



HONOURS PROJECT REPORT

Zamani Data Archive: Metadata Management Tool and Search Interface

Author:

Michael Ferguson
(FRGMIC005)

Supervisor:

Prof. Hussein Suleman

	Category	Min	Max	Chosen
1	Requirements Analysis and Design	0	20	15
2	Theoretical Analysis	0	25	0
3	Experiment Design and Execution	0	20	10
4	System Development and Implementation	0	15	15
5	Results Findings and Conclusion	10	15	10
6	Aim Formulation and Background Work	10	15	10
7	Quality of Report Writing and Presentation	10		10
8	Adherence to Project Proposal and Quality of Deliverables	10		10
9	Overall General Project Evaluation	0	10	0
	Total		80	80

DEPARTMENT OF COMPUTER SCIENCE

UNIVERSITY OF CAPE TOWN

November 29, 2014

Abstract

The Zamani Data Archive aims to manage the Zamani Project's large collection of spatial data. The system comprises of an Archive, Public Portal, Metadata Management Tool and Search Interface. This development project outlines the development of the Metadata Management Tool and Search Interface components of the Zamani Data Archive that was designed to fulfill the needs of the Zamani Project team. The two components were developed in four iterations that were designed from the requirements of the team. The Metadata Management tool was built to offer a wide variety of user tasks, with the aim of producing descriptive metadata for the Zamani Project's cultural heritage data. Notably the tool offers persistence of metadata sets via its save and load functionality and the batch modification of metadata. The Search Interface allows for navigation through the data with search and discovery metaphors such as: text-based search, collection-based search and faceted search. Additionally, it provides a representation of the data. The system was evaluated with the use of a System Usability Scale (SUS). The mean SUS score for the Metadata Management Tool and Search Interface components was used to obtain a subjective label for each component based on an adjective rating scale. The Metadata Management Tool and Search Interface components usability was considered to be 'Good' and 'Excellent' respectively. Additionally, a User Acceptance Test was conducted with the Zamani Project team. The system passed all of the test cases and the team responded positively to the system, confirmed by their interest in the system being implemented in a production environment in the future.

Acknowledgements

Firstly, I would like to thank my project supervisor, Hussein Suleman, for his guidance, support and assistance during the duration of the project. His guidance, advice, encouragement and feedback were vital to the success of the project.

I would like to thank all of the members of the Zamani Project, particularly Professor Heinz Rüter, Roshan Bhurtha, Ralph Schroeder and Stephen Wessels for their time, effort and help that made this project possible. Additionally, thank you Roshan Bhurtha and Ralph Schroeder for committing much of your time to explaining the various processes involved in the Zamani Project. A special thank you to Roshan Bhurtha who was my main point of contact within the Zamani team. Thank you for responding quickly to project related queries.

I would also like to thank my project partner, Jay Benson, for his ongoing encouragement, advice, hard work and continual coffee runs. It was a pleasure to work with you and I look forward to working with you again in the future. Good luck with your future endeavors.

To all of the University of Cape Town students who participated in the usability study, thank you for your time and effort, even whilst on university study leave. Without you, our system evaluation would have been a disaster.

Finally, I would like to thank my family for their continued support, proofreading of documents and ability to handle my all-nighters.

Table of Contents

Acknowledgements	i
Table of Figures.....	v
1. Introduction.....	1
1.1. Zamani Project	1
1.2. Problem Statement	1
1.3. Outline of Solution.....	2
1.3.1. Metadata Management Tool	2
1.3.2. Search Interface.....	2
1.3.3. Archive.....	2
1.3.4. Public Portal.....	3
1.4. System Evaluation	3
1.5. Legal and Ethical Issues.....	3
1.6. Report Overview	3
2. Background	4
2.1. Introduction.....	4
2.2. Digital Repository Concepts	4
2.2.1. Metadata Creation and Management	4
2.2.2. Data Types	4
2.3. Repository Systems.....	5
2.3.1. Fedora	5
2.3.2. Aluka Archive.....	5
3. Design and Implementation	6
3.1. Introduction.....	6
3.2. Design Considerations	6
3.2.1. Team workstation configuration	6
3.2.2. Zamani Core.....	7
3.3. Tools and Technologies	7
3.3.1. Java	7
3.3.2. PHP	7
3.3.3. JApplet	7
3.3.4. Apache Tika	8
3.3.5. Nginx.....	8
3.3.6. Fedora Generic Search Service	8
3.3.7. Research Index	8
3.3.8. Research Index Search	8
3.3.9. Solr.....	8
3.3.10. JavaScript.....	8
3.3.11. AJAX	9
3.3.12. JQuery	9
3.3.13. Google Maps API.....	9
3.4. Fedora XML (FOXML)	9
3.5. Iteration 1: Initial Feasibility Design	11
3.5.1. Design and Implementation	12
3.5.1.1. Core Metadata Management Tool Requirements.....	12
3.5.1.2. Core Metadata Schema Design	12
3.5.1.3. Overview.....	13
3.5.1.4. Detailed Overview	14
3.5.1.5. Features	17

