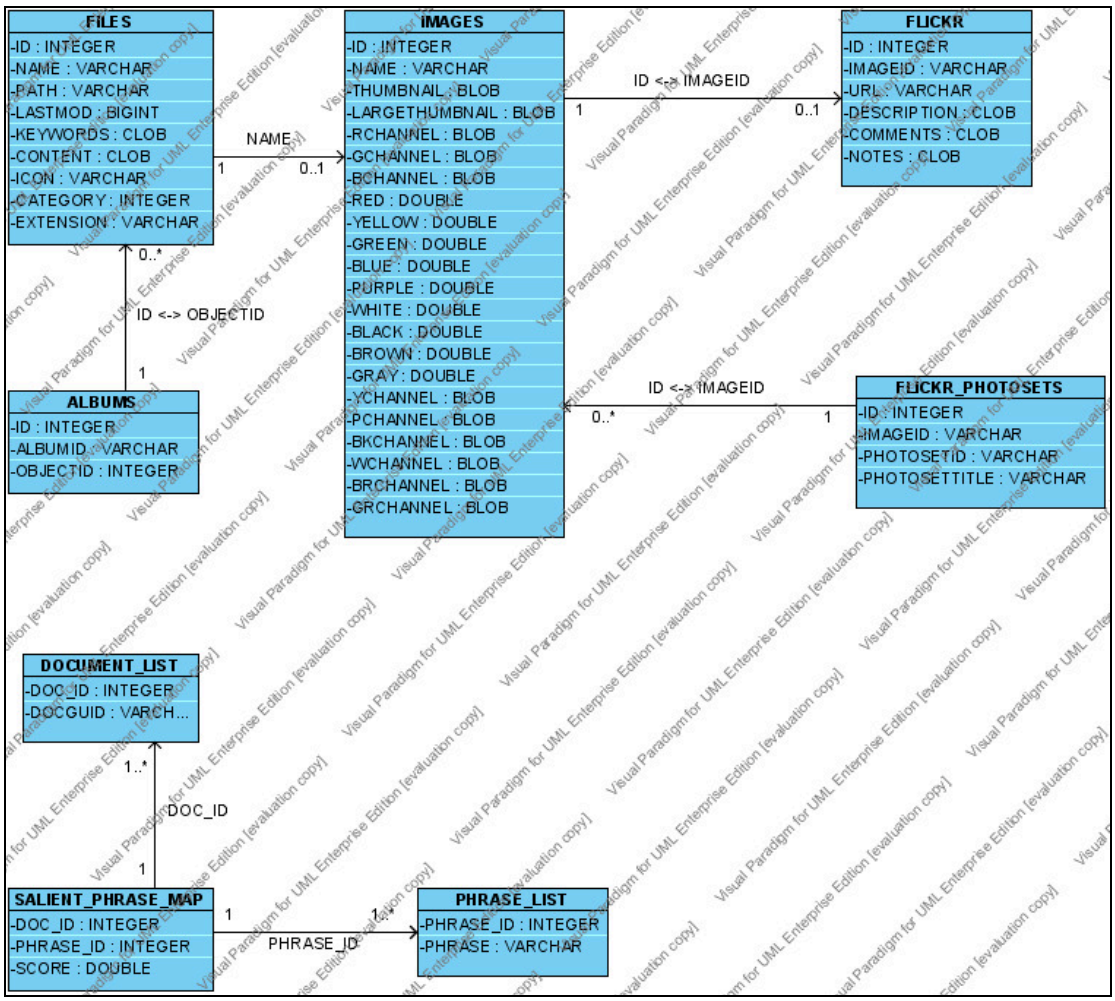

Figure 15. Derby Database Schema

The first three tables are used solely by the keyword extraction algorithm previously described. This algorithm is implemented as part of the File Indexer and the tables are not used by any other sub-application. These three tables essentially form a phrase lookup table for each document processed by the keyword extraction algorithm. These phrase lookup tables map the 100 highest scoring Salient Phrases extracted from a document with the score assigned by the first stage of the keyword extraction algorithm.

The remaining tables define the set of objects available to be displayed within the Timeline Browser and the Cluster Browser as well as the contents of the PhotoSets table as well as the user defined albums table.

There is a one-to-one correspondence between objects that can be displayed within the Timeline Browser and rows of the FILES table. This table defines metadata common to all timeline objects. Any Image or Flickr object will also have a corresponding row in the IMAGES table. This table contains thumbnails of each image as well as the data used by the previously defined Distance Metric between images. Any Flickr Object will also have a corresponding row in the FLICKR table which contains Flickr specific metadata.

User defined albums are defined in the ALBUMS table. Note that a row from the FILES table may be referenced by more than one album.

PhotoSets are defined in the FLICKR_PHOTOSETS table. Note that a row from the IMAGES table may be referenced by more than one PhotoSet.

7.2.3. Lucene Index

The Lucene index consists of a single table with a row corresponding to each object in the FILES table. The table schema is quite simple, there are four columns:

- **docId**, corresponds to the ID column in the FILES table in the Derby DB
- **contents**, if the corresponding row of the FILES table is referring to a text document, the contents field will contain the indexed text content of the document.
- **title**, corresponds to the name column of the FILES table
- **tags**, corresponds to the Keywords column of the FILES table.

The index serves two purposes. It is used by the keyword extraction algorithm to quickly identify each token within a document and calculate the TF/IDF score of a token.

The Lucene Index also serves as the basis to provide search results for queries made in the Search Pane of the Timeline Browser.

7.3. Indexer Architecture

7.3.1. Overview

As previously described, there are two indexer sub-applications that may be invoked, the File Indexer and the Flickr Indexer.

7.3.2. File Indexer Threads

The File Indexer works in three stages. First a Java Swing dialog is presented that allows the user to define a set of directories which should be indexed. When this set is defined a worker thread is spawned.

