Web-Based Emergent Manuscript Transcriptions

A Major Qualifying Project Report:
submitted to the Faculty of the
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements for the
Degree of Bachelor of Science
By:

Paul Freitas
and
Scott Glajch

Advisors:
Fabio Carrera
and
Stanley Selkow

April 26, 2007

Abstract

This project, part of the larger Emergent Transcriptions initiative, designed and implemented tools which will assist archives in digitally storing historical manuscripts and their transcriptions, and enables searching of their collections. Our contributions to the initiative included optimizing existing visual processing systems using a genetic algorithm, and building a database and web front-end for the system, to facilitate searching and editing of archival collections. We also redesigned core features using good Software Engineering principles to make future system additions easier.

Executive Summary

The history of the world is stored in a variety of artifacts including manuscripts. These manuscripts are hand-written accounts of historical events, from the day-to-day transactions of local governments, right up to personal accounts of important historical events. Historians use these manuscripts to do their research and provide accurate representations of historical events and periods. However these manuscripts are difficult to decipher, and degrade over time with use and even improper storage. Many of the documents also get transcribed several times because people do not share they work they have done in transcribing a manuscript. This is where the Emergent Transcription initiative

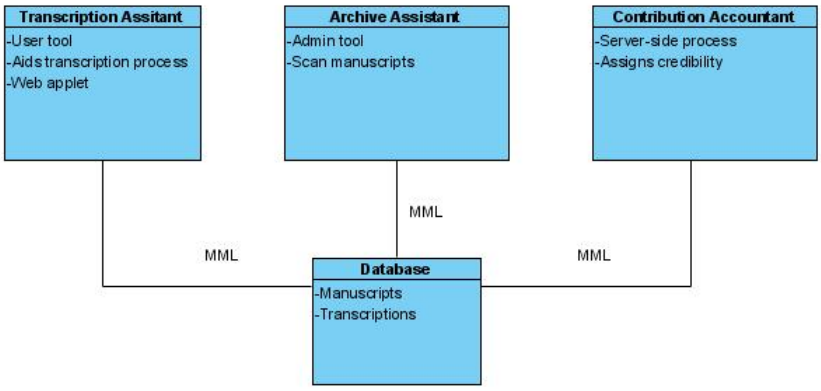

Figure 1.1: Basic Layout of the Emergent Transcriptions System

fits in.

The Emergent Transcriptions initiative is the broader system which this project seeks to advance. This system consists of three major subsystems which provide different roles in the process of storing, transcribing, and sharing transcriptions. The Archive Assistant (AA) is a tool to assist archivists in the process of uploading their digitized manuscripts, attach any metadata information about the document, and attach any existing transcriptions of the document to the manuscript. This tool would be used by archivists to upload their documents in bulk at the beginning of their involvement with the Emergent Transcriptions initiative.

The Transcription Assistant (TA) is a program to help historians transcribe existing manuscripts, creating a detailed description of the words, images, and other elements of a manuscript as well as their location on the page. This transcription can then be shared with

others for future use, but will always be linked back to the original manuscript file from which it was created.

The Contribution Accountant (CA) is a program that keeps track of the transcriptions and edits to transcriptions that different users do, and put a credibility score on them, depending on any number of factors. These factors could include the user's overall credibility, amount of transcription work done, rating of transcriptions and edits by other users, and amount of times their transcription or edits have been accepted and unchanged.

The reason this system is called Emergent Transcriptions is that it leverages the idea of Emergence, where the work of creating high-quality transcriptions is done by the vast user base of the system editing and tweaking existing transcriptions until the best results emerge from the mess of various edits made over time. The goals of this system are to make the transcribing process easier, leverage the transcribing work that is already being done on documents so that the information can be used again, as well as foster a larger set of transcription data in which historians do their research.

The original goals for this project were determined by looking at the current state of the larger Emergent Transcription initiative, and deciding which features were most important that were not yet implemented in this system. One of the goals was create a means for manuscripts and their transcriptions to be stored electronically and in a central location, using the Transcription Assistant software, and an import tool. This would give archivists a chance to start submitting their data and start building a repository of transcriptions. Another goal of this project was to start work with Optical Character Recognition (OCR). This would allow the program, through the use of existing data, to start to recognize words in a manuscript based upon previous manuscripts and their transcriptions. This will make the job of transcribing easier.

Some design decisions were made along the way to alter the original goals of our project. The first was that we decided that the Transcription Assistant, which is built upon a client-side program created by a previous project, should be migrated to an applet to be served over a web page. This makes it easier for users to upload their transcription work as well as search for transcription work already done by others. The completion of this applet became another one of our goals early on in the project.

Another thing that became apparent early on is that the prerequisites for Optical Character Recognition (OCR) were not being met in the program as it stood. Since OCR required well segmented words, it became apparently that getting the automatic boxing algorithm to correctly and fully segment words to send to any OCR algorithm would be a better goal for the project.

The first section of the project was spent looking into the current automatic boxing algorithm, and cleaning up the old version of the TA. This lead into the creating of a Genetic Algorithm, which mimics survival of the fittest from ecology to do sample runs of the automatic boxing. These runs were given a fitness score to compare the boxing with a "perfect" boxing done by hand. In this way the Genetic Algorithm optimized the automatic boxing algorithm to perform much better. At the same time, the TA was being restructured to work as a web applet.

From this genetic algorithm and new applet were a working TA system and a means to start building the larger system around it. The database was then created, as well as a server-side process meant to communicate with the TA. This communication allowed for searching the database from the TA, as well as serving the manuscript images and transcriptions to the TA.

It became evident that to make a functional import tool there not only had to be a way to automatically box the image, but also to order boxes and store their orderings in the Manuscript Markup Language (MML) that was currently storing all of the transcription information. To this end the MML was redesigned to include a lot more features, including the ability to store lines and paragraphs of boxes. A line detection algorithm was also designed. All of this work made the first working version of an import tool possible.

Finally, with what time was left, a simple web front-end was created to both serve the applet in which the Transcription Assistant ran, and allow the user to search for existing manuscripts and transcripts which he or she could load into the web applet.

All of these components were designed with the implementation of the Contribution Accountant in mind, and allowed for the storing of credibility scores to users, transcriptions, and transcription elements.

The results of all of this work was a working Transcription Assistant program inside of a web applet, in which the user could search for and download existing manuscript and transcriptions to work on, as well as the functionality to upload work back to the server.

The genetic algorithm provided the TA with a better initial automatic boxing algorithm as well, and is extendable to any other major algorithm within the Emergent Transcriptions initiative. Finally the core functionality is implemented to make the first version of an import tool possible. The system developed should be enough to let archivists start uploading their work and using the system.

Table of Contents

Table of Figures

1.0 Introduction

Since the invention of writing, civilizations have kept detailed written records about all facets of life. Now, those ancient records that have survived are a vital component in our understanding of the past. Huge archival collections of written documents exist all over the world that combined contain millions of records. Each item in these archival collections has the potential to be the key to our understanding of ancient people, events, and ideas. Unfortunately, this potential gold mine of information remains largely untapped, and we lose more and more of it each day.

Two fundamental problems exist with historical records that impede the kind of exhaustive research that might be possible in these huge collections. First, historical records are difficult to read and understand due both to damage and antiquated language. Second, historical records, regardless of medium, have a limited lifetime. Time claims more and more of these documents with each passing year.

There is a pressing need to study these documents before they are lost. Unfortunately, work progresses slowly despite the huge number of researchers that work with historical records on a daily basis. These researchers are hindered by inadequate tools for processing these manuscripts, and a lack of coordination that causes work to be repeated multiple times.

The Emergent Transcriptions Initiative seeks to address both of these issues in the historical community. By streamlining the overall effort to process these historical records, the initiative hopes to leverage the work that is already being done by researchers around the globe. This will speed the process of not just rediscovering data hidden within archival collections, but also capturing it for the historical community to examine.

This document describes one small step towards the system that will make that possible. It first describes in greater detail the conditions behind the problem, and the tools that make up the Emergent Transcriptions Initiative. It then describes the progress that was made during the 2006-2007 academic year, as a part of our Major Qualifying Project (MQP) at Worcester Polytechnic Institute (WPI). Finally, it provides a glimpse of the future of this project, and our suggestions on the next steps in the process.

2.0 Background

Development of a tool to help transcribers first requires an understanding of the research process when working with historical manuscripts. Our work is built on previous work done that has already examined this facet of the Emergent Transcriptions initiative in great detail. This chapter is intended to provide the reader with an overview of historical research using manuscripts as well as a picture of the status of the Emergent Transcriptions initiative at the beginning of our work.

2.1 Overview of Historical Research Using Manuscripts

At the core of the Emergent Transcriptions initiative are historical manuscripts. These are ancient documents that usually were never intended to be published. The Emergent Transcriptions initiative is intended to streamline the process of using manuscripts in historical research. It is therefore important to understand the importance of manuscripts, how they are cataloged, what transcriptions are and why they are important, and how manuscripts are stored digitally. This section explains all of these, and gives an overview of the process historians typically follow when doing historical research with manuscripts.

2.1.1 Importance of Manuscripts

The most important type of source of information in historical research is a "primary source". A primary source of information is a document, artifact or other source of information that was created close to or during the event or time period being studied. There are three types of primary sources used in historical research: (1) Oral traditions and oral histories, (2) Physical artifacts, and (3) Documents, both printed and handwritten (http://memory.loc.gov/learn/lessons/psources/analyze.html). Of these three types, the third is perhaps the most important and prolific.

The Library of Congress further divides the third category into published and unpublished works. This distinction is important because few copies exist of each unpublished work. These predominately handwritten documents are usually referred to as "manuscripts" and can consist of personal correspondence, town records, court proceedings, and many other first-hand accounts of historical events.

These manuscripts can be found in thousands of repositories across the globe, stored in archives which protect them as best as possible from physical deterioration, and attempt to catalogue them in a useful fashion. These tasks often prove difficult, as the collections are usually quite large—the works of a single author often total in the hundreds of thousands of pages (Rath et al., 2004).

Much of the information contained in these documents is relatively unimportant—minutiae of everyday life, well documented in other texts. However, often historians will "unearth" important data about significant events buried in these accounts. For this reason, these vast collections of historical documents are important because of their potential to reveal lost secrets about the past.

Sifting through this data is a long and arduous process, and unfortunately the time we have to do so is limited. The materials these historical accounts were recorded on are fragile; they were never intended to last hundreds of years. Archives devote immense resources to protecting and preserving the documents in their care. Water, mold, and even light can damage manuscripts until they become unreadable. Archives must restrict access to their collections in order to reduce the number of hands that handle any given document; each viewing of a manuscript quickens its deterioration.

2.1.2 Cataloging Manuscripts

In addition to physical preservation of manuscripts, archives face another major problem: organizing their collections. Organization is critical because it is what allows researchers to locate documents of importance. An archive's primary mission of preserving documents is only worthwhile if those documents can be used.

At present, there are four primary pieces of information that archives use to organize their collections: author, subject, date, and geographic area (ISAD(G): General International Standard Archival Description, Second Edition). The relative effectiveness of each of these depends on the exact nature of the archive's collection. For example, the records for the government of Venice would be less useful sorted by geographic area or author than they would if sorted by date and subject.

2.1.3 Metadata

Most of the information used to organized manuscripts can be considered "Metadata", which means is literally "data about data". There are many different elements of metadata tracked by different archives around the world, and there have been a number of efforts to standardize the metadata used. Three of these will be discussed here: MARC, the Dublin Metadata initiative, and the ISADG.

2.1.3.1 Bibliographic Machine-Readable Cataloging (MARC)

MARC is a metadata standard originally designed for the Library of Congress in the 1960s for use in library catalogs. MARC records for a given item in a collection conform to both national and international standards. MARC has been updated twice (USMARC in the 1980s and MARC 21 in the 1990s), and is the standard metadata format used in tens of thousands of libraries.

A MARC record requires four pieces of information per item in the collection, though the standard allows for further data to be entered. The four required parts can be entered in any order. The first is a description of the item, including the title, edition, and physical description. The second is the "main entry" and "other added entries", used when a given item might be found under multiple titles or authors. The third is a list of subject headings, which are selected from a standard list of subject headings. The fourth is the call number, which follows the Dewey Decimal System.

MARC is widely used for two primary reasons. First, it is a very flexible standard that allows for a high degree of granularity in the included and excluded metadata. Second, the standard is currently very widely used. However, there has been some debate in recent years because of its complexity, and because the technology it is based on is very outdated. For the archive community, MARC may not be the best choice for Metadata, as the standard was designed for libraries of contemporary books, not historical manuscripts.

2.1.3.2 Dublin Core Metadata Initiative (DCMI)

The DCMI was started in 1995 in Dublin, Ohio by the Online Computer Library Center (OCLC). "Core" refers to the construction of the Metadata set, which includes 15 basic "core" elements, but allows for extension. The standards of the DCMI are designed to

be flexible enough for many different applications. According to the group, the DCMI designs its standards "to facilitate the finding, sharing and management of information".

The 15 core elements created by the DCMI are (Element names and descriptions are from http://dublincore.org/documents/dcmi-terms/):

1. Contributor: an entity responsible for making contributions to the resource
2. Coverage: the spatial or temporal topic of the resource, the spatial applicability of the resource, or the jurisdiction under which the source is relevant
3. Creator: an entity primarily responsible for making the resource
4. Date: a point or period of time associated with an event in the lifecycle of the resource
5. Description: an account of the resource (may include things such as an abstract, table of contents, etc.)
6. Format: the file format, physical medium, or dimensions of the resource
7. Identifier: an unambiguous reference to the resource within a given context (Some unique identifier—may be a call number)
8. Language: a language of the resource
9. Publisher: an entity responsible for making the resource available
10. Relation: a related resource
11. Rights: information about rights held in and over the resource
12. Source: the resource from which the described resource is derived
13. Subject: the topic of the resource
14. Title: a name given to the resource
15. Type: the nature or genre of the resource

The DCMI also defines a number of extensions to each of these basic fields. Examples of extended fields include things such as "created" for creation date, which would be part of the "date" core element, or "abstract" as part of the "description" core element.

The DCMI makes a number of conversion utilities available on its webpage (http://dublincore.org) that allow conversion of metadata between the DCMI standard and various other standards, including USMARC. Dublin core metadata is usually expressed, stored, and transmitted as XML, and the DCMI provides XML schemas on its website.

2.1.3.3 General International Standard Archival Description (ISADG)

ISADG is a metadata standard developed by the International Council on Archives (ICA). The ISADG is a standard designed specifically for archives, and the ICA recommends it be used either in addition to or as the basis for other metadata standards. The ISADG contains twenty-six elements that are designed specifically for the description of

archival records. The standard organizes these elements in a hierarchical structure by the level of detail of the description.

2.1.4 Transcriptions

The traditional solution to the conflicting needs to preserve and use historical documents is the creation of transcriptions. The creation of transcriptions is also typically one of the steps a historian takes in processing a document during research. Transcriptions have two primary benefits over original documents. First, they are not as fragile as the source document. Second, they are often easier to understand and read than the original.

A transcription is a document which is intended to take the place of the original. It is more than a simple copy of the final text of a document; typically it also describes other features of the manuscript. Marks and notes in the margins, writing between lines, and stricken passages can all be as important to a historian as the main body of text (Finch, 1999).

Creating a transcription is a time-consuming and labor-intensive process (Rath et al., 2004). Manuscripts are often difficult to read because of damage and age, and difficult to understand because of archaic words, expressions, and constructions (Finch, 1999). Still, they are an essential part of the research process, especially because archives limit access to the original documents.

2.1.5 Digital Manuscripts

Recently, more and more archive collections have begun to be converted to digital format. This process allows for more frequent use of manuscripts without subjecting them to the handling that is involved in physical examination. This is good for both archives and researchers, as documents can be simultaneously better protected and more widely used.