3.5.2.	Evaluation	23
3.6.	Iteration 2: Metadata Management Tool Design	24
3.6.1.	Design and Implementation	24
3.6.1.1.	Revised Core Metadata Management Tool Requirements.....	25
3.6.1.2.	Detailed Overview	25
3.6.1.3.	Usability Issues	34
3.6.1.4.	Additional Functionality	36
3.6.2.	Evaluation	41
3.7.	Iteration 3: Search Interface Design.....	41
3.7.1.	Design and Implementation	41
3.7.1.1.	Search Interface API	42
3.7.1.2.	Home Page Design.....	44
3.7.1.3.	Interfacing with the Search Interface API.....	46
3.7.1.4.	Important Functions	46
3.7.2.	Evaluation	50
3.8.	Iteration 4: Final Design	50
3.8.1.	Design and Implementation	50
3.8.1.1.	Metadata Management Tool Final Refinements	50
3.8.1.2.	Search Interface Final Refinements	51
4.	Evaluation	53
4.1.	System Testing.....	53
4.1.1.	Black-box Testing	53
4.1.2.	White-box Testing.....	53
4.1.3.	Peer Review	53
4.2.	Performance Testing	53
4.2.1.	Algorithm Comparison	54
4.2.2.	Methodology	54
4.2.3.	Results.....	55
4.2.4.	Discussion of results	56
4.3.	Usability Testing.....	56
4.3.1.	User Selection	57
4.3.2.	Procedure	57
4.3.3.	Questionnaire	57
4.3.4.	Results.....	57
4.3.5.	Reliability Analysis.....	59
4.3.6.	Discussion of results	60
4.4.	User Acceptance Testing	60
4.4.1.	Procedure	60
4.4.2.	Requirement-based Test case criteria.....	61
4.4.3.	Results.....	61
4.4.4.	Discussion of results	62
5.	Conclusion and Future Work	63
5.1.	Conclusion	63
5.2.	Future Work.....	64
5.2.1.	Metadata Management Tool	64
5.2.2.	Search Interface.....	64
6.	References	65
Appendix A	67
A1:	Embedding a Java applet: Avoid user file system security constraints.....	67
A2:	Basic FOXML file – demonstrates Dublin Core data stream	69
Appendix B	70

B1: Performance Test Algorithms	70
B2: Science Faculty Ethics Clearance	73
B3: Department of Student Affairs Ethics Clearance	74
B4: Consent Form	75
B5: System Usability Test	76
B6: User Acceptance Test	80
Appendix C	82
C1: Home Page View	82
C2: Show Collection View	83
C3: Pagination	83
C4: Show Parent Collection View	84
C5: Faceted Search	84
C6: Text-based Search	85
C7: Text-based Search: Auto-complete	85
C8: Show Metadata View	86
C9: Search by Type	86
C10: Show Metadata View – View Larger Image	87
C11: Show Metadata View – Popup Image View	87
Appendix D	88
D1: Overview Tab - MetadataView	88
D2: Batch Modify Popup	89