This thread will first define list of files contained within these directories. The thread will then iterate over the list of files. All PDF, Word and Text files will have text content extracted and this will be sent to the keyword extraction algorithm.

When this stage is completed the list of files will be iterated over once again. All files will have all relevant metadata extracted and stored in the Derby database and the Lucene search engine index. If keywords were extracted in the previous stage, they are used to annotate the file using the KEYWORDS column of the FILES table and the tags field of the Lucene index. Image files will have the data used by the previously described Distance Metric extracted and dominant colors used in the Image will be used to annotate the file using the KEYWORDS column of the FILES table and the tags field of the Lucene index.

A full UML Class Diagram of the File Indexer can be found at

http://cs.nyu.edu/~jnb263/TimeInUmlDiagrams/TimeInFileIndexerUML.png

The KronosphereIndexer class implements the worker thread invoked by the UI. The UI functionality is implemented using the classes in the "ui" package. The keyword extraction algorithm and the algorithms used to populate the Lucene Index are found in

the “textfilesearch” package. The last stage of the indexing process is managed by the FileIndexer class. The Derby Database is populated using the classes defined in the “data” package. This package is illustrated in greater detail in **Figure 18**. The “utility” package contains the classes which manage the connections to the Lucene and Derby Repositories as well as a variety of other low level tasks.

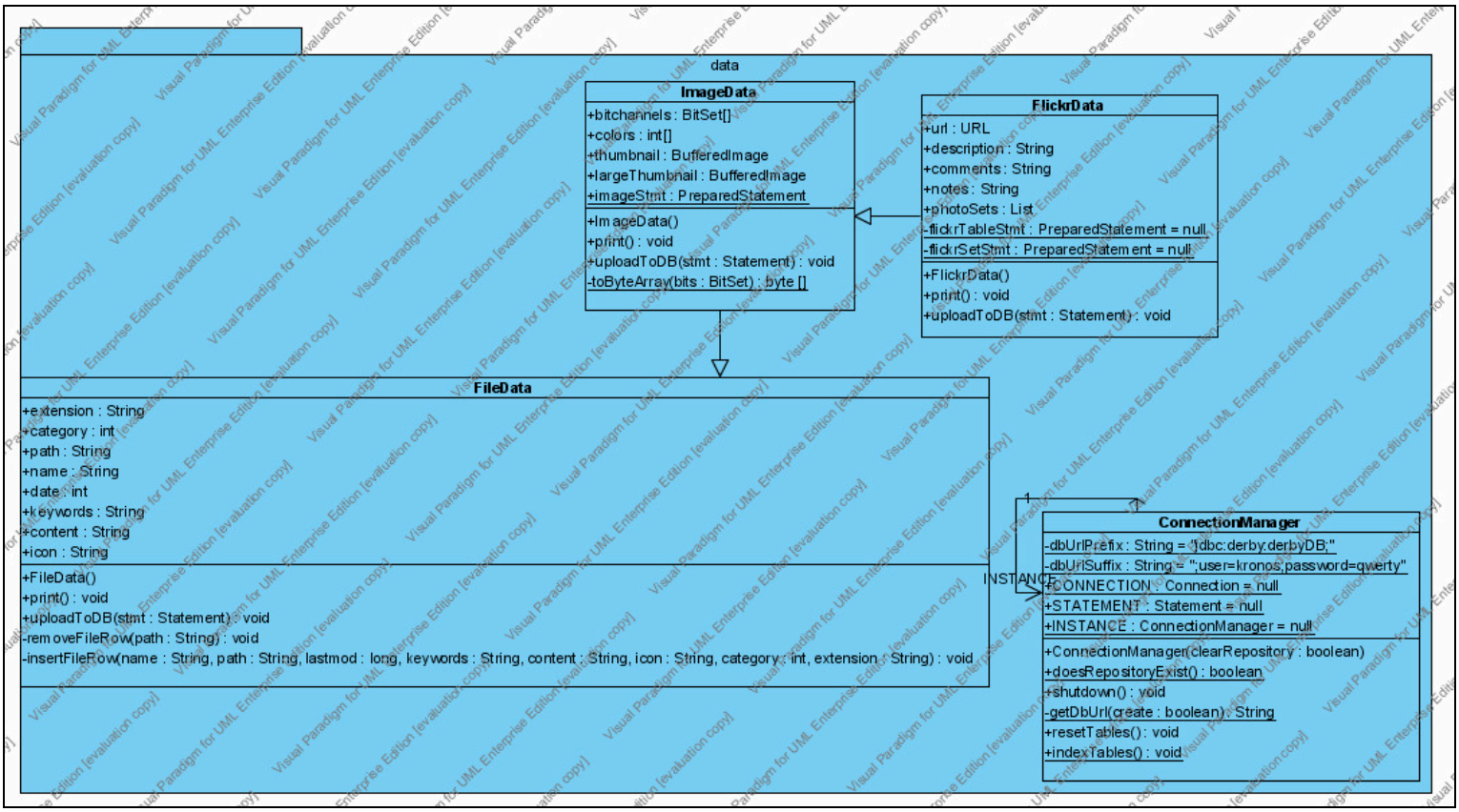

Figure 16. Indexer Data Class Heirarchy

7.3.3. Flickr Indexer

The Flickr Indexer works in two stages. First a Dialog is presented which allows the user to define a query. This query is then processed using the Jickr API[17], a java based wrapper for the Flickr API[18]. This Jickr API will perform the query and return an iterator that iterates over each image associated with the query. Each iteration will fetch all image and metadata associated with a single image file.

Since this process will block for a considerable amount of time while the data is fetched, the image fetching process was decoupled from the mechanism that writes data to the repository.

This repository writing mechanism is implemented in another single thread forked by the UI. The Jickr API thread will store each image pulled from Flickr into a queue. The repository writer thread will consume images from the queue and store all in the repository.