2.1.5.1 Manuscript Images

Digitization of a document typically begins with scanning its pages, producing a digital image. Large collections of these manuscript images are difficult to use, however, because current technology for searching image contents is not very developed (Rath et al., 2004). The common solution to this issue is to manually attach metadata describing the document to the image in the database.

2.1.5.2 The Text Encoding Initiative (TEI)

One organization that has attempted to provide a standard for representing digital versions of documents is the Text Encoding Initiative (TEI). The Guidelines published by the organization are designed to allow "a variety of literary and linguistic texts" to be represented for online study and preservation (http://www.tei-c.org/). The various XML and SGML elements that the organization provides are intended to be inserted by hand into plain-text transcriptions of texts in order to mark features such as chapter beginnings, headings, and lines of poetry.

2.1.6 Manuscript-Based Historical Research

In order to create a tool for studying historical manuscripts, the existing process for doing so must be understood. This facet of the Emergent Transcriptions initiative was previously examined. What was found is that the current process suffered from major inefficiencies, and is time-consuming and labor-intensive. Their description of the process was based on the system in place at the Venetian Archives, but a similar procedure exists in other archives. Figure 2.1 shows an overview of the procedure.

2.1.6.1 Description of the current process

The process begins with a historian searching the archive's records of manuscripts they own. Unfortunately, information about the contents of the manuscript is limited, so the search must be based on the metadata the archive has on file. This means that a historian's search results can yield a large number of possibly useful manuscripts.

The researcher then requests a manuscript from the archive by collection set and series. Archive personnel check the archive's reserve to see if the requested document is already in reserve. If it is not, they must search the archive's stacks for the manuscript. When they find it they put it in reserve. Once the document is in reserve, it is available for the researcher to study. The delay between the request and receipt of a document can be as long as two weeks due to the size of the collection and inefficiencies in the cataloging system.

Once the researcher is notified that the manuscript is available, they may view it in the archive's reading room. This is the only location in which people who do not work for the archive may view the original documents. Therefore, all work with the document must

be done in the reading room. For this reason, transcription is an important part of the research process.

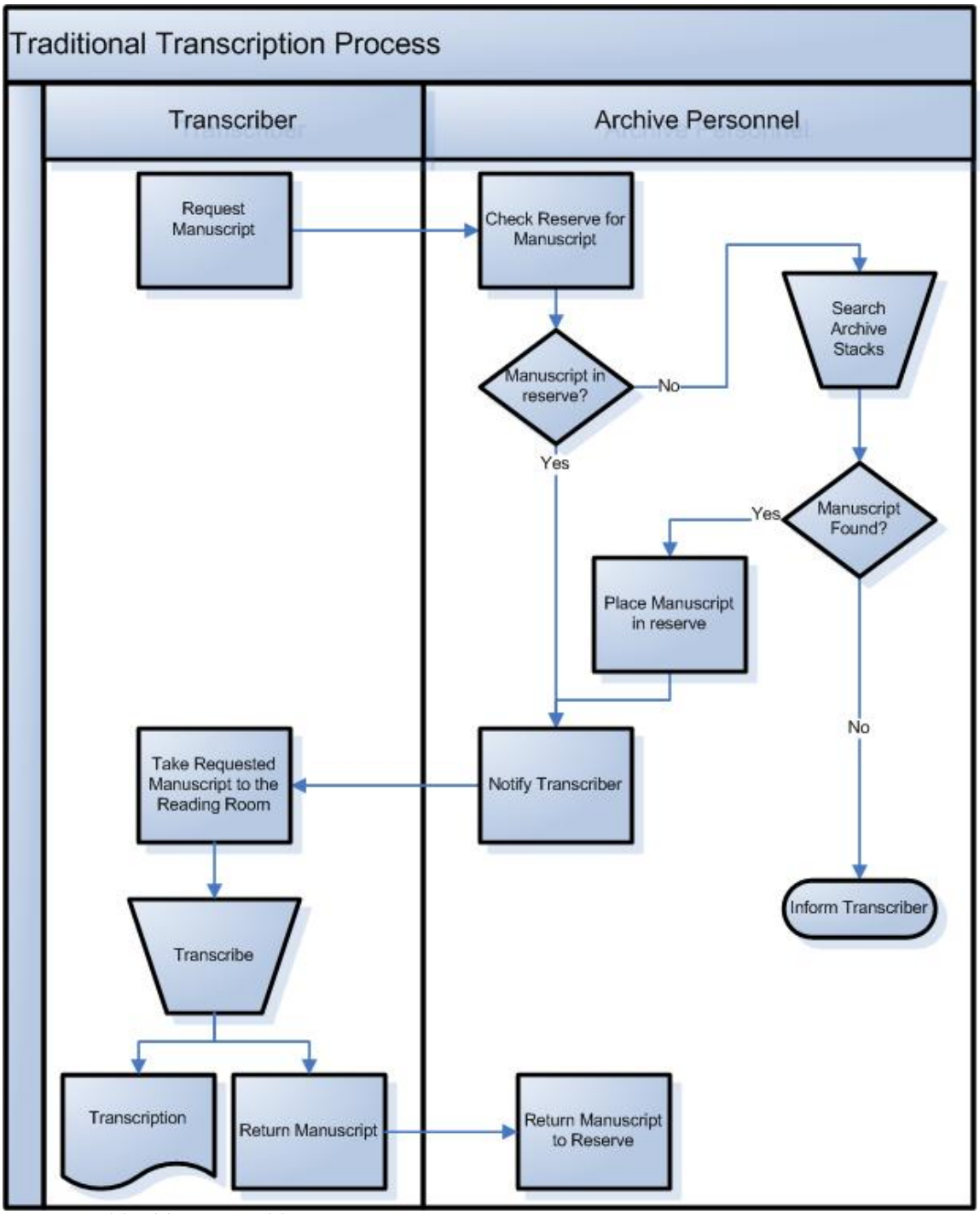

Figure 2.1: The Traditional Transcription Process

The actual transcription of the document is often a long and difficult task. The process often requires use of dictionaries of abbreviations and symbols and other reference

texts. A researcher may require multiple trips to the archive in order to complete transcription of a single document. On each visit, the researcher must again have the manuscript brought to the reading room, and must locate the point at which they stopped work. Once the transcription process is complete, the researcher returns the original document, and keeps the transcription for personal use.

2.1.6.2 Consequences of the Current Process

Two characteristics of the existing research process make it particularly inefficient. First, the methods for locating manuscripts are currently not precise enough to quickly search for manuscripts covering specific topics. Second, the transcriptions created by researchers are rarely shared with other members of the research community. Together, these two characteristics lead researchers to repeat work already completed by other historians, and to use well-known manuscripts rather than uncovering new information.

Currently, searches for manuscripts are largely restricted to the contents of the metadata for each document. Archives already spend significant time and resources keeping these records accurate (Rath et al., 2004). Unfortunately, this data is often not specific enough for researchers searching generic records for mention of specific people, places or events.

A study of history professionals in 2002 found that the most often used method of finding primary sources during research was finding leads in other printed sources, which 98 percent of respondents said they regularly used for research (Tibbo, 2002). The next three most commonly used methods (81 percent each) were printed bibliographies, printed finding aids, and printed repository guides. The first of these three is very similar to finding leads in printed sources, and the latter two must be compiled by the owners of collections (a time-consuming task). By far the most commonly used electronic method of searching was searching an OPAC database, which was only used by 78 percent of respondents.

This information means that historians are using manuscripts that have already been analyzed by their peers far more often than they examine any of the incredible volume of manuscripts which have not been touched in years. This leaves a possible vast resource of information untapped. If more sophisticated methods for searching are not developed, more and more documents could be lost to deterioration before their value can be determined.

The other major problem with manuscript research as it currently stands is that the same work is often repeated many times by different researchers. When a transcriber completes the transcription process, he keeps the transcription for his own personal use. The archive takes back the original document, but usually does not even record that a transcription was made. When other researchers choose to use the same manuscript for their own work, they often end up repeating the transcription process.

While the archives are partially to blame because they do not keep records of transcriptions, the overall system is far more responsible for this gross inefficiency. As the current historical research process stands, there is no incentive for a transcriber of a manuscript to share his work with the community. Doing so can give others working on similar projects an advantage, and allow them to publish their findings first.

2.2 Goals of the Emergent Transcriptions Initiative

The overall goal of the Emergent Transcriptions initiative is to create a system that will leverage the work that is already being done by visitors to archives around the world. It seeks to create an ever-growing collection of transcriptions that are refined by the work of many researchers. In this way, it hopes to uncover important information hidden in thousands of still untapped manuscripts (Carrera, 2005).

The Emergent Transcriptions initiative is predicated upon five assumptions, according to Carrera (2005):

- "There are precious few researchers with the necessary paleographic skills who are able to produce reliable transcriptions of ancient manuscripts;
- "Despite this crucial bottleneck, these few capable individuals frequently duplicate efforts by re-transcribing the same exact manuscript that someone else has already worked on, often unbeknownst to each other;
- "The constant manipulation of the primary sources (parchments and the like) renders them less and less legible as time goes by;
- "Very few manuscript transcriptions are published verbatim;
- "The work put into transcriptions is subsumed into scholarly journal articles and books, thus it is rarely if ever seen or re-used by others." (Carrera, 2005)

2.2.1 What is "Emergence"?

At the core of this project is the idea of "Emergence". In this case, it refers to the way in which high-quality transcriptions will emerge from the work of many different participants. The contribution of each researcher who adds to or even reads transcriptions

within the project will be added to the system's repository of information. This is not a new idea; similar projects, such as Wikipedia, have been built on similar principles.

The Emergent Transcriptions initiative builds upon that idea by adding the concept of "credibility". As a given transcriber's work is viewed and modified by others, acceptance or modification of that work contributes to the original submitters' credibility score. As more people accept a transcriber's work as correct, the credibility score of that transcriber will increase. If others modify that transcriber's work, their credibility score will decrease. When deciding upon an accepted transcription out of the work of many different transcribers, the system will consider these credibility scores. Work done and accepted by the most credible transcribers will be included in the accepted final text of the document.

2.2.2 Parts of the Emergent Transcriptions Initiative

The final version of the Emergent Transcriptions initiative will consist of three related subsystems. These are the Transcription Assistant, the Archive Assistant, and the Contribution Accountant. The role of each of these will be described in the following sections. An overview of the various components and how they interact is shown in Figure 2.2.

2.2.2.1 Transcription Assistant

The most visible part of the Emergent Transcriptions initiative will be the Transcription Assistant. This tool will be used by all transcribers while transcribing manuscript images the archive has loaded into the Emergent Transcriptions initiative. It will allow the transcriber to retrieve existing transcriptions and manuscripts, and to save the transcription work that they do.

The Transcription Assistant is particularly important to the project because it will serve as the primary incentive for researchers to use the Emergent Transcriptions initiative. It is intended to make the task of transcription easier in at least four ways. First, it makes it easier for a transcriber to find their place in the manuscript should they have to consult another reference. Second, it may in the future be able to give suggestions for words through sophisticated Optical Character Recognition techniques. Third, it will allow researchers to consult existing transcriptions in the Emergent Transcriptions database. Fourth, and perhaps most importantly, it will allow a researcher to retrieve a manuscript or

transcription almost instantly from the server. It is important to ensure that transcribers want to use the Transcription Assistant, because it will be the primary incentive for them to return their transcription work to the larger community.

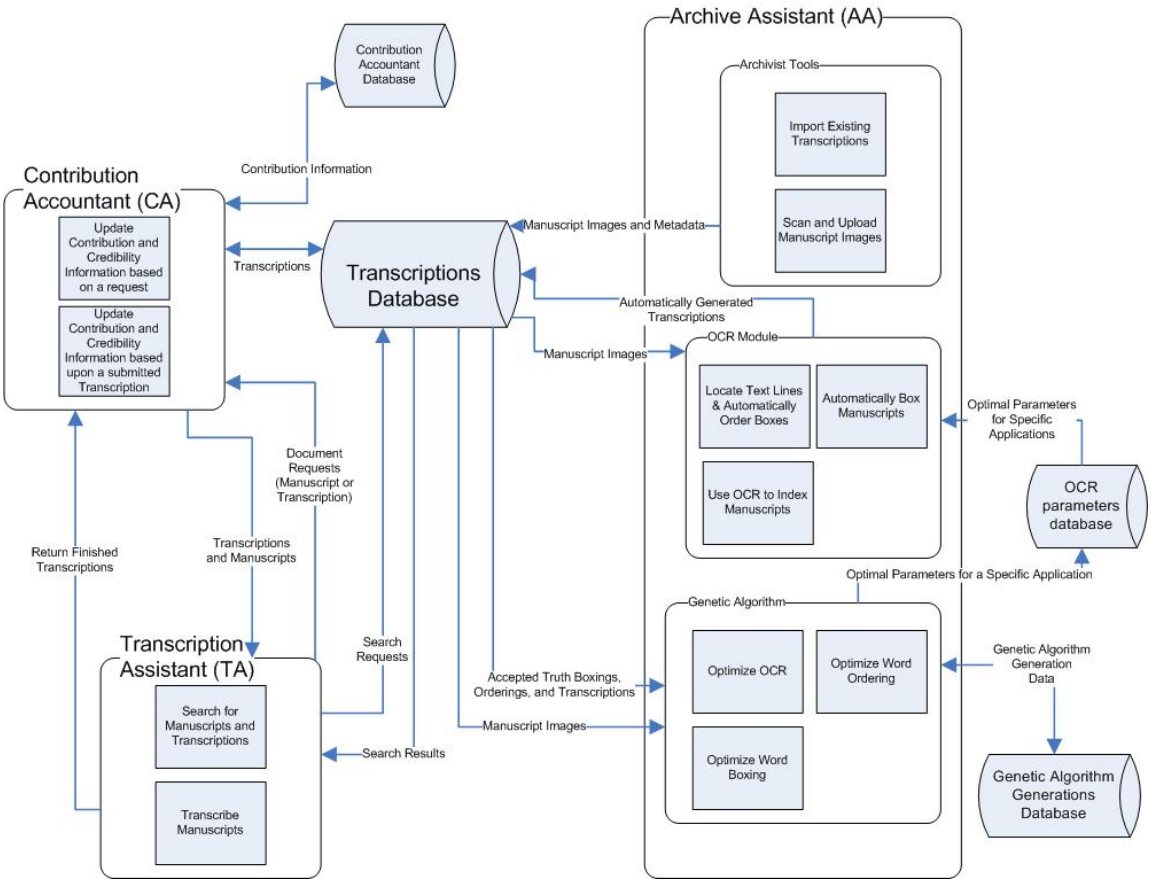

Figure 2.2: Overview of the Emergent Transcriptions System

2.2.2.2 Archive Assistant

The Archive Assistant portion of the Emergent Transcriptions initiative comprises a number of tools which assist the archive in managing the Emergent Transcriptions initiative. Most of these tools will make up the back end of the system. This portion of the system includes three major tools. First is the database for all uploaded manuscript images. Second is a tool for archivists to easily and quickly load manuscript images and metadata into the database. Third is the automatic manuscript processing tool, which automatically boxes, orders, and uses OCR techniques in order to index manuscript images which have not been transcribed. The Archive Assistant may also include a user database and manager for all active users.

2.2.2.3 Contribution Accountant

The Contribution Accountant is the portion of the system that tracks the "credibility scores" for all known users. After each modification of a transcription, and before the new version is added to the database, the Contribution Accountant will process the modifications to determine how the changes should affect the credibility of other transcribers who have worked on the document. It will also use its records of credibility to determine the current accepted text of the transcription.

Though the credibility system has not yet been designed, some basic features have already been decided. There will be two types of credibility scores: scores for individual users, and scores for each modifiable item in a transcription. Each time a user modifies a transcription, the portions of the transcription they changed are counted as "votes" against the previous values for those portions, and votes for their own submission. A given user's vote would be weighted by their own credibility, and would modify the credibility score for all items they confirmed or changed. A user's credibility score would be increased when their submissions were confirmed, and decreased when their submissions were modified. The credibility scores of elements in transcriptions would be used to determine the "accepted text" of a transcription. Simply contributing to the project by submitting a transcription would increase a user's credibility by some base amount, in order to encourage participation.