Table of Figures

Figure 1.1: Basic System Overview.....	2
Figure 3.1: Design Process	6
Figure 3.2: Zamani Network Configuration	7
Figure 3.3: Fedora Repository RELS-EXT Hierarchy.....	9
Figure 3.4: Root Zamani Project Collection RELS-EXT relationship	10
Figure 3.5: Country (Sudan) Collection RELS-EXT relationship	10
Figure 3.6: Example SITE datastream	10
Figure 3.7: Example FILE datastream contents.....	11
Figure 3.8: Example POSITION datastream contents	11
Figure 3.9: Example CALIBRATION datastream contents	11
Figure 3.10: Use case diagram of core Metadata Management tool requirements	12
Figure 3.11: Metadata Management Tool core functionality Class Diagram	13
Figure 3.12: Tag Sort.....	19
Figure 3.13: Table Selection Mechanism	20
Figure 3.14: Metadata Item Image Preview – Tooltip Feedback.....	21
Figure 3.15: Console Feedback.....	21
Figure 3.16: Preset Automation Fields	22
Figure 3.17: Use case diagram of revised Metadata Management tool functionality.....	25
Figure 3.18: Metadata Management Tool revised functionality Class Diagram	26
Figure 3.19: Class diagram of BackgroundWorker Class and PopupWindow relationship	30
Figure 3.20: Class diagram of helper Classes used in Import features	33
Figure 3.21: Class diagram of helper Classes used to Transform Aluka metadata.....	34
Figure 3.22: Look and Feel.....	35
Figure 3.23: Disabled Tag and Contain Sort Input Fields	35
Figure 3.24: Enabled Tag and Contain Sort Input Fields.....	35
Figure 3.25: Menu Categories and items	36
Figure 3.26: Save Feedback.....	37
Figure 3.27: Validation Feedback.....	38
Figure 3.29: Transform Aluka Metadata Popup	39
Figure 3.30: A line in the Aluka Error File.....	39
Figure 3.31: Edit Panel making use of Revised Field Dictionary	40
Figure: 3.32: Export Metadata Popup.....	40
Figure 3.33: Use case diagram of core functionality of the Search Interface	42
Figure 3.34: Zamani Project website	45
Figure 3.35: Zamani Data Archive - Home Page.....	45
Figure 3.36: Content Area IDs.....	46
Figure 3.37: Interfacing with the Search Interface API.....	46
Figure 3.38: URL example for RISearch Tuple Query Method	47
Figure 3.39: Faceted query using the Solr Query method	48
Figure 3.40: Search by Type.....	51
Figure 4.1: Mean Performance Analysis of Algorithms - Non-Heuristic (Java Application & Applet)	55
Figure 4.2: Mean Performance Analysis of Algorithms - Heuristic (Java Application & Applet).....	55
Figure 4.3: Approximate execution times of Classical Recursion and GET Request for a site collection .	56
Figure 4.3: Mean SUS scores of the Metadata Management Tool and Search Interface	58
Figure 4.4: Metadata Management Tool: Usability ratings	58
Figure 4.5: Search Interface: Usability ratings	59
Figure 4.6: Reliability Analysis of the SUS scores of both components	59
Figure 4.7: User Acceptance Test results.....	61

Chapter 1

Introduction

1.1. Zamani Project

The Zamani Project [32] requires an archival system for its large collection of spatial data. The system is required to manage the data, allow for navigation through the data and present the data to end users.

The Zamani Project was initiated to create a permanent metrically accurate record of important cultural heritage sites in Africa and the Middle East as well as increase the international awareness of these sites [34]. The project was started at the Geomatics Department at the University of Cape Town and is led by Professor Heinz Rüther [29]. The purpose of creating a permanent metrically accurate record is fourfold in that the data can be used for education, research, restoration and conservation [34].

Thus far, the project has documented over 40 sites in 12 countries in Africa and the Middle East. Some of the documented countries include Ghana, Sudan, Mali, Tanzania, Kenya, Ethiopia and South Africa [34]. This amounts to close to one hundred three-dimensional (3D) models of individual structures. The dataset collected by the project is currently approximated at 44TB in size and consists of a variety of different digital objects such as: 3D Models, Geographic Information Systems, Panoramic Photographs, Plans and Videos [33].

1.2. Problem Statement

The Zamani Project requires an archive for their large collection of spatial data collected at various cultural heritage sites. There are two major aspects that need to be considered: archival storage management and end-user presentation.

A Metadata Management Tool that is specific to the needs of the Zamani Project will be required to create metadata for Zamani's data store. A management system is needed to organize the created metadata as a layer over the team's current drive-based data store. A Web-based Public Portal should then allow end-users to search, view, request and download data from the archive.

Additionally, the datasets to be stored are collections of digital objects that are associated with cultural heritage sites. The relationship among digital objects within a collection needs to be maintained in the archive with the use of structural metadata. Furthermore, the relationships between objects needs to be exposed to end-users during the discovery process.

The dataset's metadata is partial and varies in quality. Part of the solution will include automating the process of generating metadata for the many different digital object file formats that are used.

1.3. Outline of Solution

The project is divided into four major components. The first two components are a metadata management tool and search interface, developed by Michael Ferguson. The second two components are a public portal and archive, developed by Jay Benson. These four components aim to solve the Zamani Project’s need for an archival system. The basic system overview is illustrated in Figure 1.1.