A full UML Class Diagram of the Flickr Indexer can be found at

http://cs.nyu.edu/~jnb263/TimeInUmlDiagrams/TimeInFlickrUML.png

Note that the Flickr Indexer shares much of the same codebase as the File Indexer. The "data" package is used to store data in the Derby Database, the "textfilesearch" package is used to store data in the Lucene Index and the "utilites" package is used to manage the connections to the repository.

The UI functionality of the Flickr Indexer is implemented in the FlickrUI class. The thread which pulls content from Flickr is implemented in the FlickrRepository class. The Producer-Consumer queue is implemented by the RepositoryWriterQueue class. The repository writer thread is implemented by the RepositoryWriter class.

7.4. Timeline Browser Architecture

7.4.1. Timeline Manager

The Timeline Manager serves as a point of entry for the Timeline Browser sub-application. As illustrated in the activity diagram shown in **Figure 3** of section 3.1, the Timeline Manager will first check to see if a Timeline Repository exists and can be loaded. If the repository cannot be loaded, the Timeline Browser sub-application will be closed and one of the two Indexers will be loaded.

Once the repository can be loaded, the Timeline manager will manage the lifecycle of the various components comprising the Timeline Browser sub-application. As with many Swing GUI applications, the major mechanisms implementing the Timeline Browser are implemented via a variety of singleton objects running in separate threads. These objects communicate via Java Beans Messages and call back methods when thread safety is not an issue.

The Timeline Manager is implemented entirely by the KronosphereFrontEnd class.

7.4.2. Components Overview

The functionality of the Timeline Browser is implemented primarily via the constituent GUI control components themselves. There are a few standalone services, but these can be construed as shared libraries used by a variety of components. Each control follows the Model-View-Controller pattern; the data for each UI control is managed primarily by the model role associated with the control, the activity is managed primarily by the controller role associated with the control and the presentation managed primarily by the view role associated with the control.

Note that while a number of the controls are too complex to have each MVC role managed by one class, the classes implementing each role are typically contained within the same package (or subpackages.)

As a general rule, classes implementing controller roles typically communicate only with other classes implementing controller roles of other controls or with the view or model roles of its own control. This communication typically occurs through Java Bean messages and in cases where thread safety is not an issue, callback methods of classes. Note that communications between the controller roles of components need not follow hierarchy; any component can communicate directly with any other component.

Timeline Browser UI consists of the Application window which contains three panels:

- Timeline Panel, contains the Timeline Control
- Search Panel, contains the Search Panel Control
- Tabbed Pane Browsers, contains the Histogram, Tag Search Cloud, User Albums and Folders/PhotoSet Controls.

7.4.3. Application Window Component

The controller is implemented primarily via the Timeline Manager which manages the lifecycle of the entire Browser sub-application. The Application Window itself processes a variety of Java Bean messages. The Application Window control contains no real model role. The View role is implemented entirely through the ApplicationWindow class

7.4.4. Timeline Control

The timeline control serves as the core of the Timeline browser sub-application. The primary functionality of other components of the Timeline Browser is to define and change the state of the Timeline control.

7.4.4.1 Controller Role

The Controller Role consists of three major components.

The **Data Reader** serves as the controller's coordinator and will define which objects are visible in the Timeline Control at any given moment.

The **Search Object Hierarchy** is used by the Data Reader to actually read data from the repository.

The **State Manager** defines the current interface state of the Timeline Control. An interface state serves to describe the current search criteria used by the Data Reader.

7.4.4.1.1 Data Reader

The Data Reader is implemented as a singleton class which forks a thread on construction. This thread will continually loop during the entire lifecycle of the Timeline Browser.

On each loop iteration, the Data Reader will check with State Manager to determine if the state of the control has changed. If it has, the Data Reader will invoke the Search Object specified by the State Manager and invoke a callback method of a class belonging to the View Role using the results.

7.4.4.1.2 State Manager

The State Manager will receive a message (or method callback) from any component of the Timeline Browser that wishes to modify the state of the Timeline Browser. These messages will be processed to define the current interface state. An interface state consists of an object belonging to the Search Object Hierarchy as well as common search criteria. Criteria common to all searches are the currently selected timeline object and the time range of the timeline.

7.4.4.1.3 Search Object Hierarchy

The Search Objects all extend the abstract DBSearch class. This class defines methods that will perform a search into the repository and return a set of objects belonging to the Model Role that are used by the View Role to define the set of objects visible in the Timeline Control.

A detailed UML Class diagram of this hierarchy of objects is shown in **Figure 19.**