Another aspect of the credibility system that has been proposed is allowing access to more sophisticated features only to high-credibility users. For example, normal users might have access to a basic OCR library or a limited number of uses of the OCR system during transcription, while a higher credibility user could use author-specific OCR systems and could do so more frequently. The credibility system could also be used to encourage users to give back to the community by restricting the number of transcription downloads allowed in a given time period based on the credibility of a given user.

2.2.3 Role of the Project in Manuscript Research

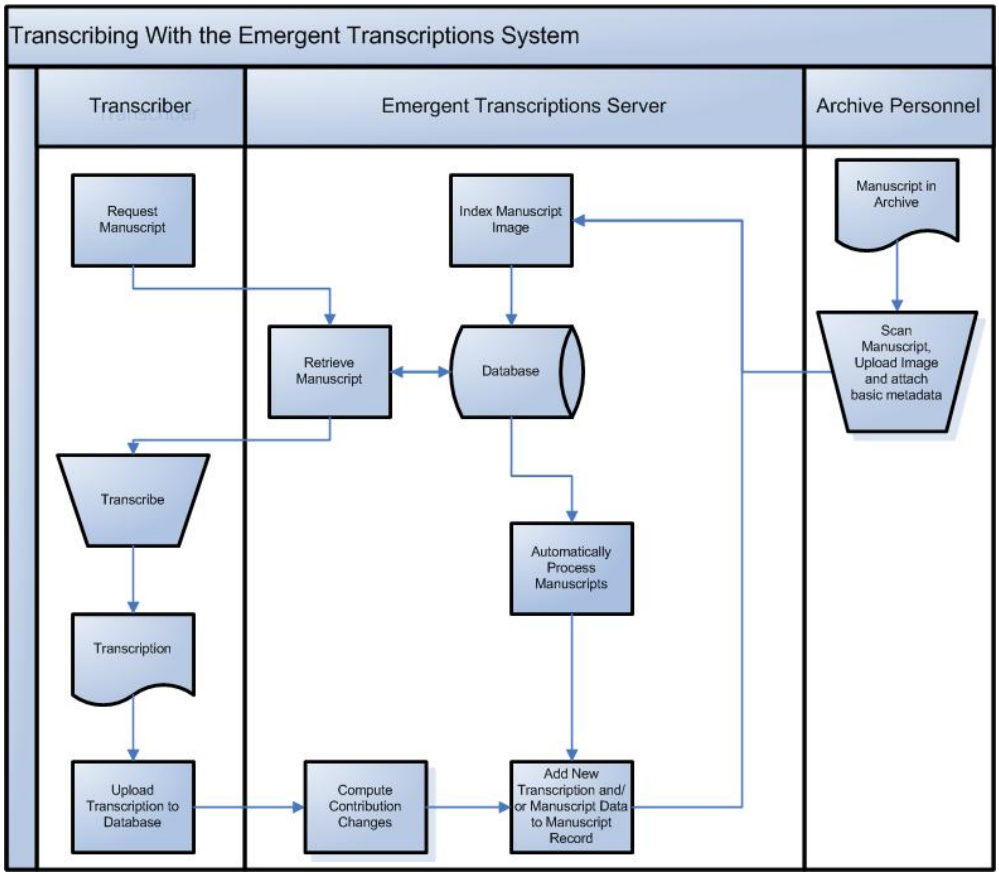

Figure 2.3: Manuscript Workflow Using the Emergent Transcriptions System

Figure 2.3 demonstrates the way in which the manuscript research process will change once the Emergent Transcriptions initiative is used at an archive. There are now three parts to the process. The transcriber and the archive personnel still have significant roles in the process, but they no longer directly interact. Both work with the Emergent Transcriptions server to get things done.

The process now begins with the archive personnel entering a manuscript into the system. This will be an ongoing process until the archive's entire collection has been added to the Emergent Transcriptions initiative. The archivist will scan the manuscript, attach some basic metadata to it, and upload it to the database, all with the help of the Archive Assistant. A portion of the Archive Assistant on the server will then index the manuscript and enter the correct data into the database. The archivist can then continue to upload additional manuscripts.

Once manuscripts have been uploaded, the transcriber can request to view manuscripts using the Transcription Assistant or an associated web application. The transcriber will be able to search the manuscripts using a variety of criteria, including full text of a transcription, and any of the standard metadata elements. The server will respond almost instantly, retrieve the requested manuscripts, and make them available to the transcriber. This will allow the transcriber to quickly browse through the results until he finds one that interests him.

At this point, the transcription process begins. The Transcription Assistant will make this step of the process much easier, but it will likely still involve a significant amount of work. When the transcriber is done with the transcription, he may keep a copy of it, and a copy is uploaded to the database.

At this point, the Contribution Accountant will examine the changes the transcriber made, and update the credibility information in the database accordingly. Any parts of the transcription the transcriber "confirmed" by not modifying them will result in an increase in the credibility of those elements and in the credibility of the transcribers who originally transcribed them. Any changes the transcriber made will decrease the credibility scores of both the changed elements and the corresponding transcribers. If the change the transcriber made was to a value a previous transcriber entered, that transcriber's credibility will increase. All changes in credibility in these cases will be weighted by the current transcriber's own credibility.

Once all these changes have been made, the Contribution Accountant will return the transcription to the Archive Assistant, which will re-index the manuscript and update its record in the database.

There is one additional portion to this process: automatic processing that the Archive Assistant does when it has free cycles. There are currently two proposed activities which the Archive Assistant could engage in when idle: Automatically indexing manuscripts that have not been transcribed using OCR methods, and optimizing OCR and segmentation parameters for specific classes of manuscripts. These two processes should allow more obscure manuscripts to be found in searches, thereby increasing the likelihood that more and more manuscripts will be examined.

2.3 Previous Work on the Emergent Transcriptions Initiative

The Emergent Transcriptions initiative has been going on for some time, with the most recent work 2004. This section gives a brief overview of the status of the initiative at the beginning of the 2006-2007 academic year, when our contribution started.

2.3.1 Transcription Assistant

The MQP in 2003 also created the first major version of the Transcription Assistant, which is the tool to help someone transcribe an existing manuscript. Their work featured a boxing system on top of the manuscript image that allowed the user to work with both the original image they were boxing, and the transcription text side by side. The user would draw a box around a word, and then double click on the box to edit the text value associated with the box. That text shows up on the adjacent panel in the same position where the box was in the image. A screenshot of the system at the beginning of our project can be seen in Figure 2.4. Though this system was updated during a 2004 MQP, it still was not a complete working program, and several of the components were either not completely implemented, or had bugs. Still, the general interface for editing the boxes was present, so we decided to work off of and redesign this program.

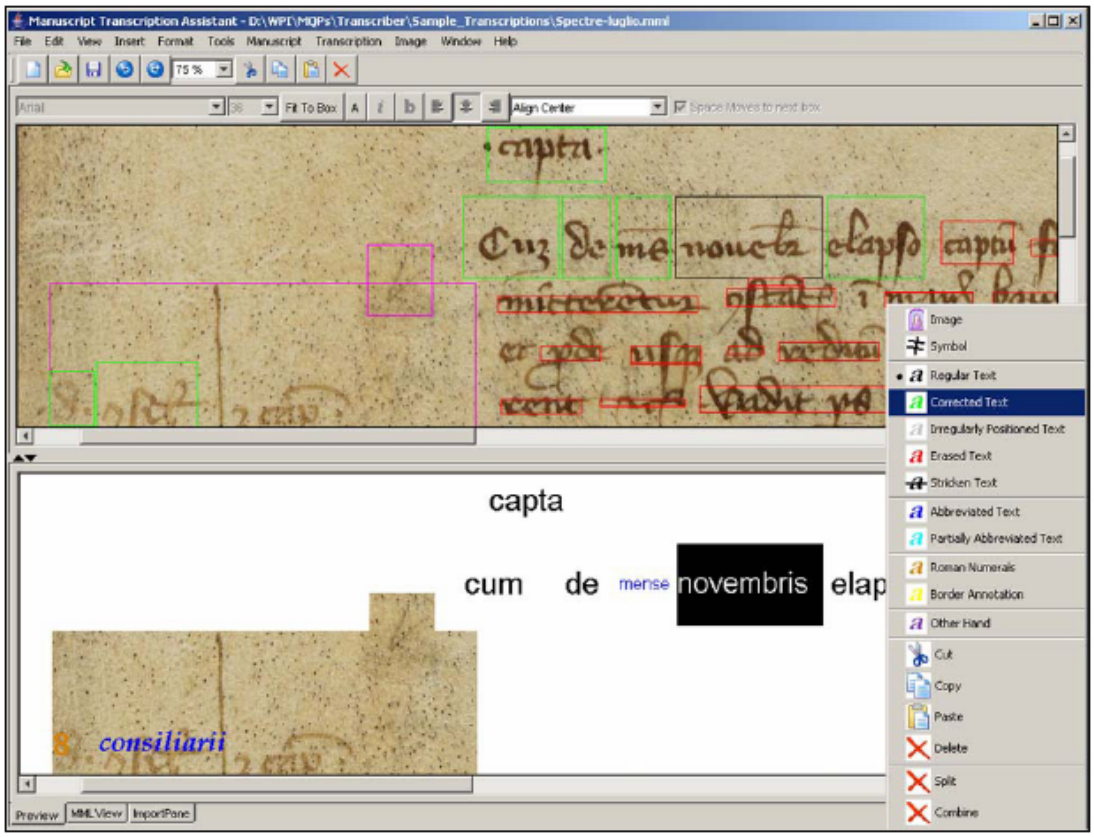

Figure 2.4: Previous Completed Work on the Transcription Assistant

2.3.2 Manuscript Markup Language (MML)

A previous MQP completed in D term of 2003 had devised a basic structure for storing the transcription information for a particulate manuscript that they labeled as MML, or Manuscript Markup Language. This was the data structure implemented in the pervious Transcription Assistant program. While the MML version was not fully XML compatible, it worked in a similar fashion. It was basically a list of unordered box tags. Each tag has five attributes, the x and y pixel locations for the top right and bottom left corners of the box, as well as a type field, which indicated whether the box was a text box or image box. Within each tag was the actual word or words that the user input as the transcription for the word or words boxed, along with some simple HTML-style text-markup identifiers, such as bold tags (Ho et al., 2003, pages 40-42). While the old projects did store metadata for each transcription entered by the user, it was not stored in the MML. We decided to restructure this MML into fully a new fully XML compliant structure which allowed for a more descriptive document representation (see Section 0).

2.3.3 Automatic Boxing

An MQP in 2004 extended the Transcription Assistant to include an algorithm for automatically creating boxes around words in a manuscript, and then displaying those boxes on the screen. Though this algorithm places no order on the boxes, and did not do a complete job of boxing every word correctly, it worked well enough so that we started off our autoboxing research based upon this existing tool.

The algorithm converted the color image temporarily into a grayscale image. Then using the Java Advanced Imaging's (JAI) existing conversion tools, it binarized the image into a black and white image. This black and white image was passed through an algorithm which goes through the image horizontally and vertically looking for runs of white pixels smaller than a certain threshold, and turns the small white pixels runs to black pixels. The result is that the image looks smeared horizontally and vertically. These two images are combined by taking the cross-section of the black smeared areas, to create an image with sections of black smears over each word. Then boxes are created around these words, and finally boxes that are too small or overlapping are combined. This algorithm has some thresholds which can be manipulated to produce better performance in different styles of manuscripts.

2.4 Optical Character Recognition (OCR)

A key piece of the final Emergent Transcriptions initiative will be a sophisticated Optical Character Recognition (OCR) system. Research on OCR algorithms revealed that at this point in the Emergent Transcriptions initiative, it was not feasible to begin work on the OCR system. However, the information found on OCR provided some insight into what the system still requires before such a system can be built. This section describes what that research found.

2.4.1 Types of OCR approaches

Most OCR algorithms can be broadly divided into one of three categories:

1. Character-by-character recognition
2. Whole-word recognition
3. Word-spotting

Each of these has a slightly different approach to the character recognition problem, and each is most effective in certain situations. Each of these three will be explained in more detail in the following sections.

2.4.1.1 Character-by-Character Recognition

The character-by-character approach to OCR can be thought of as a bottom-up technique. These algorithms function on the level of a word image, but begin by splitting that image into single-character sections. This process is called "character segmentation".

In this approach, the algorithm attempts to identify each character. It may do this independently, or it may use adjacent characters for assistance. It then combines its results for all of the characters into a result for a word.

The important feature of this approach is that any string of characters in the correct alphabet can theoretically be recognized. Therefore, it is most useful for applications in which the language of the text is large or unknown. It is also useful in cases where many different languages with the same or similar alphabets must be recognized. However, this flexibility comes with a price: because this approach uses the least information for recognition, error rates are typically higher than for the other two approaches.

2.4.1.2 Whole-Word Recognition

Whole-word recognition algorithms also begin with a word image as input. Unlike character-by-character techniques, however, whole-word techniques usually do not subdivide the word image. Instead, such an approach uses the entire word along with knowledge of the language and data that they have been trained on to match the image with the most likely word in its "dictionary".

These techniques have a much higher accuracy rate than character-by-character techniques, and have been incredibly successful in the fields of automatic mail sorting and check field reading. There performance in these fields, however, is due to the size of the dictionary they are required to keep track of in these cases. These techniques are most applicable to situations where the language of the text that will be analyzed is very small.

2.4.1.3 Word-Spotting

The word-spotting approach is a newer idea that is particularly suited to certain aspects of the Emergent Transcriptions initiative. These techniques are designed to find

keywords in images of full documents. Word-spotting systems are trained extensively beforehand using a variety of single-word images containing the same a keyword. These variations of the same word "teach" the system what that keyword "looks" like. The system is then able to spot likely matches of that keyword in whole-word documents.

This approach is not suited to more typical OCR applications, where single words or entire documents must be parsed. However, it is very useful for finding important words within documents, an application in which it has been shown to perform well (Rath et al., 2004).

2.4.2 Common Requirements for OCR methods

From this analysis, a few common characteristics of OCR methods emerge. First, at some point in the process, all OCR methods require individual word images as input. Second, most OCR methods require a set of "training" data in order to optimize the algorithm for a given application.

Because they all require single word images as input at some point, the system must be able to split up manuscript images into individual words. This process is called word segmentation. Good word segmentation is also an important feature in the Emergent Transcriptions initiative because it makes one of the jobs of the transcriber easier: marking the location of features on the manuscript page. However, the demands for OCR word segmentation are much higher. For a good OCR result, the algorithm must be given the entire word image, including the thinner stems of ascendant and descendant characters.

Training data, the other requirement for most robust OCR systems, will be plentiful once the Emergent Transcriptions initiative becomes popular. For OCR, training data takes the form of word images, each attached to the correct word contained in the image. This is exactly the data that will be provided in digital transcriptions that the system will store.

2.5 Genetic Algorithms

In any system with sophisticated algorithms, there is a need to optimize those algorithms for the specific application in which they will be used. There are many advanced algorithms for pattern recognition and image manipulation that are planned for inclusion in the Emergent Transcriptions initiative, and each of these will likely be difficult to optimize. Optimization can be made easier and faster through the use of a "Genetic Algorithm"(GA).

A GA uses a method which mimics evolution in order to find a near-optimal solution to a problem. Without doing the complex, long, and often unfeasible task of analysis required to derive the optimal solution, a GA can find an excellent solution in a fraction of the time.