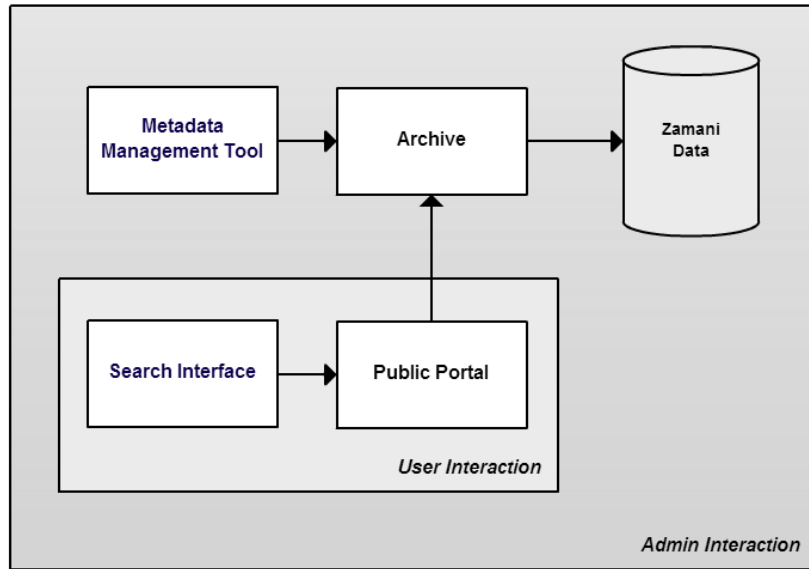


Figure 1.1: Basic System Overview

1.3.1. Metadata Management Tool

Meta-fy, a Metadata Management Tool was developed in order to assist with the difficult task of creating metadata for a large amount of a data. The tool was developed using the requirements of the Zamani Project team. It is able to handle a wide variety of user tasks and offers the ability to expand in the future via its API. The aim is to produce descriptive metadata for the cultural heritage data that can be exported in XML and then ingested into the archive.

1.3.2. Search Interface

A search interface was developed to provide navigation through the dataset as well as provide detailed information and representation of the data. Navigation through the data can be achieved by text-based search, site collection search via a Google Map overlay and faceted search. The Search Interface contains a REST API that can be used for search interface related queries as well as data representation requests.

1.3.3. Archive

The primary data storage component of this project is the data archive. The Archive is responsible for managing and preserving the data and its respective metadata. Similarly to an archivist, the archive is also responsible for a number of tasks that are present in the archive backend. These tasks include: download management, metadata management via the Metadata Management Tool and ingesting and purging of metadata. The backend is password protected and only accessible by an administrator. Additionally, the archive has an API that is used to request data from the archive. Data requests are used by the Public Portal in order to request data if permitted.

1.3.4. Public Portal

The Public Portal serves as the central point of interaction with the archive. It is responsible for dealing with requesting of data as well as distribution if requests are permitted. The Public Portal consists of a search interface and data request interface. The data request interface provides users with a means to request and download files. It requires that users are registered before requesting data. In order to ensure that Zamani's data policies are adhered to, a user permission system has been integrated into the public portal. This ensures that users can only access digital objects that they have permissions for.

1.4. System Evaluation

The system was evaluated on two fronts in order to determine the applicability of the proposed solution:

Usability Testing

The usability of a system is a critical component that needs to be considered when designing and evaluating a system. Nielson [25] presents that the benefit of a usable system is that the productivity of its users is greatly increased. A usability test was conducted to ensure that the system provides an appealing and usable interface.

User Acceptance Testing

In order to determine whether the proposed system met all of the project requirements, User Acceptance Testing took place. The purpose of this evaluation was to determine if the project was a success and could be adopted by the Zamani Project.

1.5. Legal and Ethical Issues

The tools and technologies used in the development of the Zamani Data Archive are all free or open source. This was done to ensure that the system could be extended in the future. Secondly, the Zamani Project provided the development team with data and information that could be used for testing purposes. Permission was obtained from the Zamani Project team to use this data. Lastly, University of Cape Town students were used in the Usability Testing that was done during the evaluation phase. Ethical clearance was obtained from the Science Faculty Research Ethics Committee (See Appendix B2). Additionally, access to students was granted by the Department of Student Affairs (See Appendix B3).

1.6. Report Overview

This report covers the design, implementation and evaluation of two components of the Zamani Data Archive, namely the metadata management tool and search interface. The report begins with a review of the Background literature in Chapter 2. This is followed by the Design and Implementation of the two components in Chapter 3. The evaluation methods that were used, results of the evaluation and discussion of the results are in Chapter 4. Lastly, conclusions and avenues for further work are outlined in Chapter 5.

Chapter 2

Background

2.1. Introduction

The background chapter consists of information that is considered relevant to the development of the proposed system. Firstly digital repository concepts are explained, including the creation and management of metadata and the different data types created within the Zamani Project. The Zamani Project's dataset consists of a number of different data formats that were of importance. This is followed by an overview of repository systems, including Fedora, the chosen Archive repository and Aluka's archive.