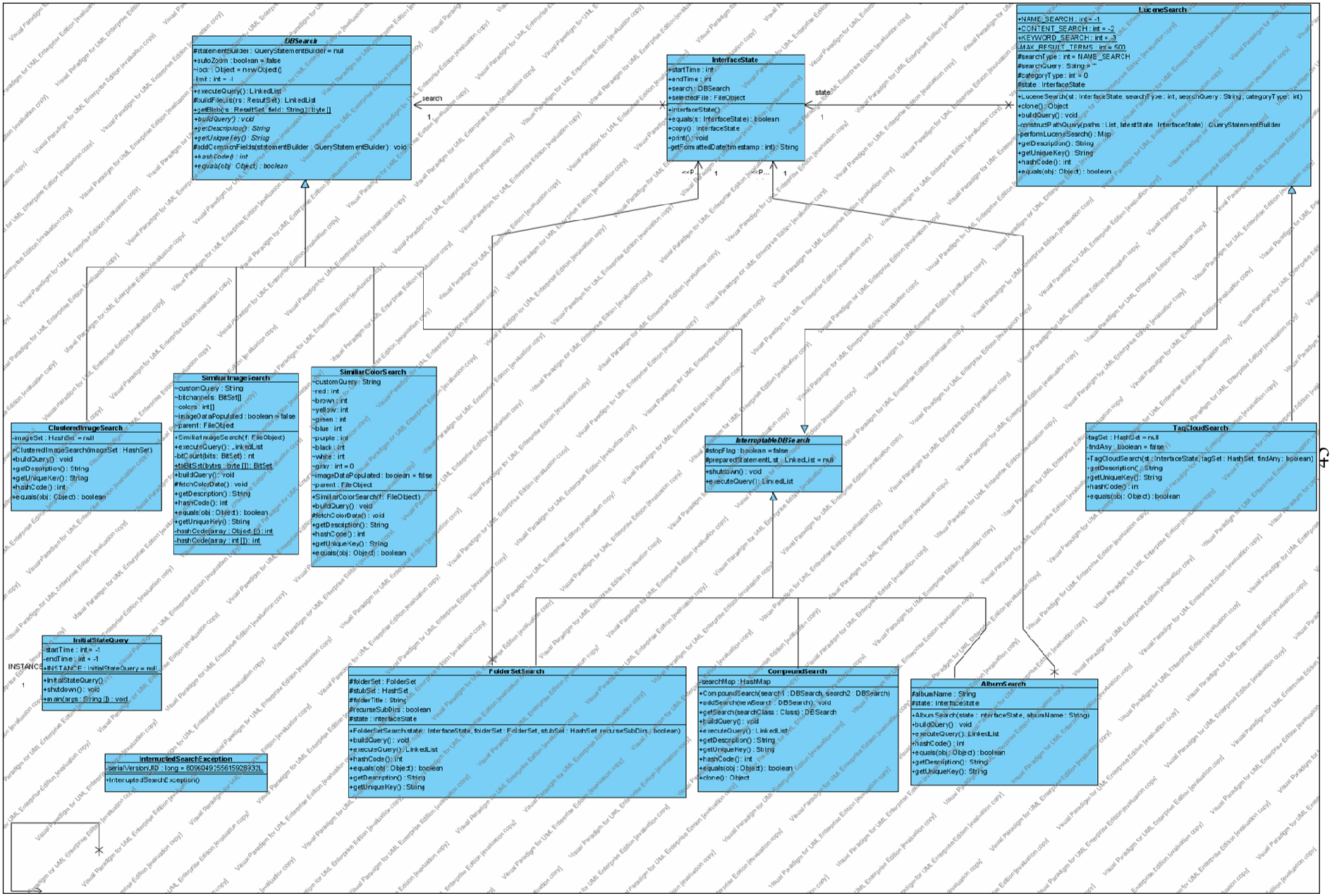
DBSearch
#statementBuilder : QueryStatementBuilder = nul
+autoZoom : boolean = false
-lock : Object = new Object()
-limit : int = -1
+executeQuery() : LinkedList
#buildFileLists(rs : ResultSet) : LinkedList
+getBlob(rs : ResultSet, field : String) : byte []
+buildQuery() : void
+getDescription() : String
+getUniqueKey() : String
#addCommonFields(statementBuilder : QueryStatementBuilder) : void
+hashCode() : int
+equals(obj : Object) : boolean
InterfaceState
+startTime : int
+endTime : int
+search : DBSearch
+selectedFile : FileObject
+InterfaceState()
+equals(s : InterfaceState) : boolean
+copy() : InterfaceState
+print() : void
-getFormattedDate(timestamp : int) : String
search
state
LuceneSearch
+NAME_SEARCH : int = -1
+CONTENT_SEARCH : int = -2
+KEYWORD_SEARCH : int = -3
-MAX_RESULT_TERMS : int = 500
#searchType : int = NAME_SEARCH
#searchQuery : String = ""
#categoryType : int = 0
#state : InterfaceState
+LuceneSearch(st : InterfaceState, searchType : int, searchQuery : String, categoryType : int)
+clone() : Object
+buildQuery() : void
-constructPathQuery(paths : List, latestState : InterfaceState) : QueryStatementBuilder
-performLuceneSearch() : Map
+getDescription() : String
+getUniqueKey() : String
+hashCode() : int
+equals(obj : Object) : boolean
ClusteredImageSearch
-imageSet : HashSet = null
+ClusteredImageSearch(imageSet : HashSet)
+buildQuery() : void
+getDescription() : String
+getUniqueKey() : String
+hashCode() : int
+equals(obj : Object) : boolean
SimilarImageSearch
~customQuery : String
~bitchannels: BitSet[]
~colors : int[]
~imageDataPopulated : boolean = false
~parent : FileObject
+SimilarImageSearch(f : FileObject)
+executeQuery() : LinkedList
-bitCount(bits : BitSet) : nt
-toBitSet(bytes : byte []) : BitSet
+buildQuery() : void
#fetchColorData() : void
+getDescription() : String
+hashCode() : int
+equals(obj : Object) : boolean
+getUniqueKey() : String
-hashCode(array : Object []) : int
-hashCode(array : int []) : int
SimilarColorSearch
~customQuery : String
~red : int
~brown : int
~yellow : int
~green : int
~blue : int
~purple : int
~black : int
~white : int
~gray : int = 0
~imageDataPopulated : boolean = false
~parent : FileObject
+SimilarColorSearch(f : FileObject)
+buildQuery() : void
#fetchColorData() : void
+getDescription() : String
+hashCode() : int
+getUniqueKey() : String
+equals(obj : Object) : boolean
InterruptableDBSearch
#stopFlag : boolean = false
#preparedStatementList : LinkedList = null
+shutdown() : void
+executeQuery() : LinkedList
TagCloudSearch
-tagSet : HashSet = null
-findAny : boolean = false
+TagCloudSearch(st : InterfaceState, tagSet : HashSet, findAny : boolean)
+getDescription() : String
+getUniqueKey() : String
+hashCode() : int
+equals(obj : Object) : boolean
InitialStateQuery
-startTime : int = -1
-endTime : int = -1
+INSTANCE : InitialStateQuery = null
+InitialStateQuery()
+shutdown() : void
+main(args : String []) : void
INSTANCE
1
InterruptedSearchException
-serialVersionUID : long = 8096049755615928332L
+InterruptedSearchException()
FolderSetSearch
#folderSet : FolderSet
#stubSet : HashSet
#folderTitle : String
#recurseSubDirs : boolean
#state : InterfaceState
+FolderSetSearch(state : InterfaceState, folderSet : FolderSet, stubSet : HashSet, recurseSubDirs : boolean)
+buildQuery() : void
+executeQuery() : LinkedList
+hashCode() : int
+equals(obj : Object) : boolean
+getDescription() : String
+getUniqueKey() : String
CompoundSearch
-searchMap : HashMap
+CompoundSearch(search1 : DBSearch, search2 : DBSearch)
+addSearch(newSearch : DBSearch) : void
+getSearch(searchClass : Class) : DBSearch
+buildQuery() : void
+executeQuery() : LinkedList
+getDescription() : String
+getUniqueKey() : String
+hashCode() : int
+equals(obj : Object) : boolean
+clone() : Object
AlbumSearch
#albumName : String
#state : InterfaceState
+AlbumSearch(state : InterfaceState, albumName : String)
+buildQuery() : void
+executeQuery() : LinkedList
+hashCode() : int
+equals(obj : Object) : boolean
+getDescription() : String
+getUniqueKey() : String