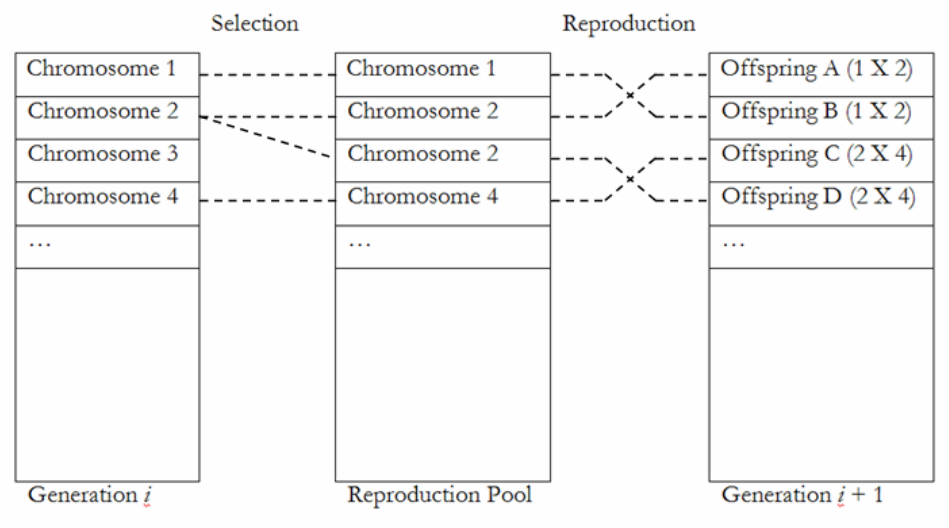

Figure 2.5: Genetic Algorithm: Creating a New Generation (From Whitely)

At the core of the GA is a "chromosome", which is a series of bits that represents a unique solution to the problem. Typically, these are made up of all of the different parameters that determine how a process functions. Chromosomes are the elements in the GA that evolve and improve over time. The purpose of the GA is to find the chromosome that represents the best solution to the larger problem.

The chromosomes in the GA are created in "generations". Each generation is a collection of chromosomes that are created and evaluated at the same time. All of the generations in a given GA typically contain the same number of chromosomes. Each chromosome in the first generation is randomly generated. Each chromosome in subsequent generations is produced by "reproduction" between pairs of chromosomes in the previous generation. Reproduction pairs are selected based on a "fitness value" that is calculated for every chromosome. Both the processes of reproduction and selection in a GA will be discussed further in the following sections.

2.5.1 Reproduction

Reproduction is the process in a GA by which two chromosomes for a new generation are created from two chromosomes from the previous generation. The process

ensures that both new chromosomes are similar to their "parents", but introduces an element of randomness that prevents the GA from halting prematurely at local maxima. The reproduction process consists of two independent steps: crossover and mutation.

Crossover is the basic reproduction step, and is performed in all chromosome reproduction. A random point in the chromosome is selected as the "crossover point". Each child chromosome is given a portion of each parent's chromosome. One child receives a copy of the first parent's chromosome up to the crossover point, and a copy of

Figure 2.6: Crossover and Mutation in a Genetic Algorithm

the second parent's chromosome from the crossover point to the end. The other child receives the reverse: the second parent's chromosome is copied up to the crossover point, and the first parent's chromosome is copied from that point on.

Mutation is an effect which is applied randomly after crossover. It is applied at a pre-set "mutation rate" to a portion of the new generation's chromosomes. Mutation is usually implemented as a single random bit flip. One bit in the new chromosome undergoing mutation is changed from 0 to 1 or vice versa. This introduces additional randomness into the process that allows a GA to find optimal solutions that were not covered by the original randomly generated chromosomes.

2.5.2 Selection Methods

The other key process in a GA is selection, the process by which the best chromosomes are chosen to reproduce. It is the most critical element of a GA because it determines how quickly the GA finds a solution, and its likelihood to get "stuck" at a sub-optimal solution.

Selection is determined based on the "fitness values" of the chromosomes in a generation. The fitness value is a score that represents how well a particular chromosome solves the problem. Although selection is based on this value, it is usually done with an element of randomness, to preserve variety in the population of chromosomes.

The goal of the selection algorithm is to ensure that the set of chromosomes that are selected to reproduce have an average fitness level that is higher than the average fitness level of the generation. However, it must balance against that the need to preserve diversity in the next generation—if the chromosomes in a generation become too homogenous early on in the GA process, the GA will likely get "stuck" at a sub-optimal solution. This problem is called "premature convergence".

Two popular methods for selection are "roulette wheel selection" and "tournament selection". In roulette wheel selection, each chromosome in the current generation is given a space on a roulette wheel proportional to its fitness value. The wheel is "spun", randomly choosing chromosomes to add to the reproduction pool until all of the positions in the reproduction pool are filled. Note that a given chromosome can be added to the reproduction pool multiple times.

In tournament selection, a series of "tournaments" are run among randomly selected sets of chromosomes in the current generation. Winners are chosen from each tournament and added to the reproduction pool. Tournaments are run until the reproduction pool is full. Winners are chosen as follows: the chromosome with the highest fitness is chosen with some probability p. The chromosome with the second highest fitness is chosen with probability $p(1-p)$, the third with probability $p(1-(1-p))$, and so on.

2.5.3 Characteristics of Problems Appropriate for Genetic Algorithms

Genetic algorithms are designed to be used on problems with two features. First, the solution space for GA problems is usually too large to feasibly search exhaustively. Second, the various parameters that define a given solution for a GA problem have complicated relationships that make it difficult to calculate an optimum solution.

3.0 Architecture and Design

The design and architecture of the Emergent Transcriptions system is described in the section below. First the requirements analysis describes the requirements that our work must meet. These requirements are supported by four use cases which show the basic functionality of the Emergent Transcriptions system as well as explaining which requirements each use case implies. Then each feature of the system that we developed is described in detail, including the MML, redesigning of the Transcription Assistant, conversion of the TA to a web applet, the application server development, web front-end and database, text line detection algorithm, and finally the genetic algorithm.

3.1 Requirements Analysis

The features that were implemented for this stage of the Emergent Transcriptions initiative were chosen through analysis of the system's requirements and identifying those which are most critical.

First we decided that the Transcription Assistant stand-alone program must be converted into a web applet. This is done to make it easier to force the user to upload their work once they are done their transcription. Also this allows the creation of a central website for our software where everyone must go through. This will help create a community of transcribers which will in turn give our initiative more material to house.

Next there needs to be a place to store all of the manuscripts and transcriptions submitted, as well as a way to search through the data. To do this a database must be created and searching functionality must be added to the Transcription Assistant.

Some substantial progress had to be made to make Optical Character Recognition (OCR) feasible in the near future. OCR is important because it will allow manuscripts with no previous transcriptions to have at least some keywords identified in them and stored. To this end we needed a way to both order and segment words, or the boxes which represent the words. The current word segmentation needed work before OCR was possible, and the box ordering had to be implemented.

Finally we needed to dramatically improve the Manuscript Markup Language (MML) which is used to pass the information about the transcriptions and manuscripts between each part of the system, as well as store the data.

In order to demonstrate how these requirements fit into the Emergent Transcriptions system, we created some use cases below which show basic functionality of the Emergent Transcriptions system. The use cases which informed this stage of the project are described in the following sections, along with a brief description of the requirements which each implies.

3.1.1 Searching for Manuscripts or Transcriptions

The more information gets stored in transcriptions, the more important search functionality becomes to the user. A user must be able to easily, and at several steps in the process of using the Transcription Assistant, be able to stop and say "Where else can I find information on this?" It is this functionality that allows our system to be able to leverage all of the manuscripts, as opposed to just those that are well traveled and consistently referenced. There are two times a user will want to be able to search through manuscripts and transcriptions, and those are before the user has started working on a document, and while a user is already working on a document.

The functionality of initially searching for a manuscript is taken care of in the web front-end. A user starts his work here with an initial search to start working on a document. A second kind of searching appears in the Transcription Assistant, where a user can search for a manuscript either by its name, or by the metadata and transcription information contained within it.

3.1.1.1 Use Case: Initial Manuscript Search

A historian goes to the website and searches using a given number of fields. The historian could be searching by the file's name, title, or content. From the results the historian picks one manuscript to edit. From there the user chooses to launch the applet with that manuscript, or manuscript and transcription if applicable, by default. The user can specify whether or not he or she wants to work off of an already existing transcription for that manuscript.

3.1.1.2 Use Case: Search for Manuscripts and Transcriptions from the Applet

The java applet should have a menu option to search. This should allow the user to search for either a manuscript, or transcriptions of manuscripts. Results are returned partially to the user. From this limited set of data the user selects either the bare manuscript, or the manuscript with one of the transcriptions to load. At this point the applet loads the

manuscript and transcription if appropriate, and asks the user to save their current work before doing so.

3.1.1.3 Requirements

The most obvious requirement for this use case is that web front-end must be created for the user to launch all of this searching and the applet. Second the Transcription Assistant must be modified to include this new searching functionality. The database had to be designed and implemented, just as many other components of the system required it. Last a server-side process must be created to handle client requests from the Transcription Assistant for this data.

The web front-end has to be created in such a way where the user can select a manuscript. This means it has to be able to communicate with the data base and to provide the user with a means to launch the applet. In its final stages the web front-end should start to become more of a complete web system, with the ability to login as a user, and modify credibility for other transcriptions. The users and credibility is not within the scope of this project however.

3.1.2 Modifying an Existing Transcription

3.1.2.1 Use Case: Searching for an Existing Transcription

A user should be able to search all existing transcriptions for those that match a number of criteria, including author, date, region, collection, and other elements in the metadata, and for elements in the full text of the document. Search results should be returned quickly, and the user should be able to easily browse through them and select interesting ones.

3.1.2.2 Use Case: Loading an Existing Transcription

A user should be able to load a transcription into the Transcription Assistant that other transcribers have worked on. The user should be able to view the work that previous transcribers have done, and modify it. The user should also be able to contribute new elements to the project.

3.1.2.3 Use Case: Uploading a New Copy of an Existing Transcription

When a user has completed work on a transcription, they should be able to upload their changes to the database. Their changes should be merged into the document on the database, and the credibility scores of all the transcribers who have worked on the transcription should be factored into the system's merging process. The system should preserve all changes that have been made to the document, but keep one version of the transcription as the "accepted text" for that manuscript.

3.1.2.4 Requirements

There are two major aspects of this portion of the project. First is how the database handles existing transcriptions and modifications to them. Second is the manner in which the Transcription Assistant represents the same data.

In regards to the database, these use cases clearly state that searching power is paramount in the design of the database. It is not enough for the Emergent Transcriptions database to hold all of the data generated by the system and its users; it must also make it accessible. The metadata and full text of the transcriptions in the database must be stored in such a way that it can be quickly and easily searched.

These use cases also specify what data must be stored by the system. The MML version of a transcription, which is the fundamental data format that will be transferred between the server and the Transcription Assistant, must be able to express the changes that previous users have made to the document. In addition, the system must be able to extract from the change history of the document an "accepted text".

The final element of the database that these use cases detail is how the Contribution Accountant needs to deal with changes to the transcription data. Although the design of the Contribution Accountant will be left for future projects, its functionality will influence how transcription data is stored and represented.

3.1.3 Automatic Boxing and Ordering

For a high-quality transcription to be generated, a transcriber must be able to specify both the boxes which represent words and the order of words in a transcription. Order is also important when extracting a plain text representation of a document from a digital transcription. The previous version of the Transcription Assistant used only the implicit

order of elements found in the MML representation to determine this. Our use cases showed that a more sophisticated system was required.

3.1.3.1 Use Case: Hand-Ordering of Boxes

Use case: A user of the Transcription Assistant will be able to specify not only the boxes which represent words but also the order of word boxes in the transcription. The user will be able to modify the existing order of boxes, or to specify the order if there is no such structure already in the transcription. The user will be able to do this for both automatically boxed and hand boxed transcriptions.

3.1.3.2 Use Case: Automatic Ordering of Boxes

A user of the Transcription Assistant will be able to instruct the application to automatically box words and attempt to order the word boxes in the transcription automatically. The application will use pattern recognition techniques in order to determine the correct ordering of boxes already entered into the transcription. This process will not modify existing boxes. Automatic ordering will be possible with boxes created automatically and by hand, and for boxes that have already had contents entered.

3.1.3.3 Requirements

These use cases imply three major requirements that were not implemented at the time we started our work on the Emergent Transcriptions initiative. First, features inherent in manuscript images that imply an order for the words they contain must be identified. Second, the Emergent Transcriptions initiative requires a representation for those features both in the MML and internally in the Transcription Assistant so that boxes may be ordered explicitly, rather than implicitly by the order in which they are stored. Third, the Transcription Assistant requires a pattern recognition system in order to automatically locate those features and use them to order existing boxes.

It is important to note that word boxes alone do not give enough information to order themselves. Because languages based on the Latin Alphabet are read top-to-bottom and left-to-right, a rough ordering of words can be determined; however, historical manuscripts have a number of features which prevent this information from being used to get a precise ordering of words. Three of the most important of these features are skewed text lines, curvilinear text lines, and uneven spacing between text lines.

All three of these features involve difficulty with locating text lines; therefore, all of them can be dealt with by finding the text lines on a page and grouping the boxes by text line. Once the text lines have been found, it is easy to order them by their vertical position (assuming two text lines never cross), and to order word boxes within the lines by their horizontal position. This ordering, of course, assumes that more complicated page layouts are not used, such as multiple columns.

All of these ordering issues can be addressed by using a hierarchical structure of features that together describe the order of the documents. Words can be grouped into text lines, which can be grouped into paragraphs and other larger structures. For these features to be utilized however, the MML and TA had to be updated to be able to express these features. In addition, the TA needed a system so that it could automatically identify text lines.

3.1.4 Automatic Indexing

The task of transcribing a manuscript is a difficult and time-consuming process. It will be a major impediment to some aspects of the Emergent Transcriptions initiative, as one of the project's goals is to allow the community of historians to rediscover "lost" manuscripts. Rediscovery can only occur if historians stumble upon important information in those manuscripts and take the time to transcribe them. Unfortunately, these manuscripts are unlikely to be found until they have had some data attached to them. This problem is difficult to overcome unless there is a way for the Emergent Transcriptions initiative to automatically index these manuscripts before humans touch them.

3.1.4.1 Use Case: Initial Automatic Indexing

An archivist should be able to upload a manuscript with limited metadata and have the system automatically index it. This should include limited extraction of keywords in the full text of the manuscript using OCR techniques.

3.1.4.2 Use Case: Optimization of Automatic Indexing for Similar Manuscripts

The system should be able to group similar manuscripts by their metadata, and index those manuscripts using techniques optimized for characteristics specific to that group. For

example, the system should be able to optimize the automatic indexing of manuscripts all written by the same hand.

3.1.4.3 Requirements

The main feature required for this use case is an OCR algorithm to extract the text from manuscript images with which the image will be indexed. However, there are two other requirements contained in this use case as well. First is the creation of a database. This is an aspect of many of our use cases, and each impacts different details of its design. Second is a module for automatically optimizing all aspects of the OCR algorithm, either for general-purpose use or for use on specific collections of manuscripts.

We examined the state of the system at the beginning of the project and determined that it was not yet ready for development of the OCR module. OCR techniques depend heavily on good binarization and segmentation of the source image. The binarization and segmentation put in place by previous projects did a decent job of identifying the location of words on the page, but rarely caught all of the features of a single word. A survey of binarization and segmentation techniques revealed that far more sophisticated and accurate algorithms exist (Sezgin and Sankur, 2002).

Creation of a database, however, was such a critical element of so many functional requirements we identified, however, that we decided it was important to design the database at this stage in the project. These use cases suggest some of the data that should be separated from the MML and stored in separate, searchable fields. Our database will require a table with metadata fields for all stored manuscripts in it so that the metadata elements for all manuscripts can be quickly searched for manuscripts that match search criteria. In addition, it will require some method of storing the currently accepted full text of the transcription so that it can be searched as well.