2.2. Digital Repository Concepts

2.2.1. Metadata Creation and Management

Metadata can be defined as structured information that describes and enables the easy retrieval, use and management of an information resource [27]. It is often referred to as 'data about data'. A metadata schema is used to structure information into descriptive records by label or tag. It is used to establish and define data elements and the rules that govern how a data item can describe a resource [27].

The creation of metadata can be supported by semi-automatic processes such as: document properties (e.g. creation date), spatial and temporal information contained within data captured by cameras and sensors. However, a large number of other metadata characteristics require a certain degree of human intervention. Additionally, metadata needs to be managed to ensure:

1. *Availability*: Metadata needs to be stored in an accessible location and outputted in a format that can be indexed in order to be found.
2. *Persistence*: Metadata needs to be kept over time.

To this end, metadata management tools are used to create and manage structured sets of metadata. Semi-automatic processes are made use of where possible but additional mechanisms are needed to enable the ease of modifying a metadata set. Such mechanisms include sorting the metadata set, modifying multiple metadata items simultaneously and image previews.

2.2.2. Data Types

There are a number of different data types that the Zamani Project team typically produce. A basic definition of the various data types, gathered from the team are provided below:

3D Models

Three-dimensional scaled visually metrically accurate representations of historically relevant structures in digital form. The level of detail of the 3D models varies and is differentiated by resolution. The relative scale is 'High', 'Medium' and 'Low' resolution.

Laser Scans

Laser scans are used to accurately determine the position of surface points on an object. A laser scan consists of a three-dimensional point cloud that represents the object.

Elevations and Sections

Plans consist of elevations and sections. Elevations and sections are similar to architect drawings but possess increased accuracy.

Ground Plans and Top Views

Line drawings showing the layout of buildings by means of plans near ground level.

Geographic Information Systems (GIS)

Consisting of spatially referenced information about the environment. GISs consist of site features in geographically referenced positions and can be displayed in map form.

Contextual Images and Videos

Images and videos captured at a site level to capture the state of the outlying environment of heritage sites.

Photogrammetric Images

Digital photographs that have been captured with the use of calibrated amateur cameras or metric cameras. These images are used to texture 3D models that were not laser scanned.

Panorama Tours

Images with a broad (360 degree) view of the environment or building interior. The images are combined to form a virtual tour.

2.3. Repository Systems

2.3.1. Fedora

Fedora [16] is an architecture for digital asset management that can be used to build digital archives. It is highly customizable due to its modular architecture and should not be seen as a complete application for managing, indexing and discovery. The Fedora repository provides a management layer over digital objects [13]. Digital objects are composed of datastreams that contain content and/or metadata about the object. Datastreams can be used to add relationships between objects as well as store additional metadata relating to the object.

2.3.2. Aluka Archive

Aluka [1] is a collaborative international programme that is aimed at creating an online digital repository about Africa (<http://www.aluka.org>). It archives non-spatial data such as books and scientific papers in digital form as well as presents its spatial data in the form of GIS, Spatial Information Systems (SIS), 3D models, elevation views, ground plans, sections and panoramas. It makes use of various technologies to acquire such data, namely: photogrammetry, laser scanning, remote sensing, image processing, conventional surveying, GIS and CAD. Additionally, all of the spatial data is associated with metadata about the survey details. Aluka had a total of nineteen sites that had been documented by the end of 2009. The Zamani Project team were responsible for documenting the sites, this was accomplished with the use of Geomatics technology that was used to scan the heritage sites.

Chapter 3

Design and Implementation

3.1. Introduction

During the development life cycle an iterative design process was followed. Four design iterations were used to implement the system. The iterations are as follows: i) Initial Feasibility Design; ii) Metadata Management Tool Design; iii) Search Interface Design; and iv) Final Implementation Design. Each iteration followed a defined design process to design, implement and evaluate the system. The design process is illustrated in Figure 3.1. The evaluation of previous iterations was used to provide the following iteration's design step with additional input.

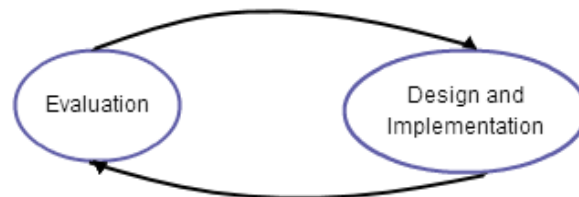


Figure 3.1: Design Process

Weekly meetings with the project supervisor were arranged in order to keep track of progress. Additionally, this approach ensured that any scope creep, risks or delays were minimized. The Zamani Project team were also involved in the design and development process, being the client it was vital to get continued feedback from the team. Regular meetings with the team were arranged to provide evaluation, feedback and gather further requirements on each iteration.