Figure 17. Search Object Heirarchy

Classes implementing DBSearch are described in the section below.

7.4.4.1.3.1 Lucene Search

This is the most basic search mechanism and the one initially used when the Timeline Browser sub-application is invoked. This search is also invoked by the Search Pane Component when a query is entered.

If a search query string is supplied as an argument, the query will be parsed and be used to invoke a Lucene search. Only documents returned from the Lucene Search are eligible to be returned by this Search Object.

Users can select to perform the query search on one of the three search fields defined in the Lucene Index, Object Title, Tag or Contents/Flickr Description.

A subset of a full regular expression query language has been implemented for use in queries. The grammar is illustrated in **Figure 20**.

```
<SimpleQuery> ::= <Term> ( "OR" <Term> )*
<Term> ::= <Factor> (("AND")? <Factor>)* | "\"" (<Factor>)+ "\""
<Factor> ::= "(" <SimpleQuery> ")" | <WildcardSTR>
<WildcardSTR> ::= <REGEX> | <SIMPLE_STRING>
<REGEX> ::= (<STR>)((<REGEX_DELIM>)+(<STR>)?)+ | ((<REGEX_DELIM>)+(<STR>)?)+ (<STR>)
<STR> ::= (<CHAR>)+
<CHAR> ::= ~[" ", "\t", "\n", "\r", "(", ")", "\"", "*", ".", "+"]
<REGEX_DELIM> ::= ["*", "?"]
```

Figure 18. Query Language Grammar

Note that if the last term in a query is a <STR> term, the Lucene Query processing mechanism will essentially append a "*" operator to it.

Users can specify the type of Timeline Object they wish to view via a selection in the Search Panel Control. Only Objects of the type specified are eligible to be returned by this Search Object.

7.4.4.1.3.2 Clustered Image Search

Invoked by the Cluster Browser sub-application. Returns the set of objects belonging to a specified Cluster.

7.4.4.1.3.3 User Album Search

Invoked by the User Album component. Returns the set of objects belonging to a specified UserAlbum.

7.4.4.1.3.4 Similar Image Search

Invoked by the Timeline Control View Role. Returns mapping of images to the Distance Metric between each image and a selected image object.

7.4.4.1.3.5 Similar Color Search

Invoked by the Timeline Control View Role. Returns a ranked set of images who contain similar colors to a selected image object. Uses a variant of the Image Distance Metric. Described in more detail in [19]

7.4.4.1.3.6 Tag Cloud Search

Invoked by the Tag Cloud component. Returns the set of objects containing a specified set of tags. Can find objects containing all or any of the tags.

7.4.4.1.3.7 PhotoSet Search

Invoked by the PhotoSet component. Returns the set of objects belonging to a specified PhotoSet/FlickrSet.

7.4.4.1.3.8 Compound Search

The Compound Search serves as an aggregate of instances of the other search types. This search will perform each member search and take the intersection of the sets of objects returned from each. Note that only one instance of any given type of search can exist in the Compound Search, existing search instances will be overwritten.

7.4.4.2 Model Role

The Model Role of the Timeline Control is implemented via the FileStub class. This class contains essentially all of the data present in a row of the FILES table in the Derby Database as well as the small thumbnail of an Image object.

7.4.4.3 View Role

The View Role of the Timeline Control is implemented primarily in the custom Swing Control defined in the Timeline class. A variety of other mechanisms have been implemented to support this control.

7.4.4.3.1 Timeline Custom Swing Control

The Timeline Control itself consists of a Swing Panel with the paint() method overridden. This panel serves as an area in which Timeline Objects which are extensions of the Swing JComponent class are displayed.

7.4.4.3.2 Timeline Objects

The timeline objects are essentially wrapper objects for the JComponent objects displayed in the Timeline Control as well as the metadata associated with each object. These objects are implemented in two class hierarchies: The Object and Graphics Hierarchies. UML Class diagrams illustrating both hierarchies are illustrated in the diagrams below.

The classes in the Graphics hierarchy essentially define the JComponent that will be displayed in the timeline and all of its sub components including labels and thumbnails.

The classes in the Object hierarchy serve as wrappers for the Graphcs objects, manage the layout of the Graphics objects in the Timeline Control Panel and contain all of the associated metadata.