The last requirement is a module for automatically optimizing the various segments of the OCR algorithm. This requirement was important at this stage not only as a part of the final project, but also to evaluate the current condition of the binarization and word segmentation modules. We identified this as a perfect task for a genetic algorithm. This technique not only allows us to optimize the existing algorithms, but also can be used in the final project to compare automatic OCR results to human-entered transcription data. This

will allow the OCR subsystem in the final project to "learn" from the data that human transcribers enter into the system.

3.2 Redesign of MML

A preliminary version of the Manuscript Markup Language was designed for a previous project. The characteristics of that version of MML are described in 0. We decided that in light of the new requirements and use cases that the MML needed a significant overhaul.

3.2.1 MML Design Process

We identified four major things that the new MML had to be able to express. First was a wider variety of features. Second was a way to express the full document history. Third were metadata elements integrated with the MML. Finally, the new MML needed to enable transcribers to make notes about any and all aspects of a transcription.

3.2.1.1 Expressing a Wider Variety of Features

The new requirements we found for this stage of the project included requirements for extracting the transcription text from an MML document. This implies two other requirements: first, the MML must contain enough information to allow the system to extract an order for the word boxes in the MML. Second, the MML should be able to express what parts of the document are text, marginal notes, images, or other types of information.

The problem of ordering could easily be solved by defining higher levels of document features above word boxes. Identifying the order of arbitrarily placed words on a page is difficult; however, the human brain is able to determine the reading order of a page by grouping words into text lines, paragraphs, and columns. Each of these elements is easy to order within the context of the next larger element. For example, ordering words is difficult given just their position, but once they are grouped into text lines, their horizontal position is sufficient to determine the order of a text line, and the order of the text lines can be determined by the vertical position of the boxes it contains.

Larger, nested document features also solve the issue of expressing the characteristics of each element on the page. The previous version of MML allowed the user

to define the class of each document element, for example, an image, stricken text, text in another hand, marginal notes, and so on. However, the old MML only allowed a user to do so on a box-by-box (and therefore, word-by-word) basis. This method is counterintuitive, as typically entire lines or paragraphs carry these traits. Expressing larger document features as groups of those boxes, and allowing traits to be set for the larger elements would save storage space on the server and transmission time when distributing MML documents. It would also make it easier to change the characteristics of document features, as each change would only have to be recorded in one location.

3.2.1.2 Expressing the Full Document History

A number of the requirements we identified suggest that the MML needs to record a much larger portion of a transcription's history. This is an important part of the new MML because the new requirements for the Transcription Assistant and Contribution Accountant specify that these two modules must be able to examine the document's history. Since MML is the form of the transcription that will be moved around the system, it must be able to express this information.

Further decisions had to be made, however, beyond the simple fact that history must be included. We also had to decide what elements would have their history recorded. Keeping track of all the changes in all elements could potentially become very expensive, as even a 1-pixel change in the location of an element would have to be recorded in the document's history, and would be a part of every version of the transcription from then on. For this reason, we decided to record only the changes in the text contents of transcription elements, not changes in the location. This decision was further supported by the fact that the location of features is likely to be the subject of far less debate than the contents or significance of those features.

When recording the history of a feature, two things are vital: recording all of the proposed values for that feature, and recording the community's opinion on those features. The community's opinion can be encapsulated in the element "credibility scores" that have already been discussed. Each of the submitted options for an element can be recorded, attached to its credibility score. We also decided to add to this one additional piece of data: each option would also be connected to the original transcriber that submitted that proposal. This last decision both streamlines the process of modifying the credibility of users when a

change is made or confirmed, and provides a way for a transcriber's work to be recognized, and therefore for his reputation to grow within the Emergent Transcriptions community.

3.2.1.3 Integrating Metadata into MML

While editing the transcription of a document, it is likely that transcribers may wish to edit the metadata attached to that element. In addition, because of the requirements for searching metadata in the database, we decided that the metadata needed to be more accessible, rather than embedded in the manuscript image's data. For this reason, we decided that the new MML should also contain the metadata information for a manuscript.

3.2.1.4 Enabling Transcriber Notes

We realized that a transcriber might wish to provide an explanation for a given interpretation or to provide additional information about certain features in a manuscript. Therefore, we decided that the new MML should allow a transcriber to write a note about any feature in a document.

3.2.2 New MML Structure

The above design decisions were all combined and produced a new design for MML. The new MML consists of three major pieces: the elements that describe the structure of the document, the elements that contain the document's metadata information, and the elements that contain information about all of the transcribers that have worked on a transcription. The way these elements fit into the larger MML structure is shown in Figure 3.1.The structural details of the new MML standard are described in the following sections.

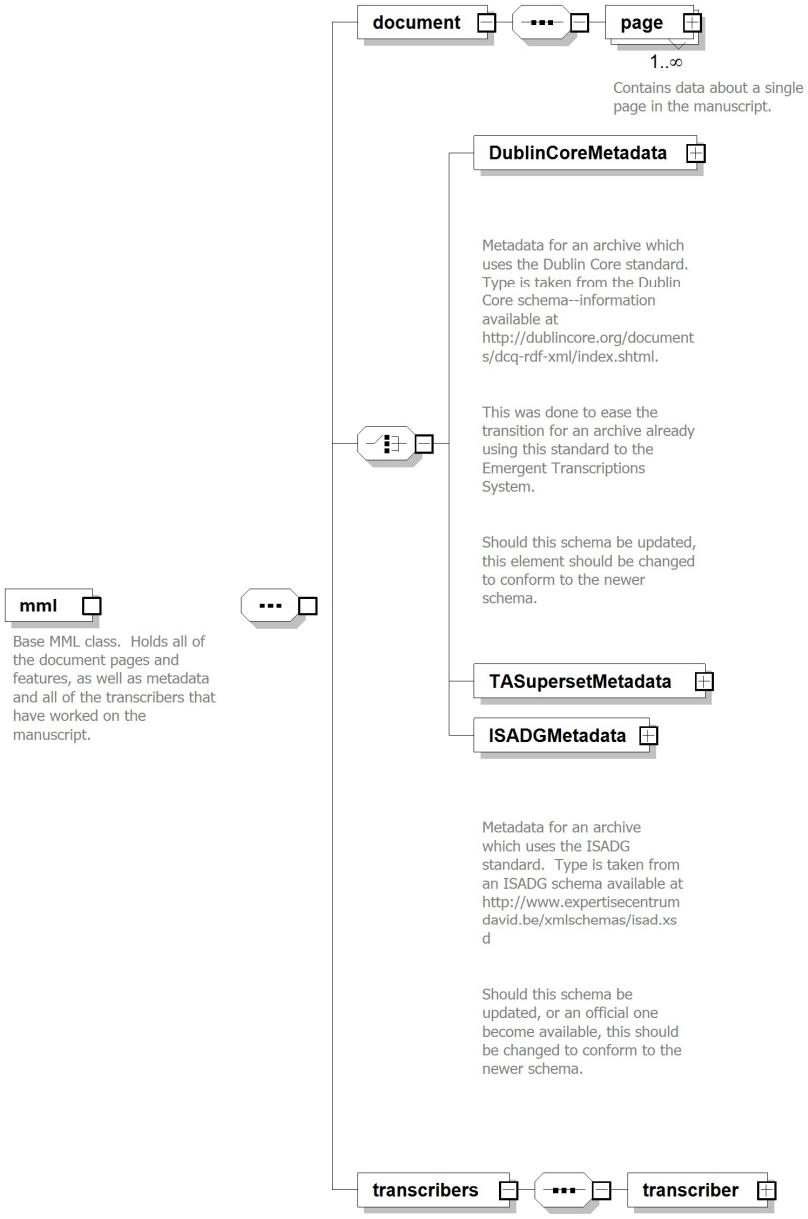

Figure 3.1: Overview of the New MML Structure

3.2.2.1 Document Feature Elements

The first portion of the new MML structure are the document feature elements, which describe the layout of a manuscript and its contents. These elements are all under the "document" element in the MML hierarchy. The document element can be made up of any number of pages (but must include at least one). Each page has a single manuscript image and page number associated with it. Other than that, the page follows the general structure for a "genericFeature", the base element type for all features found in a manuscript.

All manuscript features have been constructed as extensions or restrictions of the basic genericFeature type in order to provide all document elements with the ability for nesting of features. In addition, the genericFeature type allows all manuscript features to have their content type as a common attribute in each feature. This allows the user to define characteristics such as "other hand" for paragraphs as well as boxes. The genericFeature type also defines a "notes" element for transcriber notes as a part of each feature. This allows transcriber notes to be entered for any feature in a manuscript. The genericFeature hierarchy for page elements can be seen in Figure 3.2.

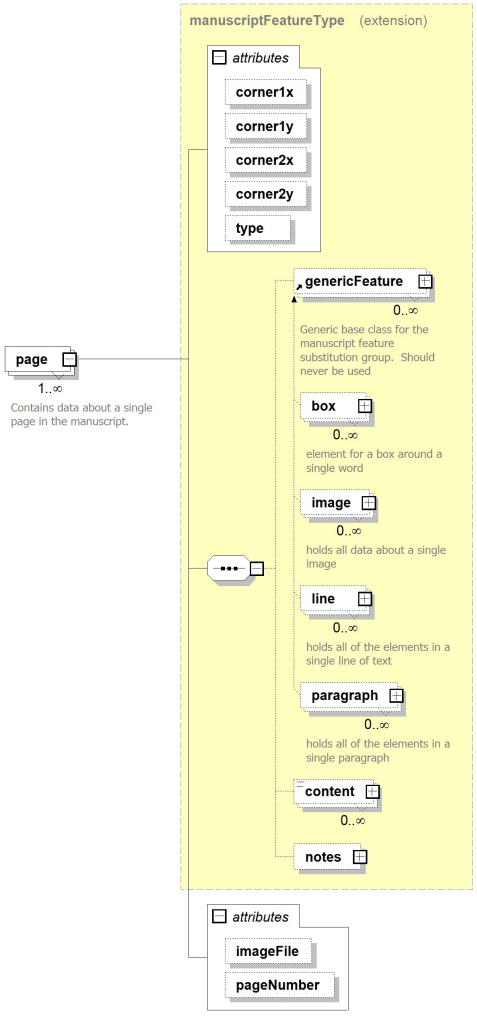

Figure 3.2: Page Element

Since all of the elements inherit from the genericFeature type and can be freely exchanged for each other, the MML schema does not define a rigid hierarchy of features. For example, the user is able to define text lines that contain paragraphs. It has been left up to the creator of software that uses MML to preserve these relationships rather than defining them within the schema for two reasons. First, doing this allows for easy substitution of

lower-level features where higher-level features would ordinarily be used. For example, word boxes can be entered as direct children of a page, rather than having to define a paragraph that contains a line that contains those boxes. Second, this method of defining document features makes it easy for future developers to easily and quickly add new features that we may have overlooked in designing this schema. A developer need only to define a new type that extends or restricts the genericFeature class and is included in its substitution group in order to add it to any MML document.

Most of the features that inherit from the genericFeature type are relatively uninteresting. Paragraphs and lines are relatively uninteresting features, as they serve only to suggest the structure of the word boxes they contain. They are restrictions of the genericFeature type that only allow for child elements to be defined beneath them, and only use the style attributes, rather than all of the style and permissions attributes.

Box elements, however, use all of the remaining genericFeature elements, and do not allow manuscript features to be defined as children. Instead, they have an unlimited number of "content" elements as children. Each content element defines a single suggestion by a transcriber for the contents of a word box. Each content option has attributes for its credibility score and transcriber. This will allow the TA to display that information when offering other suggestions for a word in the transcription.

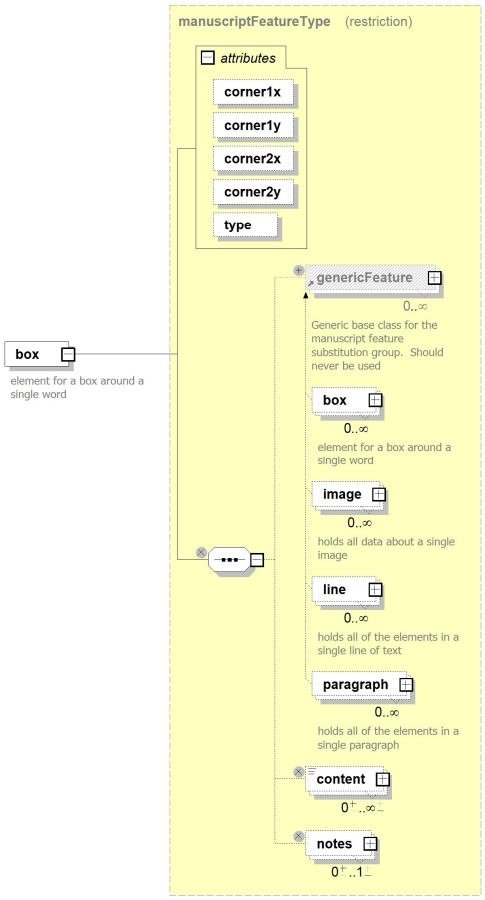

Figure 3.4: Box Element

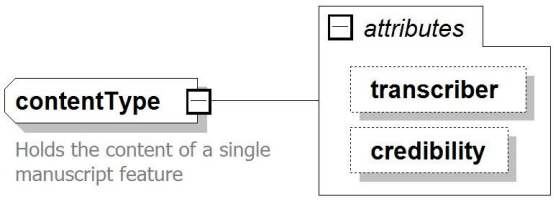

Figure 3.3: Content Elements

The box element and the content element can be seen in Figure 3.4 and Figure 3.3.

3.2.2.2 Metadata Elements

The next portion of the new MML standard is a set of metadata elements. Since archives may already have metadata sets that they use for their own collections, we decided not to design the metadata section ourselves. Rather, we allowed the MML user to decide on one of three internal MML representations. One of those representations is the TA superset metadata set designed by the 2003 MQP group (2003 MQP). The other two are accepted standard representations for both the Dublin Core and ISADG metadata sets. In addition, the MML definition makes it easy to add other metadata sets to these three. The inclusion of these standard metadata schemas should make an archives transition to using the Emergent Transcriptions initiative easier.

3.2.2.3 Transcriber Elements

The final portion of the new MML standard is data on all of the transcribers who have worked on the transcription. This portion is intended to record all of the users who have contributed to a given transcription. The transcribers section consists of a list of transcriber elements, each of which gives the name of the transcriber and the dates that he or she worked on the project. In the future, this element may be updated to also include information such as the organization of the transcriber or contact information. The structure for the transcriber element can be seen in Figure 3.5.

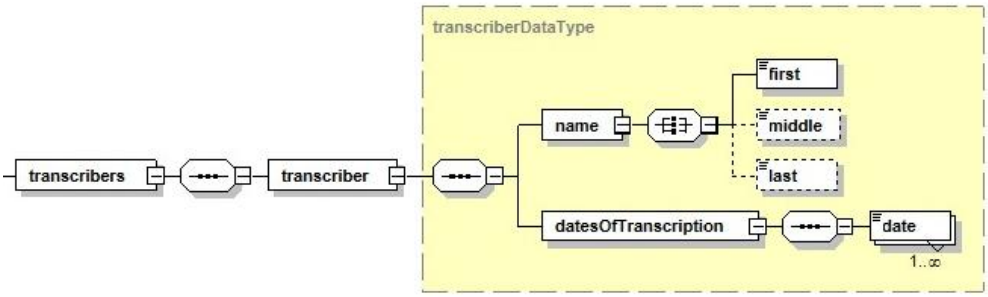

Figure 3.5: Transcribers Element

3.2.3 Internal Representation

Older versions of the Transcription Assistant recorded the boxes in a document in a list structure. Each box had an index in the list that was used to identify that particular box. Every module that made changes to the boxes or retrieved information about the boxes used the indices of boxes to refer to them. Therefore, all of the modules in the old Transcription Assistant that worked with the boxes knew that they were stored in a list. Unfortunately, the list method of dealing with boxes does not work as well with the new way that MML models manuscript features. The new MML is more accurately represented as a tree. Therefore, the new representation of manuscript features internally in the Transcription Assistant needed to change to reflect the changes in the MML.