This chapter outlines the processes of the design and development of the Metadata Management Tool and Search Interface components of the Zamani Data Archive. This chapter will first describe the initial design consideration (See Section 3.2) and then the technologies that were used during development (See Section 3.3).

3.2. Design Considerations

The system that was developed was for the Zamani Project. Thus, a thorough understanding of the team's setup and requirements was needed. Additionally, the general requirements of a Metadata Management Tool and Search Interface were considered.

3.2.1. Team workstation configuration

Each member of the Zamani team has their own workstation in the Geomatics Department at the University of Cape Town. A workstation can be operationally defined as a special computer that is designed to handle performance-intensive tasks. The specifications of the workstations vary but are predominantly powerful Microsoft Windows 7 machines that are connected to a high speed Local Area Network (LAN). Additionally, a server is connected to the LAN via a one Gb/s (gigabit per second) network cable.

The server is used by the Zamani team as a repository for the documented sites contained in Zamani Core. The described Zamani network configuration is illustrated in Figure 3.2.

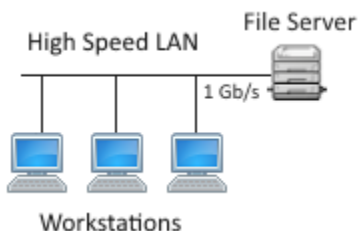


Figure 3.2: Zamani Network Configuration

3.2.2. Zamani Core

Zamani-Core is the main directory on the server. The directory contains a number of documented countries, inside of which are a number of site directories of sites that were documented at the associated country. The general structure for a site directory is that it contains three directories, namely: Models, Images and GIS. These directories contain the various different types of data that the team have captured at the site.

3.3. Tools and Technologies

3.3.1. Java

Java is a class-based object-oriented programming language.

An object-oriented approach was used when designing the Metadata Management Tool. Object orientated programming is a programming paradigm that maps classes to objects. Objects interact with one another to form fully designed and functional applications [23]. This approach was used as it provided a clear modular structure for the application and enabled easy maintenance and extension of code. The Metadata Management Tool was implemented using java and is run as a Java applet imbedded in an HTML document.

3.3.2. PHP

Hypertext Preprocessing (PHP) is a server side scripting language that is embedded into HTML documents and processed by the server when requested for by a Web browser. PHP was used to dynamically generate HTML based on the values given to the HTTP GET method. The Search Interface API consists of a number of PHP scripts.

3.3.3. JApplet

A Java applet is a Java application that is delivered to users in byte code. An applet is launched from a user's browser and executed by the JVM in a separate process to the Web browser. Applets are fast and can have similar performance to natively installed software. Additionally, Java applets are cross platform friendly and are supported by most Web browsers. One of the core advantages of using an applet over a native application is that there is minimal configuration required, if any. If an applet requires a later version of the Java Runtime Environment (JRE), the user will be prompted to download the required software. One of the requirements of the Metadata Management Tool was that it is Web-based, to this end, a Java Applet was chosen.

3.3.4. Apache Tika

Apache Tika toolkit [2] is used to detect and extract metadata as well as text content from a large number of different file types. Apache offers a Tika app JAR that can be used to extract such content. The Apache Tika library was chosen as it is a robust and extensible solution that can be applied to a large number of different file types. Additionally, it allows for possible future extension if additional metadata fields need to be extracted in the future.

3.3.5. Nginx

Nginx [24] is a Web server that can be used to run application servers and website servers. Nginx was used to run Fedora as well as the Web-based Zamani Data Archive. Testing was performed using WAMP, due to its ease of installation and ability to handle PHP. Testing for the Search Interface was carried out locally, by hosting it using WAMP. This enabled debugging of code before it was deployed to the server hosted in the University of Cape Town Computer Science building.

3.3.6. Fedora Generic Search Service

Fedora Generic Search (GSearch) [12] adds a variety of additional features to Fedora, such as full-text indexing of Fedora XML (FOXML), which includes data stream text content. Additionally two other major features of GSearch are: its ability to search the index and plugin selection of search engines such as Solr. This service is not an out-of-the-box feature offered by Fedora, it needed to be installed alongside it. GSearch was predominantly used for its ability to query object data streams.