As illustrated in **Figure 21** there is a type of Object associated with each possible type of Timeline File object. There is a one to one correspondence between Object instances and FileStub instances in all but one case, the CompoundObject.

The CompoundObject contains a list of FileStub objects and is constructed when there are more than one Timeline Objects that would overlap in a layout of objects in the Timeline Control Panel.

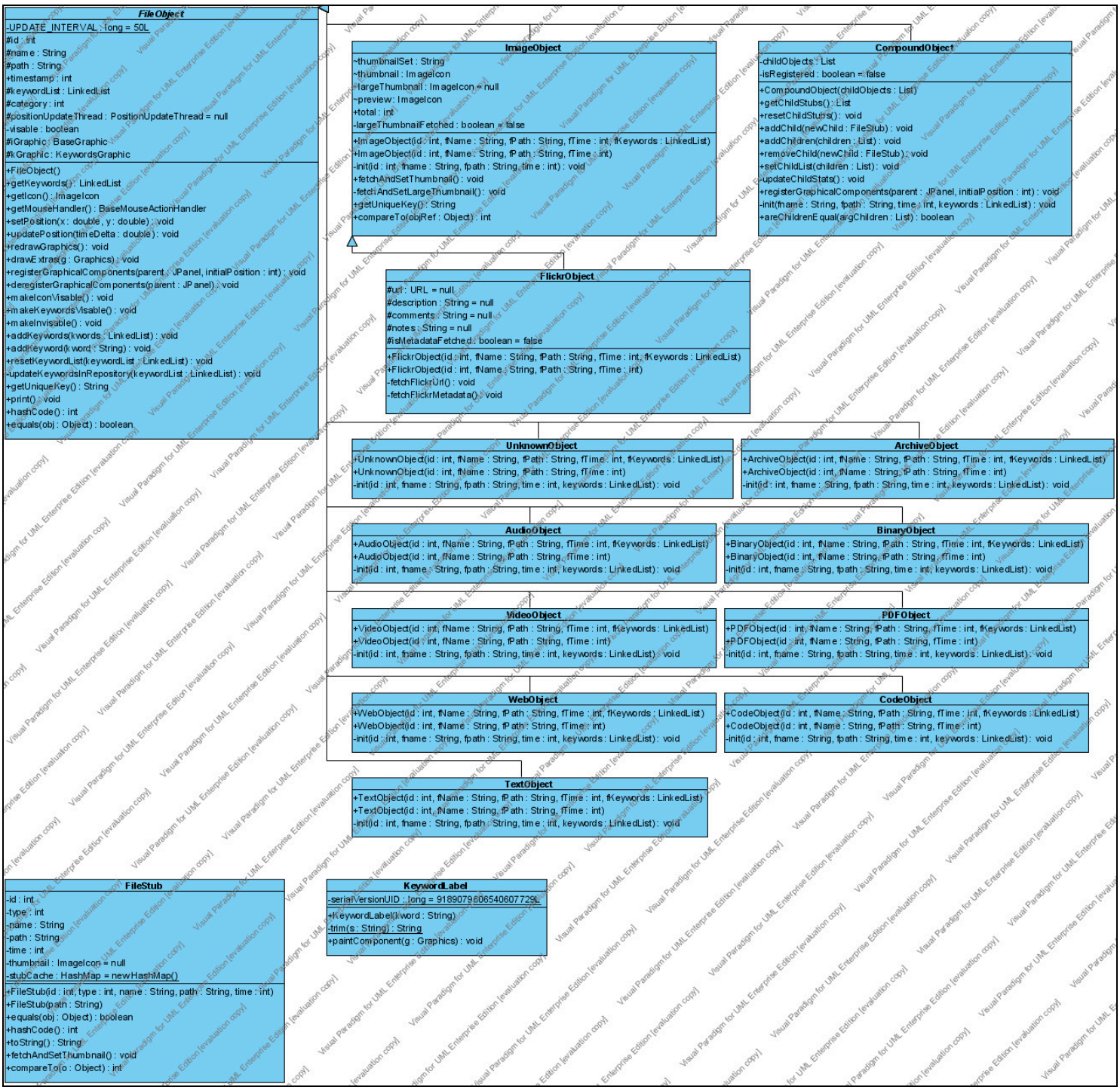

Figure 19. Timeline Object Hierarchy

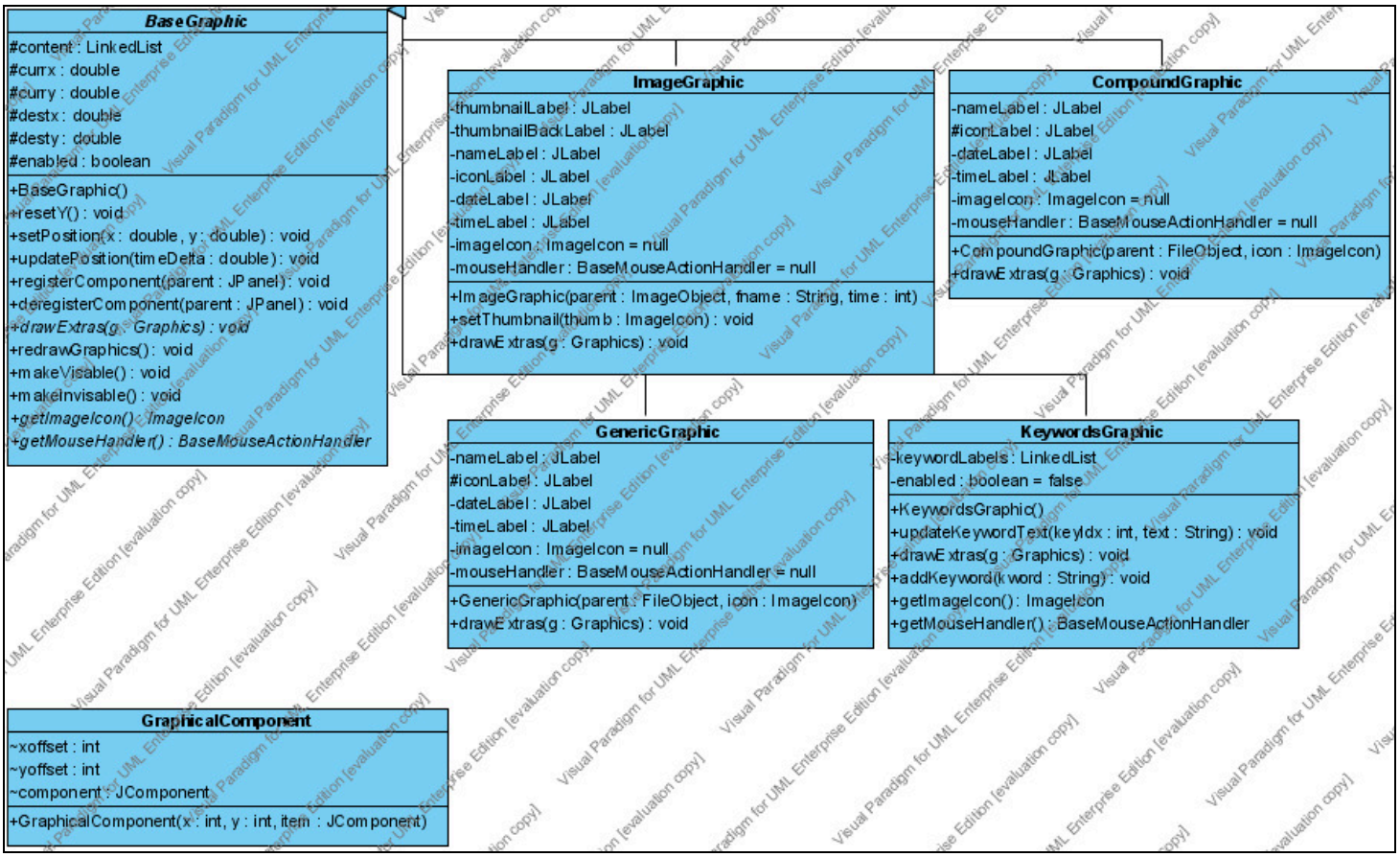

Figure 20. Timeline Graphics Object Hierarchy

7.4.4.3.3 Timeline Object Coordinator

The Timeline Object Coordinator serves as a bridge between the Controller Role and the View Role. FileStubs returned by a search object are sent to the Timeline Object Coordinator to be displayed as Timeline Objects in the Timeline Control Panel.

The Timeline Object Coordinator will determine if Objects corresponding to the each FileStub sent will overlap other Objects corresponding to other FileStubs. FileStubs that will not overlap other Objects will be used to construct a new Object and displayed in the Timeline Control Panel. All FileStubs that will overlap other objects will be used to construct CompoundObjects that are displayed in the Timeline Control Panel.

7.4.4.3.4 Object Mouse Controls

A variety of MouseEvent Handlers used by the Timeline Control Panel are implemented in the actions package. These Event Handlers will allow the user to change the selected file in the timeline as well as invoke a variety of searches by sending messages to the Timeline State Manager

7.4.5 Search Pane Control

The Model, View and Controller Roles of the Search Pane Control are all implemented in the SearchPane class. This control essentially provides a mechanism for users to specify search queries that are sent to the Timeline Control.

7.4.6 User Album Control

The Model Role of the User Album Control is implemented in the AlbumModel class. A singleton instance of this class is instantiated by the Front End Manager populated by a single query into the Derby database. If additional albums are added to the model, they are serialized when the model is shutdown by the Front End Manager. The View and Controller Roles are implemented by the ObjectAlbumPanel class. This control essentially provides a mechanism for users to browse and specify search queries based on user defined albums that are sent to the Timeline Control.