3.2.3.1 The ManuscriptFeature Class

The new internal model for document features mimicked the new MML model for features in the way in which all features inherit from a generic feature class. In the internal representation, all features extend the "ManuscriptFeature" class. There are a few features of the ManuscriptFeature abstract class. First it follows the "Composite" design pattern in order to enable the nesting of features. Second, it defines the basic structure for hierarchical style definitions—that is, the system by which style definitions percolate down to the features which they contain unless those features redefine the same attributes. Third, it defines the basic interface for all features—they know their position and size, they can draw themselves, and they can convert their contents to an MML representation.

3.2.3.2 Page, Paragraph and Line Classes

Each of the new MML feature elements has a corresponding ManuscriptFeature subclass. In the case of the Page, Paragraph and Line features, these are relatively simple. They do not override any of the ManuscriptFeature non-abstract methods, and therefore act primarily as containers for other features. The Page class has one additional field that a ManuscriptFeature does not: a filename for the picture that contains the image for that page.

3.2.3.3 BoxBoundedFeature class

The Paragraph, Box, and ImageBox classes all have one thing in common: they can all be defined by a box around an area of the image. In order to reduce the amount of code that is rewritten, they therefore all extend the BoxBoundedFeature abstract class. This class implements the position and size methods from the ManuscriptFeature class to work with a rectangle defined by two points. In addition, it implements the draw method to draw the bounding rectangle around the feature it represents.

3.2.3.4 Box class

The Box class is more sophisticated than most of the other new ManuscriptFeature subclasses. It does not override the draw or position methods, because it extends the BoxBoundedFeature class. However, it does override all of the methods pertaining to updating the list of a feature's children, as boxes may not have child features. Instead, a Box has a list of content elements as one of its members. Each entry in this list contains one contents item, which corresponds to the contentType in the MML schema.

3.3 Redesign of Transcription Assistant Backend

The changes to the internal representation of MML are described in 3.2.3. However, these changes caused a series of changes to be required through a cascading chain of dependencies. In many cases, we not only fixed the problems caused by the changes, but also redesigned the affected classes in order to reduce the large number of interdependencies in the code. Those changes should allow future updates to the code to avoid massive changes to multiple classes. The changes can be broadly divided into four categories: changes to the internal model of MML, which have already been described, and changes to importing and exporting MML, editing manuscript features, and GUI elements.

3.3.1.1 Importing and Exporting MML

Updating the method by which the Transcription Assistant imports and exports MML files involved three portions. First, a XML parser was written for MML to extract data from an MML document. Second, a means to obtain the MML from the internal representation was created. Finally, a means to store that MML data was written.

We only implemented a portion of the XML parser. Since the transcriber and metadata portions of the MML are not fully fleshed out, we chose not to write a parser for these portions of the MML document at this point. Instead, we focused on parsing the document elements of the MML, which are perhaps the more important and more difficult elements to parse.

We chose to use a SAX (Simple API for XML) parser to do this task. The more traditional method of XML parsing is DOM (Document Object Model), which converts an XML file to a series of intermediate objects in a tree structure. In the case of simple structures, using a DOM implementation can add a significant memory usage overhead. The Transcription Assistant already had periodic heap space issues, so we chose a lighter weight SAX implementation instead.

We added the class FeatureParser to the Transcription Assistant to handle the parsing of the document features. This class uses the XercesJ SAX parser to convert MML directly to the ManuscriptFeature objects that will represent it internally. The MMLReader class contains the main parsing method, readMyFile(), which splits the MML into its three sections, and sends the document feature portion to the FeatureParser class. In order to make the parser as flexible as possible, the parsing methods all take Readers as arguments. This allows the calling method to extract MML data from a file, string, or other data source, and submit it to the parser without the parser ever noticing.

3.3.1.2 Editing Manuscript Features

The old Transcription Assistant had methods for modifying boxes scattered in many different locations, including within GUI classes. We consolidated the feature modification code into a few more focused classes. In the process, we also overhauled the system for selecting features, and created a new virtual clipboard for ManuscriptFeatures that can be used throughout the Transcription Assistant.

In order to consolidate those methods which modify the ManuscriptFeatures in a document, we created a new FeatureEditor class as a façade for the more low-level methods in the internal representation of the ManuscriptFeatures. The new FeatureEditor class took methods that were previously in the ScrollingPicture GUI class that contained non-GUI logic.

In addition, we redid the selection and added a clipboard system. Previously, the Selection class kept track of a selection by the index of the first selected object in the box list and the last selected object in the box list. Since the box list no longer existed, this had to be changed, but we chose to improve it as well.

The new Selection class is paired with a Selectable interface. A selection may only contain classes that implement Selectable. Each Selection item has a unique string identifier which allows multiple Selections to be active at once. A Selection contains an internal list of all of the Selectable items that have been added to the selection. Each Selectable item in turn keeps track of the identifiers of all the Selections which contain it. Selections can be checked to see if they contain particular elements, and elements can be checked to see what selections they're part of. Adding or removing a Selectable object to or from a selection updates fields in both objects.

The new implementation for the Selection class allows all of the previous selection functionality, but also adds the ability for non-continuous selections. Any set of features in a manuscript may be selected at once with the new selection system.

The new clipboard system is just temporary storage for document features. It contains an internal list of the contents of the clipboard, and allows other modules to add, remove, and check on values in the clipboard. This will allow for easy implementation of cut, copy, and paste in the future.

3.3.1.3 GUI changes

The changes in the internal MML representation also affected the GUI for the Transcription Assistant as well. All of the GUI features for editing manuscripts were rewritten from scratch. In doing this, care was taken to remove as much of the logical methods contained in the GUI classes as possible.

3.4 Conversion of Transcription Assistant to Web Applet

As it stood, when a user made a transcription using the TA, they saved the file locally. For them to upload their transcription to the server was an extra step and an inconvenience. On top of this, since the files were being saved locally using the program, there was no way to force a user to upload his saved transcription. To fix this loophole in our system, we decided to convert the TA from a java program into a java web applet. The

user works on his or her transcriptions and upon completion, the work done is automatically uploaded to the server. Putting the TA on the web also makes it more accessible and easier to try out than having it as a separate application.

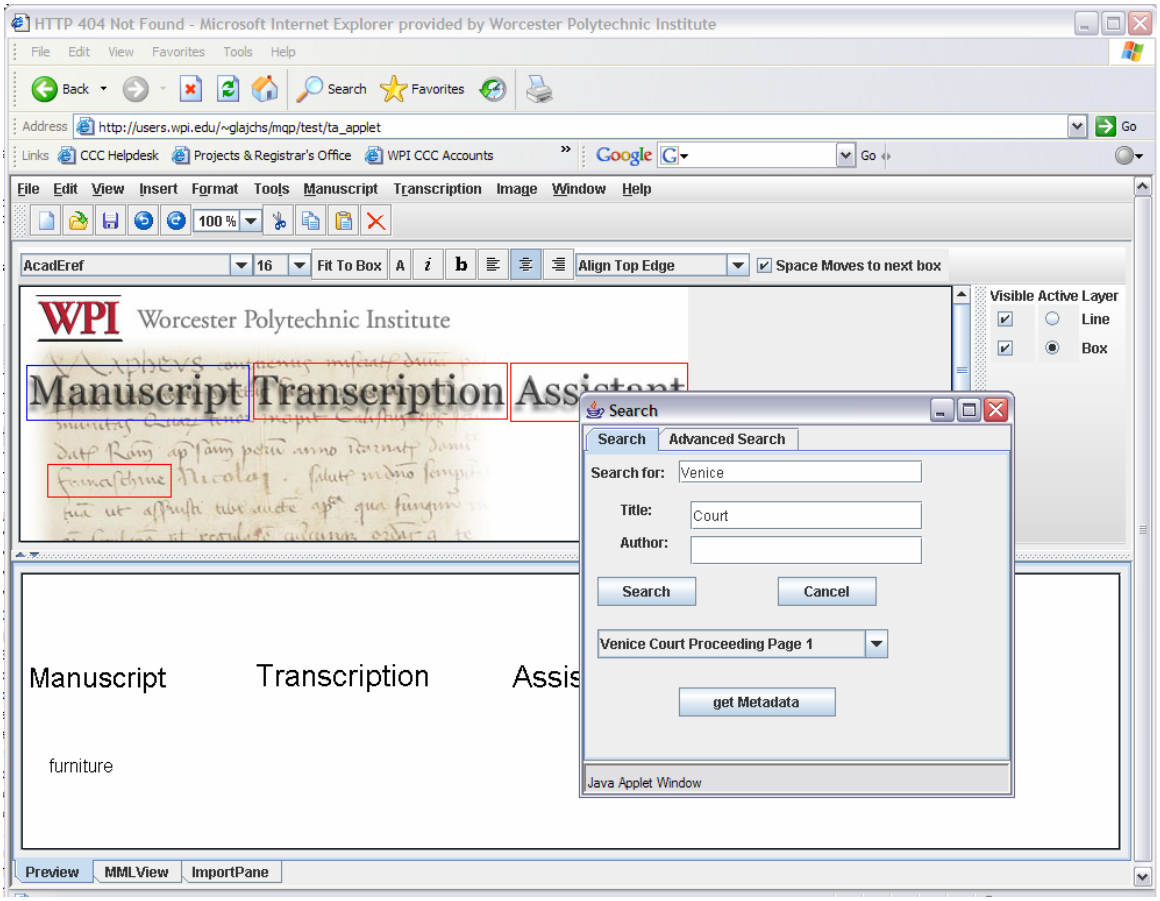

Figure 3.6: Transcription Assistant Web Applet

First we had to consider the feasibility of converting such a large program into an applet, because there are several things that an applet cannot do that a regular Java program can. One of these limitations was that an applet cannot access files on the local disk, or in other words the client connecting to the web page cannot access files on their local disk. It is still possible for the server to give files to the applet. This presented immediate issues because the way the program was designed, instead of keeping the image and the related data in the program at all times, it wrote out the image and the data about the image to temp files many times. This required a fair amount of rewriting code, but was the first thing issue with converting to the applet that had to be handled.

The program also loaded many image files required for the icons in many different locations of the program. This also presented a problem because even though an applet can load images from the server, it has to do so in the main applet class. This meant a restructuring of the way the classes passed hierarchical information from and to each other, to allow classes further down the chain to access the main applet class so they could load their images. Many of these issues stemmed from the fact that the GUI and the rest of the implementation of the program were intertwined unnecessarily. So if a toolbar needed some image icons, it had to have a reference to the parent panel it was created from, and that panel had to have a reference to its parent, something another pane or panel, etc.

Finally, there was some functionality that had to just be scrapped altogether when using the applet, and that was the saving and loading of images and MML files from disk. This was acceptable because we were instead creating a way to download the images and MML data from a server and then upload it back to the server in place of the traditional saving and loading. It did mean that until both the server and applet were fully functional, the applet was always going to have less functionality than the original program we started all of this off of.

There are also strict limitations as to how an applet running in a client's browser can connect to other computers. More specifically the applet can only connect to a server which is located on the same machine that the applet is being hosted off of. While this limits our current system to requiring the web server and application server to being on the same machine, we feel that this is not an unreasonable requirement.

All of these were considerations that we had to take in our decision to convert the program into an applet, but in the end doing so created a more flexible way to distribute our system. This change also gave us the opportunity to have a more concrete way to ensure that the client uploads their changes after they have finished, since they work they have done is not actually saved on their local disk at all. This makes it much more feasible to integrate emergence into our transcription process.

3.5 Application Server Development

The application server is a general purpose server written in java and designed to be run from command-line on the server where the applet is being hosted on the web. Right now the applet's primary functionality is to take and handle search requests from the client,

and return information about the searched, as well as transfer to the client any pictures or MML data needed to load up a specific transcription searched on. This is done through a standard Java socket creation. The server currently listens on port 6060 by default, and the client (applet) creates a temporary socket to connect to the server and obtain the transcription information needed.

The server was designed in such a way that any number of commands and data can be sent back and forth from it, so that when the server or applet is extended later on, other commands can be implemented. The biggest command which should be put in next is one to request an automatic boxing for the image that the user currently has open. The reason this is not currently in the program is because the server does not contain the code currently used to autobox a picture, and some sort of adapter must be implemented. While both Paul and Scott worked together on this project, usually the autoboxing code and the applet and server code were being worked on separately and simultaneously, so this feature was labeled as less important than those that made these separate components function by themselves.

3.5.1 Client/Server Architecture

For the applet to both receive and send image and mml files, there needs to be a process running on the server to handle these requests. This server, the *ta_server* branch, opens a socket listening on port 6060 and waits for incoming connections. Upon receiving a connection it loops through receiving text commands from the client, which is the applet in this case, and processes each command received accordingly. After the server processes one text command it loops and waits for more, until it receives a disconnect command, at which point it disconnects the client from the socket.

Each command consists of the command name and a parameter. Currently there are a number of commands which the server recognize. *getImage* takes one parameter, the image name, and loads that file into memory, and then sends it to the client. *getMML* takes on parameter, the mml file name, and loads the mml into memory, and then sends it to the client. s*earch* takes a parameter string, which is a list of parameters in the form fieldname1=value1&fieldname2=value2. The database is then searched according to what fieldnames were provided, and a list of results is sent back to the client. *autobox* is an unimplemented command will work much like *getMML* but instead of returning an existing

boxing it will return a boxing produced from the autoboxing algorithm on the server side. *disconnect* removes the client from the socket.

These commands are all parsed in the same manner, to make the server application easily extendable in the future. This server is also where extra processing will be done during down time to help enhance the autoboxing algorithm.

3.6 Web Front-End and Database

For the web applet to be useful we needed a web presence from which the user could pick his or her files to load and work on, as well as use to search for images and files. We created a simple web front-end using Perl, HTML, and JavaScript which allows the user to search for a transcription or for words within transcriptions, and retrieve those transcriptions to work on inside of the web applet. The user then launches the web applet with the transcription or image that they select.

To search from and store into information about the various manuscripts and transcriptions we created a basic mysql database, using the mysql server located on campus. There is a table for each unique image file, containing the name of the image file as well as the person who originally uploaded the file, the date of upload, and a non-unique title name to be used for the image.

There are a series of metadata tables, one for each type of metadata standard used. Each table contains the necessary fields required for that standard, as well as a field saying whether or not this standard is currently the default for the specified manuscript. Each table also contains the unique filename that the metadata is linked to. Note that this allows for multiple sets of the same type of metadata for each transcription, which allows the Contribution Assistant to keep records of the updates made to the metadata, as well as selecting which set is used by default based upon the credibility of each set of metadata.

Then there is a table to store the remaining MML information for each transcription. Again each set of transcription data is linked to the unique filename that the image has, to tie it to that image, and again this system allows for multiple sets of transcription information, so that the Contribution Assistant can do its job in scanning through to find which data it deems to be the official data for any particular manuscript.

3.6.1 Database Architecture

The database used to store the manuscript information was the MySQL database provided on campus. The basic organization of the database is centered on one table, *manuscripts*, which stores a unique id name, and unique filename, author, organization, and description for each manuscript. The last three fields are optional and only exist for easily obtaining basic information about the document from searches. Linked to this are ten other tables, all containing metadata information about the manuscript, and all of which link back to a unique manuscript image filename. These tables are *dublin1*, *dublin2*, *dublin3*, *dublin4*, *dublin5*, *meta_archive_information*, *meta_image_format*, *meta_manuscript_information*, *meta_ownership_information*, and *meta_related_information*.