3.3.7. Research Index

The Research Index [10] is a Fedora module for indexing the relationships among objects. Fedora defines a base relationship ontology in order to express relationships. This base relationship ontology is defined using the RDF Schema. Additionally, Fedora provides a data stream for RDF expression based ontology. The Research Index module is included out-of-the-box with Fedora.

3.3.8. Research Index Search

The Research Index Search (RISearch) [9] is a Fedora Web service that is used to expose the contents of the Resource Index of a repository. The service includes a Web browser interface but can also be accessed programmatically. RISearch was used for collection-based searching and backtracking to a collection from an object in the collection.

3.3.9. Solr

Solr [30] is a scalable enterprise search platform that is open source. Solr provides a number of features that the Search Interface requires. Such features include: full-text search, faceted search and auto-complete. Solr provides a HTTP API that can be used to leverage such features. All of the above mentioned features were implemented in the Search Interface.

3.3.10. JavaScript

JavaScript [17] is a programming language typically bundled with Web browsers. It enables code to be run on the client side.

3.3.11. AJAX

Asynchronous JavaScript and XML (AJAX) [18], is used to send and retrieve data from a server asynchronously. Thus, AJAX is used in the background and does not affect the display or behavior of the Web page.

3.3.12. JQuery

JQuery [20] is a JavaScript library that is small in size, fast and rich in features. It provides an API to simplify HTML document manipulation and AJAX. For the Search Interface, it was used to make calls to the Search Interface API via a GET request. Additionally, it was used for dynamic content generation. With the aid of AJAX, content can be dynamically generated without a page refresh.

3.3.13. Google Maps API

The Google Maps API [19] is a JavaScript library that can be used to create interactive maps. It was used in the Search Interface for collection based searching.

3.4. Fedora XML (FOXML)

Fedora XML (FOXML) [11] is an XML format that is used to express the Fedora digital object model. Fedora currently offers ingest and export support for FOXML and Metadata Encoding and Transmission Standard (METS). FOXML version 1.1 was chosen as the XML format for digital object encoding as it provides a number of benefits over METS, namely: simplicity, optimization, performance and flexibility [11]. The Fedora Digital Object Model follows a compound digital object design, aggregating one or more data streams into one digital object. The core components of a Fedora digital object are: PID, object properties and datastream/s. The PID is the persistent, unique identifier of the object. Object properties are a set of properties that are required to manage digital objects within a Fedora repository. A data stream is an element of a digital object representing a content item [13].

Fedora's datastreams have a Control Group property that is used by the datastream to encapsulate its contents. The Control Group property chosen was Internal XML Content, as it provides the desired effect of storing the XML in-line within a digital objects XML file. Fedora has a DC (Dublin Core datastream) that is used to store metadata about the digital object. Fedora offers a RELS-EXT datastream that is used to describe relationships amongst digital objects. Additionally, custom datastreams can be made to record further details about the digital object [13].

FOXML was used to create a relational structure within the Fedora repository with the use of the RELS-EXT datastream (See Figure 3.3).

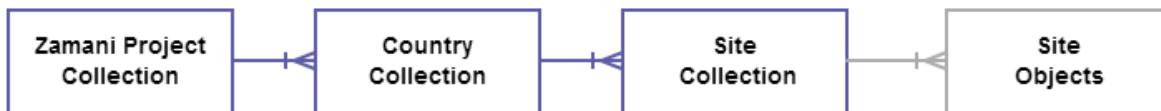


Figure 3.3: Fedora Repository RELS-EXT Hierarchy

This was achieved with the Resource Description Framework (RDF) which encodes object-to-object relationships in XML [14]. A separate digital object (FOXML file) is used to represent each Collection and Site object. The relationship must be specified using the standard RDF authoring style, this is illustrated in Figure 3.4 and Figure 3.5.


```

<rdf:Description rdf:about="info:fedora/zamaniProjectCollection:1">
  <rel:isCollection>true</rel:isCollection>
</rdf:Description>

```

Figure 3.4: Root Zamani Project Collection RELS-EXT relationship

```

<rdf:Description rdf:about="info:fedora/SudanCollection:1">
  <rel:isMemberOf rdf:resource="info:fedora/zamaniProjectCollection:1"/>
  <rel:isCollection>true</rel:isCollection>
</rdf:Description>

```

Figure 3.5: Country (Sudan) Collection RELS-EXT relationship

All of the FOXML files produced for the Fedora repository consist of two standard datastreams, namely: DC and RELS-EXT. The DC datastream is used to store the majority of metadata relating to digital objects. The RELS-EXT datastream is used to create relationships amongst the digital objects in the repository. There are a number of additional custom datastreams that add additional information to the digital objects.