7.4.7 FolderSet Control

The Model Role of the FolderSet Control is implemented in the FolderSetBrowserModel class. A singleton instance of this class is instantiated by the Front End Manager populated by a single query into the Derby database. The View and Controller Roles are implemented by the FolderSetBrowser class. This control essentially provides a mechanism for users to browse and specify search queries based on filesystem folders or Flickr Albums that are sent to the Timeline Control.

7.4.8 Tag Cloud Control

The Model Role of the Tag Cloud Control is implemented in the TagCloudDataReader class. A singleton instance of this class is instantiated by the Front End Manager populated by a single query into the Derby database. The View and Controller Roles are implemented by the TagCloudBrowserPanel class. This control essentially provides a mechanism for users to browse and specify search queries based on object tags that are sent to the Timeline Control.

7.4.9 Histogram Control

The Model Role of the Tag Cloud Control is implemented in the HistogramModel class. A singleton instance of this class is instantiated by the Front End Manager populated by a single query into the Derby database. The View and Controller Roles are implemented by the DbHistogram class. This control essentially provides a graphical overview of the content available to be displayed in the timeline.

7.5 Cluster Browser Architecture

The Cluster Browser sub-application will take as input a set of file id strings that reference rows in the FILES table of the Derby database. This set of strings can either be passed as a method call when the sub-application is invoked via the Timeline Browser sub-application or as a command line argument to the application when run standalone.

The set of file ids are used to define a set of DbImage objects. These objects implement the previously mentioned Image Distance Metric algorithm.

This set of DbImage objects are then used by the previously mentioned Image Clustering Algorithm to define a data structure which represents clusters of DbImage objects. Each DbImage is associated with the silhouette score of the object in the cluster.

The cluster display is implemented via an instance of a force-directed Prefuse Graph[20]. To display the data structure of DbImage objects in the prefuse graph, all image thumbnails corresponding to a DbImage object must be serialized to disk.