While these tables created an efficient way to store manuscripts and their metadata, Emergent Transcriptions runs off of leveraging several transcriptions per manuscript, and keeping track of each transcription and its associated credibility score. At the base of storing each transcription is the *transcriptions* table, which links the different aspects of each transcription together. Each entry has a unique id, as well as a unique manuscript filename which serves as a secondary index. Using the manuscript filename as a secondary index links the manuscripts and transcriptions table together for easy searching. Also an *mml_filename* field is added to optionally store the name of the file where the MML file for that specific transcription is stored. This *mml_filename* does not have to be unique, as multiple uploads of transcriptions can use and update the same MML file. The table also contains the author's name who originally created the transcription and the timestamp for when it was created, as well as the author's name who most recently modified the transcription and the timestamp for when it was uploaded.

To allow for searching for fields within each individual transcription, a table must be created for each manuscript. Thus when a manuscript is first uploaded, a table is created with the name of the table being the name of the manuscript image file. This ensures that each table is unique and that there is room for the table to be created. Within each of these tables there are elements, which are just boxes for now. Each element contains the data about it, who created it, who last modified it, the credibility for that specific element, and the transcription from the *transcriptions* table that it is linked to.

This database structure is very in depth and stores all of the data contained in the mml in a way that makes it easy to run search queries on. It still requires both the mml and

image file to be referenced as file names, so that sending both the image and mml file representing the transcription back to the client a simple task. If the mml file was not also separately stored on disk, it would have needed to be recreated every time the user wanted a transcription, which would put a very large unnecessary load on the database and server.

3.7 Text Line Detection

Our approach to detecting the reading order of words in a manuscript document was to find text lines in the manuscript image. We developed a proof of concept for this technique using a line detection algorithm based on what is called the "Vertical Projection Profile". The algorithm examines fixed-width columns of the entire image, and locates Partial Segmentation Lines (PSLs) at locations likely to be the bottoms of words. After finding all the PSLs for all of the columns, the algorithm joins PSLs across the entire page into word line borders. This process is described in greater detail in the following sections, including our own implementation.

3.7.1 The Partial Segmentation Line Algorithm

There are two major steps to this algorithm, as defined in Pal and Datta (2003). First is locating Partial Segmentation Lines (PSLs) in the document image, and pruning redundant PSLs. Second is joining those lines across the page to create text lines. Each of these steps will be expanded in the following sections.

3.7.1.1 Finding PSLs

The algorithm begins by splitting the black and white (not greyscale) source image into n columns of equal width. The n^{th} column may be narrower than the others, if the column width chosen does not divide the width of the source image evenly. Each column is then scanned from top to bottom searching for PSLs.

This process is done as follows. The algorithm begins at the top of a column in the document, and counts the number of black pixels in each scan line from the leftmost edge of the column to the rightmost edge. If the number of black pixels on a line in the column is zero, the algorithm marks a PSL at that location. It then ignores all lines containing zero black pixels until another line is found with black pixels. At this point, the scanning process continues as normal.

After all of the columns in the document have been scanned in this manner, the algorithm prunes from its list any PSLs that are too close together. It does this by finding the statistical mode of the distances between two adjacent PSLs in the same column, and removing the lower PSL whenever two PSLs are found closer together than this threshold. The leftover PSLs, which are all separated from their neighbors by at least the mode distance between PSLs, are the final accepted PSLs.

3.7.1.2 Joining PSLs

The next stage of the algorithm joins the final PSLs together across the page into text lines. It does this in two passes: one from right to left, and another from left to right. The algorithm starts in the rightmost column and searches the adjacent column for a match for each PSL in the current column. A matching PSL is one that is closer than the mode PSL distance calculated earlier. If there are no matching PSLs, the line is extended from this PSL horizontally across the next column. This process is repeated for every column moving right to left, then the results from that scan are joined in the same manner from left to right.

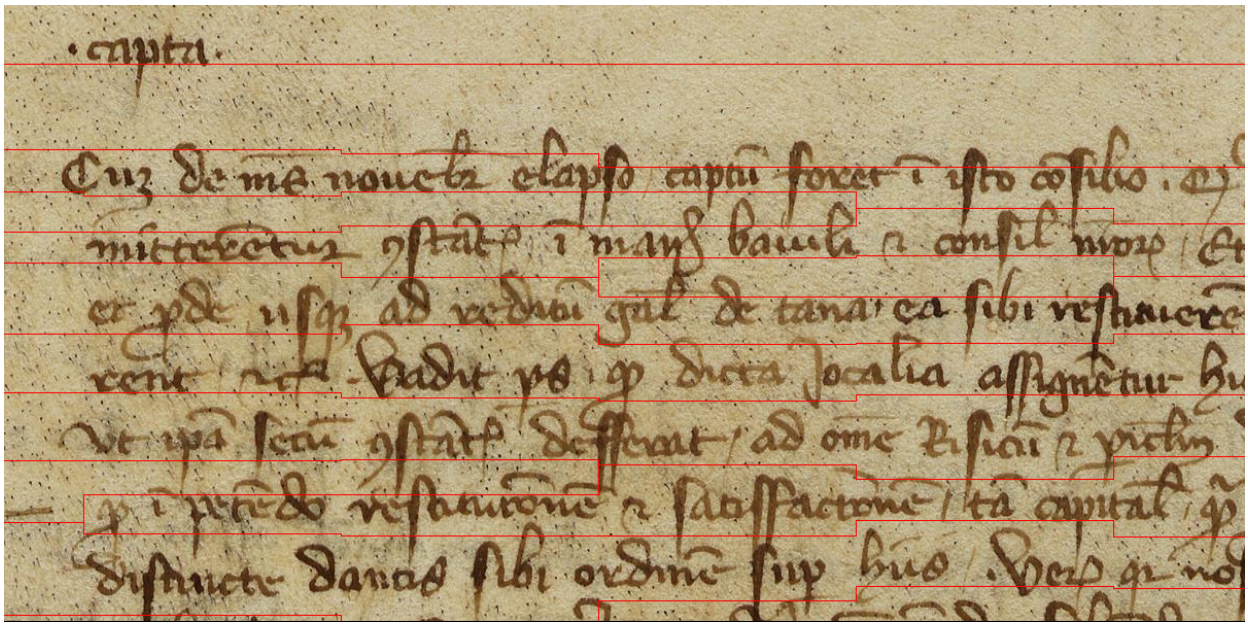

Figure 3.7: A Result from the ProjectionLineSegmenter

3.7.2 Our Implementation

This section describes our implementation of the algorithm for the Emergent Transcriptions project. In general, the proposed algorithm was followed, and was split into

three classes: the ProjectionLineSegmenter, ProjectionColumn, and LineSegmentationBorder classes. Each of these is described in the following sections.

3.7.2.1 ProjectionLineSegmenter

The ProjectionLineSegmenter class is the top-level class that coordinates the other two classes in the implementation of the algorithm. It takes all of the information about the source picture, and all of the parameters for the algorithm. It then locates the text lines in the document when the method findTextLines is called. The ProjectionLineSegmenter uses a set of ProjectionColumns to find the text lines, and stores the located text lines as a list of LineSegmentationBorder.

The result from that call is stored within the class, and it can be used to order a set of boxes by handing a ManuscriptFeature root element with boxes in its hierarchy. A call to orderBoxes with a manuscript, for example, will modify that manuscript, adding line elements around each set of boxes that match the text lines found by findTextLines.

3.7.2.2 ProjectionColumn

This class is responsible for locating and pruning PSLs in a single column of the manuscript image. It is initialized by the ProjectionLineSegmenter to be responsible for a given area of the image, between the column's right and left bounds, and then a call to findPSLs initiates the process of locating the PSLs in that column. After findPSLs has been run, the ProjectionLineSegmenter retrieves information about them to calculate the mode by calling getAllPSLDistances. The ProjectionColumn prunes its list of PSLs when the ProjectionLineSegmenter then uses setMinPSLDistance to set the minimum distance between any two PSLs to the mode of the PSL distances. During the joining phase of the algorithm, the ProjectionLineSegmenter requests matching PSLs in each column by calling getConnectingPSL. The ProjectionColumn keeps track during this process of those PSLs that have been matched to adjacent column PSLs.

3.7.2.3 LineSegmentationBorder

During the joining step, PSLs are joined into the LineSegmentationBorder class, which holds information about a single text line that has been found in the image. All of the LineSegmentationBorders are then stored in the ProjectionLineSegmenter class until a call to orderBoxes is made. Each of the boxes found in the manuscript passed to that function is

compared to each LineSegmentationBorder using calls to the LineSegmentationBorder method getRelativeBoxLocation. Successive calls to that function allow the ProjectionLineSegmenter to determine what text line each box should be placed in.

3.8 Genetic Algorithm

In order to optimize the existing word segmentation algorithm, and to optimize other future elements of the OCR subsystem of the Emergent Transcriptions initiative, we designed and build a module that allows Genetic Algorithms to be built for many different applications. The module was built at this point for optimization of word segmentation, but it was designed with flexibility in mind. A developer should be able to easily adapt the genetic algorithm to many other tasks.

3.8.1 Genetic Algorithm Architecture

The core of the genetic algorithm can be broken up into 4 basic classes: the Allele abstract class, the Chromosome class, the GeneticOrganism abstract class, and the GeneticAlgorithm abstract class. Each of these components contains the basic functionality for one piece of the system. The abstract classes are extended by concrete classes to provide the functionality required for the specific application that the genetic algorithm is being used for. In the case of the word segmentation application, the genetic algorithm has additional classes for IntAlleles and DoubleAlleles, which inherit from the Allele class; and GeneticBoxOrganisms and SmearBoxOrganisms, which inherit from the GeneticOrganism class.

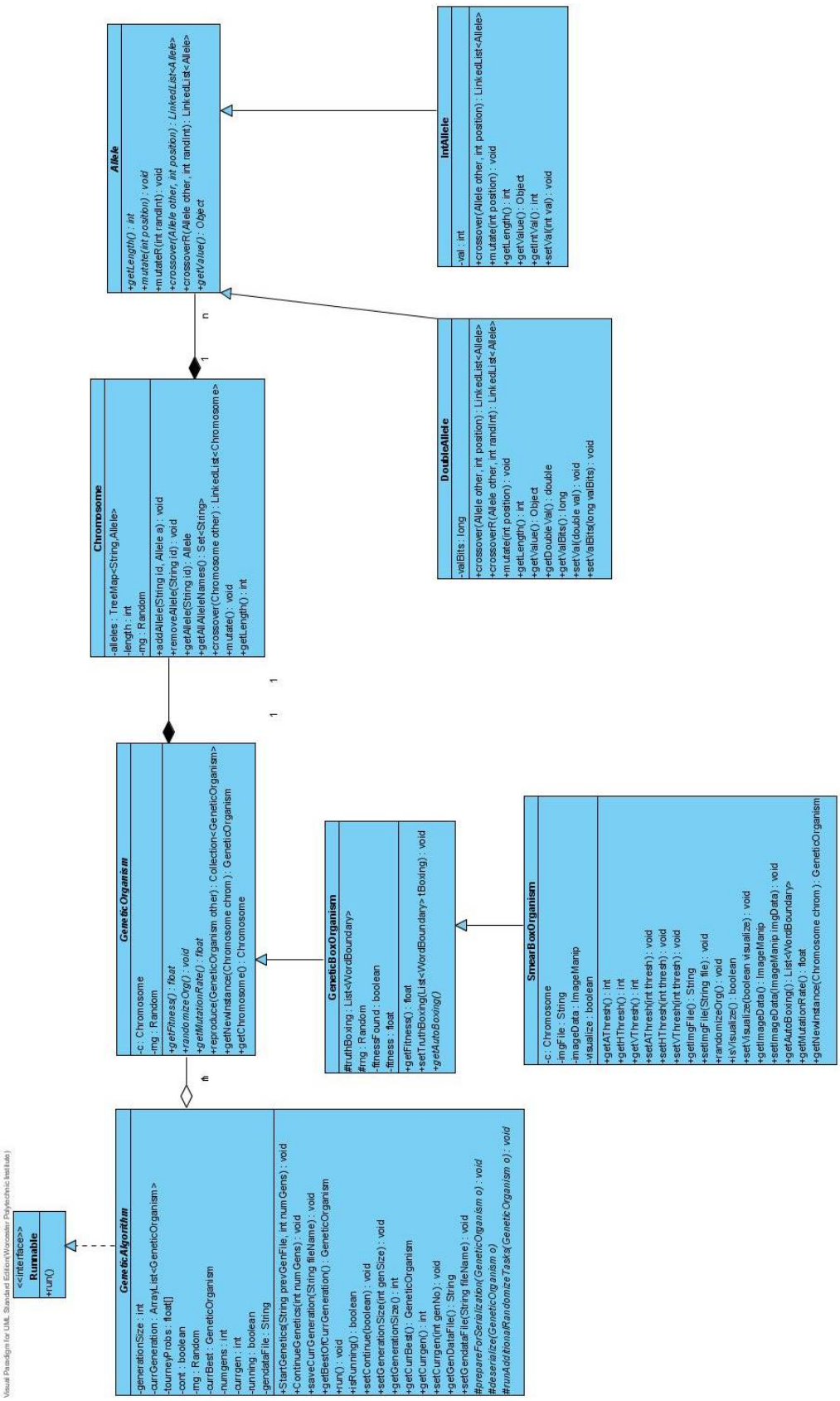

Figure 3.8: Class Diagram for the Genetic Algorithm Subsystem

3.8.1.1 Allele Classes

The allele classes hold single values for parameters in the Genetic Algorithm chromosome. The Allele abstract class makes the underlying structure of the value appear to be a simple binary string that can be easily crossed with other alleles and mutated. However, the Allele class hides the actual implementation of the individual alleles and allows for more complicated representations behind the scenes. We have written two classes that inherit from the Allele abstract class, and demonstrate both a simple implementation and a complicated implementation.

The first class that inherits from Allele is the IntAllele class. This is a basic implementation of an Allele subclass, and therefore only implements Allele's abstract methods, and does not modify the concrete methods in Allele. Crossover and mutation are accomplished via the mutate and crossover methods, which give a position between 0 and the length of the allele in bits (in this case, 32) for the operation to take effect at. This means that for an IntAllele, mutation and crossover are uniformly distributed amongst its bits.

The DoubleAllele concrete class is a more complicated Allele subclass. There are two major issues with the use of double values in a Java genetic algorithm that require this more sophisticated implementation. The first is a limitation of the Java language: java does not allow the bitwise operators that are necessary for crossover and mutation operators to be used on floating point or integer numbers. For this reason, the DoubleAllele class stores its data internally as a long representation of the bits that make up a double. A static function in the Double wrapper class allows for easy conversion between that representation and the double representation.

The other issue is a universal one when dealing with floating point numbers in a genetic algorithm. It has to do with how floating point numbers are represented in the computer. A floating point number consists of three parts: a sign bit, an exponent, and a mantissa. A diagram of the parts of a floating point number can be seen in Figure 3.9. In the calculation of a floating point number, the mantissa is multiplied by a factor of two raised to the exponent field. The issue here is that a mutation in the sign bit or the exponent bit can cause huge fluctuations in the value of the parameter. This can disrupt a genetic algorithm's normal path to an optimal solution. The solution is to weight the choice of the bit to mutate so that it favors the mantissa more heavily. A good solution suggested at

http://www.coyotegulch.com/products/libevocosm/index.html was to keep the likelihood of selecting the exponent portion of the number under 15%.

The implementation of the Allele class allows this to be done by overriding the mutateR(int randInt) method. This method is what normally takes a random integer generated higher up in the program flow and uniformly maps it to one of the positions in the allele when mutation is being done. The DoubleAllele class does not do this mapping uniformly, but rather weights it to favor the mantissa. The beauty of this solution is that this change is transparent to the calling method, and it does not disrupt the chance that a position in the allele will be selected as a mutation point, as that is still based on the relative lengths in bits of each of the alleles in the chromosome.

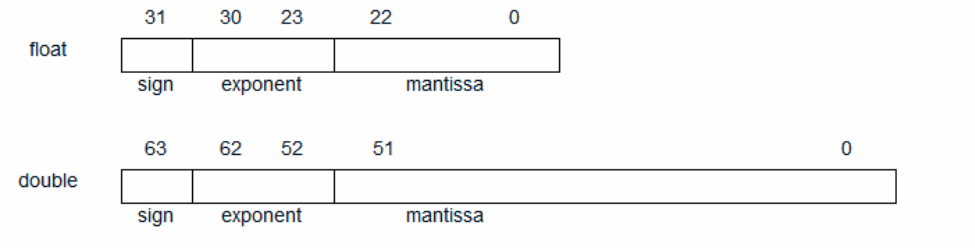

Figure 3.9: Floating Point and Double Numbers as they are Represented in Binary (Borrowed from http://www.coyotegulch.com/products/libevocosm/index.html)

Though the DoubleAllele class is not currently being used in the Word Segmentation genetic algorithm, its implementation is an important example for developers who might want to incorporate strings or other objects into their genetic algorithm.

3.8.1.2 Chromosome Class

The chromosome class' primary purpose is to collect a group of alleles together and allow for each allele to be given a human-readable string identifier. It hides the details of the allele implementation so that from the outside the Chromosome appears to function as a simple string of bits that are used in crossover and mutation as they would in more traditional implementations of a genetic algorithm. The Chromosome class is also responsible for ensuring that crossover and mutation can only occur between two chromosomes with the same alleles. An attempt to do these operations between two different types of chromosomes would result in an exception being thrown. The chromosome class does not need to be modified when converting the genetic algorithm for use in other applications.

3.8.1.3 GeneticOrganism Class

The GeneticOrganism class is the primary class that needs to change when the genetic algorithm module is applied to a new problem. It has three methods which must be overridden in any implementation of the genetic algorithm. One is relatively minor, (getMutationRate), and is only abstract to remind developers to set the mutation rate themselves. The other two, randomizeOrg and getFitness, are important parts of the algorithm.

In designing an implementation of the GeneticOrganism class, a developer needs to design the chromosome of their GeneticOrganism. The chromosome will be handled entirely by the GeneticOrganism—it will extract the values from that chromosome when calculating fitness, and generate values for its chromosome when it is randomized. These two manipulations of the chromosome should be handled in the getFitness and randomizeOrg methods, respectively.

For word segmentation, we have added a chain of two classes that inherit from the GeneticOrganism class. This is to make it easier for future developers to make implementations of the genetic algorithm for other word segmentation algorithms. Other word segmentation approaches should be able to be optimized based on the same fitness algorithm we have used to optimize the smearing algorithm, even though their chromosomes will be very different. For this reason, we have created a GeneticBoxOrganism abstract class. The GeneticBoxOrganism abstract class provides our implementation of the fitness algorithm (described in Section 3.8.2), but uses a new abstract method to help it do its calculation. A developer wishing to extend the GeneticBoxOrganism class should implement the randomizeOrg method as usual, but instead of implementing the getFitness method, he or she should instead implement the getAutobox method instead. This method should return the list of boxes obtained by the boxing algorithm. It will be used during the calculations in the getFitness method. For an example of an implementation of the GeneticBoxOrganism class, examine the SmearBoxOrganism class, which is used in the genetic algorithm for the smearing word segmentation algorithm.

3.8.1.4 The GeneticAlgorithm class

The GeneticAlgorithm class is the class that manages and runs the overall genetic algorithm process. It implements runnable, so it can be easily run in a separate thread. The genetic algorithm can be started by starting the thread, then stopped either automatically by setting the number of generations to terminate at, or stopped on the fly by using the call setContinue(false). This call will terminate execution at the end of the current algorithm, and automatically save all of the information about the generation to the disk. For an example of implementing a GeneticAlgorithm class, see the GeneticSmearBoxing class.

There are three small methods that must be implemented in order to make the GeneticAlgorithm class functional. Two, prepareForSerialization and deserialize, have to do with saving generation information when the algorithm is stopped. Sometimes, although the GeneticOrganism, chromosome, and Allele classes are all serializable, they contain data which is not serializable and must be removed or converted before saving. If that is the case, these methods allow the user to add or remove that kind of data from each GeneticOrganism before saving or after loading data from a file.

The third abstract method, runAdditionalRandomizeTasks, is used whenever data that the GeneticAlgorithm has must be inserted into every new GeneticOrganism. In this case, that data can be added by the runAdditionalRandomizeTasks method, which is run on every GeneticOrganism immediately after it is randomized.

3.8.2 The Word Segmentation Fitness Algorithm

The genetic algorithm implementation that we built and is currently in the codebase for the project is a tool to optimize the system's method of word segmentation. While the above sections give a good overview of the changes to the Genetic Algorithm classes that were required to create such a system, there is one major aspect of the design of the word segmentation genetic algorithm that remains to be described: the fitness algorithm.

The inspiration for the fitness algorithm came from a paper by Liang et al. (1997) that describes a basic evaluation algorithm for word segmentation. The method relies on a human creating a "ground truth" boxing beforehand which will be compared to the system's result. The algorithm then compares the automatic boxing to the "ground truth via the following criteria. Suppose the ground-truth boxing consists of a set boxes G = { G_1,G2,…,GM}, and the automatic boxing consists of a set of boxes D = {D1,D2,…DM}.

Then for some ground-truth box G_i and some automatic box D_j, a comparison between Gi and Dj can be made by analyzing the two equations shown in Figure 3.10.

$$\sigma_{ij} = \frac{\text{Area}(G_i \cap D_j)}{\text{Area}(G_i)} \text{ and } \tau_{ij} = \frac{\text{Area}(G_i \cap D_j)}{\text{Area}(D_j)}$$

Figure 3.10: Two Formulas Used in the Calculation of Fitness

These two formulas calculate the fraction of boxes G_i and D_j that overlap each other compared to the total area of each of the boxes. This is an excellent measure for the fitness of two boxes, but the genetic algorithm requires a more broad result. An optimum score had to be extracted from the data by matching each ground-truth box G_i to a single automatic box D_i. This was identified as a situation appropriate for the application of the Hungarian algorithm for solving the assignment problem. This technique pairs each automatic box with the ground-truth box that is closest to it. A more detailed description of the Hungarian Algorithm can be found in Appendix A. The fitness algorithm then averages the fitness found for each pair of boxes to produce the final fitness score.

3.8.3 Modifying the Genetic Algorithm for Other Applications

The major design goal while creating the Genetic Algorithm framework was that it should be extremely flexible and easy to modify for other applications. In that sense, this section, which is designed to give guidelines to a new developer for what to modify, is also a results section. It is proof that the genetic algorithm requires very little modification to be applied to new techniques.

There are only two classes that typically require extensions when creating a new system that uses the Genetic Algorithm framework. The first is the GeneticAlgorithm class. This class must be have three of its methods implemented in any implementation of the genetic algorithm: runAdditionalRandomizeTasks(), prepareForSerialization, and deserialize. These three methods are described in Section 3.8.1.4.

The user must also set up the GeneticAlgorithm class. This requires setting up the generation size and ending generation, and the base chromosome for the entire system. Once these are completed (examples can be found in the code), the GeneticAlgorithm class will handle most of the details of the genetic algorithm.

The only other major extension required to convert the genetic algorithm to a new application is creating the fitness algorithm in the GeneticOrganism class. This should return a floating point value between 0 and 1, with 1 as the highest score possible and 0 as the lowest possible. Once this is completed, the genetic algorithm should be complete as well.

4.0 Recommendations and Conclusions

While the Emergent Transcriptions initiative is not fully implemented, this project implemented some important functionality to allow for several of the components to be individually refined. We encapsulated the Transcription Assistant into the larger project by putting it in an applet for easier access, and developing a system of storage, retrieval, and searching on the information created by the Transcription Assistant. We created a genetic algorithm which was able to refine the existing automatic boxing algorithm for better results, as well as making this genetic algorithm extendable for future portions of this system. Finally we redesigned the Manuscript Markup Language (MML) to allow for the successful storage and transmission of all of the data relevant to a transcription. This new MML made possible implementing the information passing required for the new database, as well as for future components such as the Contribution Accountant.

There are many recommendations for how to take this project and build upon it to enhance the functionality of the Emergent Transcription initiative. Among these recommendations are the ideas of implementing the Contribution Accountant, trying new automatic boxing algorithms, fine tuning the line detection algorithm, extending the web front-end to create a community based system, implementing a functional import tool for the Archive Assistant, making an MML to TEI adapter, and doing user studies on the system components to refine the user interface.

The Contribution Accountant is the last remaining part of the Emergent Transcription initiative which is yet to be implemented. There are many areas in which we designed around the idea that this component will be implemented soon. The idea of credibility is built into both the new MML as well as the database structure to allow for a program to edit credibility on transcriptions and elements of transcriptions in any number of ways. This component should be a server-side process that takes information from the database about transcriptions and assigns credibility scores to components of the

transcription, or to the transcription as a whole and store this information back into the database and the MML. It should also handle updates to transcriptions and reassign credibility as needed. This process could even communicate with a more thorough web based community system to allow for users to influence credibility.

Although the current automatic boxing algorithm has been refined, it is certainly not the only algorithm for word segmentation. Other algorithms should be tried with our system and similarly tailored to the problem of segmenting words on manuscripts. Every algorithm tried should also be run through the existing genetic algorithm on several types of documents. Leveraging this genetic algorithm's work will allow the algorithm to be rated for different types of manuscripts, and ideally used automatically when it is seen as the best algorithm for a specific type of manuscript.

The line detection algorithm needs to be worked on further before it can be trusted by default. Currently the user usually still needs to modify the lines made to correctly place all of the boxes where they should be.

The web front-end also needs to be designed into a website capable of hosting a community. To do this, user logon and verification must be implemented, so that work done through the website, as well as work done in the applet loaded in the website, is associated with a users account. This new system should also allow a user to view and attach credibility scores to other transcriptions. A method of displaying these transcriptions and allowing for such scores to be edited outside of the applet must be implemented.

The Archive Assistant needs to be fully implemented. As it stands a stand-alone version of the Transcription Assistant still exists, and we recommend that this be stripped and made into an import tool designed at allowing the uploading of manuscripts, and optionally importing of existing manuscripts. This tool should still have elements capable of boxing, so that when an archivist goes to match an existing transcription to the automatic boxing, they can manually match up words to boxes. The new manuscript and transcription should be uploaded to the database for storage.

It would be useful is an adapter could be made to export an MML transcription into the Text Encoding Initiative (TEI) format, to make it more compatible with other software. Since the TEI format wasn't the best way to internally store transcription information, its inclusion the Emergent Transcription initiative should be at this higher, converter level. If such an export tool were made available, it could be placed in the web-front end, in the

Transcription Assistant, or even in the Archive Assistant, as well as in any combination of the above.

Finally the Transcription Assistant and web-front end, as well as any other system implemented, should be subjected to user testing to determine the effectiveness of each system, and the quality of the user interface given for each system. This should be partially completed by allowing for feedback in the various parts of the system, such as the Transcription Assistant and the web-front end. Also a formal set of user studies could be conducted.

5.0 Bibliography

Calhoun, Shaun, Anthony Tiscia, Terrence Turner. "Manuscript Transcription Assistant Initiative", 2004. WPI Major Qualifying Projects Collection.

"Dublin Core Metadata Initiative (DCMI)". Retrieved 4/25/07 from http://dublincore.org

Finch, Nathan. "Venetian Transcription Software", 1999. WPI Major Qualifying Projcects Collection.

"ISAD(G): General International Standard Archival Description, Second Edition". Released by the International Council on Archives (ICA). Retrieved 4/25/07 from http://www.ica.org/biblio/isad_g_2e.pdf

Ho, Oliver, Chirag Patel, Ravi Patel, Ricardo Kligman. "Manuscript Transcription Assistant", 2003. WPI Major Qualifying Projects Collection.

Liang, J, R. Rogers, R.M. Haralick, and I.T. Phillips: "UW-ISL Document Image Analysis Toolbox: and Expreimental Environment". 1997 IEEE. Retrieved 4/25/07 from http://web.cs.wpi.edu/~trascrizione/layout_analysis_toolbox.pdf

"Library of Congress Learning Page". Library of Congress, 2002. Retrieved 4/25/07 from http://memory.loc.gov/learn/lessons/psources/types.html

Rath, Toni M., R. Manmatha, Victor Lavrenko. "A Search Engine for Historical Manuscripts". Published by the ACM, 2004. Retrieved 4/25/07 from http://ciir.cs.umass.edu/pubfiles/mm-341.pdf

Sezgin, Mehmet, and Sankur, Bülent. "Survey Over Image Thresholding Techniques and Quantitative Performance Evaluation". Journal of Electronic Imaging, January 2004. Volume 13, pp. 146-168

"TEI: Yesterday's Information Tomorrow". Text Encoding Initiative 2007. Retrieved 4/25/07 from http://www.tei-c.org/

Tibbo, Helen R. "Primarily History: Historians and the Search for Primary Source Materials". *International Conference on Digital Libraries*. 2002. pp. 1-10

"Understanding MARC Bibliographic: Machine-Readable Cataloging". Library of Congress, 2003. Retrieved 4/25/07 from http://www.loc.gov/marc/umb/

Whitely, Darrel. "A Genetic Algorithm Tutorial". Retrieved 4/25/07 from http://samizdat.mines.edu/ga_tutorial/ga_tutorial.ps

Appendix A Hungarian Algorithm Description

(Taken from http://www.public.iastate.edu/~ddoty/HungarianAlgorithm.html)

Munkres' Assignment Algorithm
Modified for Rectangular Matrices

The following 6-step algorithm is a modified form of the original Munkres' Assignment Algorithm (sometimes referred to as the Hungarian Algorithm). This algorithm describes to the manual manipulation of a two-dimensional matrix by starring and priming zeros and by covering and uncovering rows and columns. This is because, at the time of publication (1957), few people had access to a computer and the algorithm was exercised by hand.

Step 0: *Create an nxm matrix called the cost matrix in which each element represents the cost of assigning one of n workers to one of m jobs. Rotate the matrix so that there are at least as many rows as columns and let k=min(n,m).*

Step 1: *For each row of the matrix, find the smallest element and subtract it from every element in its row. Go to Step 2.*

Step 2: *Find a zero (Z) in the resulting matrix. If there is no starred zero in its row or column, star Z. Repeat for each element in the matrix. Go to Step 3.*

Step 3: *Cover each column containing a starred zero. If K columns are covered, the starred zeros describe a complete set of unique assignments. In this case, Go to DONE, otherwise, Go to Step 4.*

Step 4: *Find a noncovered zero and prime it. If there is no starred zero in the row containing this primed zero, Go to Step 5. Otherwise, cover this row and uncover the column containing the starred zero. Continue in this manner until there are no uncovered zeros left. Save the smallest uncovered value and Go to Step 6.*

Step 5: *Construct a series of alternating primed and starred zeros as follows. Let Z_0 represent the uncovered primed zero found in Step 4. Let Z_1 denote the starred zero in the column of Z_0 (if any). Let Z_2 denote the primed zero in the row of Z_1 (there will always be one). Continue until the series terminates at a primed zero that has no starred zero in its column. Unstar each starred zero of the series, star each primed zero of the series, erase all primes and uncover every line in the matrix. Return to Step 3.*

Step 6: *Add the value found in Step 4 to every element of each covered row, and subtract it from every element of each uncovered column. Return to Step 4 without altering any stars, primes, or covered lines.*

DONE: *Assignment pairs are indicated by the positions of the starred zeros in the cost matrix. If C(i,j) is a starred zero, then the element associated with row i is assigned to the element associated with column j.*

Some of these descriptions require careful interpretation. In Step 4, for example, the possible situations are, that there is a noncovered zero which get primed and if there is no starred

zero in its row the program goes onto Step 5. The other possible way out of Step 4 is that there are no noncovered zeros at all, in which case the program goes to Step 6.