Next a String object composed of an XML Graph conforming to the Prefuse schema must be constructed. Since the graph will become muddled if too many nodes are displayed, only a percentage of the highest ranked DbImages in each cluster are displayed in the graph. For each DbImage object eligible to be displayed in the graph, a node is constructed which references the corresponding thumbnail file. An edge is drawn between each node in the cluster and all other nodes in the same cluster.

A dummy "hub" node is also constructed and an edge is drawn between this node and a single node from each cluster.

8. User Studies

To test the usability of the system, five users were provided with TimeIn, an online help page[21] and a set of tasks to complete. The actual questionnaire is shown in Appendix A.

The sample tasks were as follow:

- Given a Flickr user name and a vague textual description, import all the images belonging to the Flickr user and find the annotated image.
- Find a similarly described image using the Flickr site alone.
- Create a few albums containing a dozen "related" images.
- Find a dozen images of "two person portraits"

After reading the instruction page, users first needed to import the set of images from Flickr. Most users found the Flickr Indexer to be intuitive and the task was completed without issue.

To complete the first task users utilized the Tag Cloud browser and the search query field. Users found the interface to be "intuitive and natural" taking only a few seconds to figure out.

The first task of finding a vaguely described image was completed by all users in less than 2 to 3 minutes. Once the set of images were imported from Flickr, users were able to complete this task significantly more quickly than using the Flickr site alone. Users described the experience of finding the image in TimeIn to be more straightforward than using the Flickr site alone.

Album creation was found to be fairly straightforward by most users. The task was completed in less than 5 minutes. While users found the interface to be useful, comments included: "it would be nice to add batches of images" and "there should be a mechanism to export albums to either the file system or Flickr."

In attempting to complete the final task, it was not immediately apparent for some users that the cluster browser would be useful. The cluster browser, once discovered, was found to be impressive by most users. They found the interface to be novel and were able to find sets of "two person portraits" within a matter of minutes.

While most users found the system to be intuitive, a few users found the timeline slider bar to be a bit tricky. Users preferred to use the mouse wheel to change the zoom level of the timeline.

A few users commented that the automatic annotation of images was not particularly useful. The sentiment was that the cluster browser is more useful for navigating images that have not been user annotated than the automatically generated tags.

9. Conclusion and Future Work

In this paper we've presented TimeIn, a system implementing variety of novel approaches for browsing and organizing large sets of file content across content. While our limited user studies have shown the system to be useful and promising, we still consider the application to be in beta testing phase. We plan on releasing the system to a wide audience over the next month and refining the interface based on further user input.

Based on user comments a variety of additional features are currently in development. The biggest demand seems to be the ability to import data from additional web based hosting services. Some candidates include Myspace.com and Picasa Web Albums.

Considering the positive response to the cluster browser, we also plan on adding support for text objects as well as expanding the types of relationships used to cluster items.

There have also been requests to add features allowing users to use TimeIn to export organized content to either a local file system or web based site.

10. References

[1] **Google Desktop Search**, http://desktop.google.com/

[2] **Apple - Mac OS X – Spotlight**, http://www.apple.com/macosx/features/spotlight/

[3] **Yep**, http://www.yepthat.com/

[4] **Google Picasa**, http://picasa.google.com/

[5] **Flickr photo sharing site**, http://www.flickr.com

[6] **TF/IDF Algorithm**, http://en.wikipedia.org/wiki/Tf-idf

[7] **Ling Pipe**, Open Source Linguistic Analysis Project, http://www.alias-i.com/lingpipe/index.html

[8] **Hidden Markov Model Definition**, http://en.wikipedia.org/wiki/Hidden_Markov_Model

[9] **Ling Pipe Linguistic Pipeline Architecture and Algorithms**, http://www.colloquial.com/carp/Publications/LingPipe04.ppt

[10] **20 Newsgroups Corpus**, http://people.csail.mit.edu/jrennie/20Newsgroups/

[11] **K-Means Clustering Algorithm**, http://www.elet.polimi.it/upload/matteucc/Clustering/tutorial_html/kmeans.html

[12] **PAM Clustering Algorithm**, http://www.unesco.org/webworld/idams/advguide/Chapt7_1_1.htm

[13] **Agglomerative Clustering Algorithm**, http://ncisgi.ncifcrf.gov/~lukeb/agclust.html

[14] **CLARA Clustering Algorithm**, http://www.unesco.org/webworld/idams/advguide/Chapt7_1_2.htm

[15] **Apache Derby**, Open Source Database Project, http://db.apache.org/derby/

[16] **Apache Lucene**, Open Source Search Engine Project, http://lucene.apache.org/java/docs/

[17] **Jickr**, Open Source Java Wrappers for Flickr API Project, https://jickr.dev.java.net/

[18] **Flickr API Specification**, http://www.flickr.com/services/api/

[19] http://www.charrison.net/ImageSearchThesis.pdf

[20] **Prefuse**, Open Source Visualization Toolkit, http://prefuse.org/

[21] http://cs.nyu.edu/~jnb263/TimeIn_Instructions.htm

Appendix A. User Study Questionnaire

Thank you for participating in the TimeIn User Study. Please attempt to work through the following questions on your own and roughly time how long it took to complete each task. If you get stuck on any task feel free to ask for help.
.

1. Refer to the instructions page at http://cs.nyu.edu/~jnb263/TimeIn_Instructions.htm and install TimeIn on your local machine

2. Import all photos from the Flickr User: slice

3. Using TimeIn, find a picture consisting of a plastic green animal toy on the hood of a car taken in Portland.

4. Using the Flickr web site, find a picture of a person in a red car taken in Portland. Comment on the differences

5. Create a few albums containing a dozen "related" images.

6. Find a dozen images of "two person portraits"

Please comment on any steps that proved difficult or were particularly useful.