Site Collections have an additional SITE datastream used to store site related information (See Figure 3.6). This datastream is vital in ensuring the Search Interface's map-based Collection search can be implemented.

```

<foxml:datastream CONTROL_GROUP="X" ID="SITE" STATE="A">
  <foxml:datastreamVersion ID="SITE1.0" LABEL="Site info about this object" MIMETYPE="text/xml">
    <foxml:xmlContent>
      <site>
        <country>Sudan</country>
        <name>Musawwarat es-Sufra</name>
        <position>
          <latitude>16.488064</latitude>
          <longitude>33.415227</longitude>
        </position>
      </site>
    </foxml:xmlContent>
  </foxml:datastreamVersion>
</foxml:datastream>

```

Figure 3.6: Example SITE datastream

Site Objects consist of an additional two datastreams namely: FILE and POSITION. The FILE datastream stores information relating to the file that the digital object represents (See Figure 3.7). This ensures that the DC datastream that contains the core information about the file is accompanied by the path to the file. This was significantly important as the FILE data stream is used in the Search Interface and enables files to be downloaded via the Data Request Interface from the Research Data Archive. Thus, the FILE data stream is fundamental in providing a mechanism to accessing the files on the server's mounted drive. The POSITION datastream is used to store additional information relating to where the file was captured (See Figure 3.8).

```

<file>
  <size>4.4 MB</size>
  <server_path>
  /mnt/shares/zamani/Zamani-
  Core/Sudan/Musawwarat/Images/submitted/Photograph Submit/DSCF3515.jpg
  </server_path>
</file>

```

Figure 3.7: Example FILE datastream contents

```

<position>
  <latitude>16d24'46"N</latitude>
  <longitude>33d19'25"E</longitude>
</position>

```

Figure 3.8: Example POSITION datastream contents

Site Objects can conditionally contain the CALIBRATION datastream (See Figure 3.9) if they represent an image type. This datastream is used to add additional information to images, namely the camera calibration settings that were used during capture.

```

<calibration>
  <date>06/07/2009</date>
  <camera>NIKON D200</camera>
  <decenteringP1>-7.18639e-005</decenteringP1>
  <decenterinP2>-1.36250e-006</decenterinP2>
  <distortionAffineB1>-2.56338e-004</distortionAffineB1>
  <distortionAffineB2>1.65759e-004</distortionAffineB2>
  <distortionRadialK1>2.10732e-004</distortionRadialK1>
  <distortionRadialK2>-2.54109e-007</distortionRadialK2>
  <distortionRadialK3>1.42781e-010</distortionRadialK3>
  <lengthFocal>28.1331</lengthFocal>
  <pixelSizeX>0.007</pixelSizeX>
  <pixelSizeY>0.007</pixelSizeY>
  <principalPointX>0.0134</principalPointX>
  <principalPointY>-0.5442</principalPointY>
  <sizeX>3872</sizeX>
  <sizeY>2592</sizeY>
</calibration>

```

Figure 3.9: Example CALIBRATION datastream contents

3.5. Iteration 1: Initial Feasibility Design

The focus of this iteration was to perform an initial feasibility study to ensure that all of the requirements of the Zamani Project could be met. Tasks that could cause technical challenges were identified. During this iteration a significant amount of time was spent comparing various Metadata Management Tools and analyzing the Zamani Project's in-house metadata management tool called Metty-D. This phase produced an evolutionary prototype that supported the core metadata management tasks, avoided Java applet security constraints and facilitated metadata management using the table selection mechanism.

3.5.1. Design and Implementation

The focus of the initial design process was to enable the core tasks of the Metadata Management Tool as well as any features that were identified as having an associated technical challenge. The three main challenges identified were: metadata extraction from digital objects, JApplet security constraints and the table selection mechanism.

The first iteration of implementation was used to build an evolutionary prototype to test the feasibility of the core Metadata Management Tool components as well as deal with any functionality that had associated technical challenges.

3.5.1.1. Core Metadata Management Tool Requirements

The core requirements of the Metadata Management Tool were identified and are illustrated in the use case diagram in Figure 3.10.



Figure 3.10: Use case diagram of core Metadata Management tool requirements

The metadata management tool would be designed to be functional, with less emphasis given to aesthetics. Figure 3.10 illustrates the core set of requirements for the first iteration, the functionality and requirements would be further refined and added to in the second design iteration.

3.5.1.2. Core Metadata Schema Design

The core metadata schema contains 17 field names that are common to all of the different types of data and are the following: Description, Identifier, Title, Coverage Spatial Country, Coverage Spatial Name, Coverage Spatial Structure, Coverage Spatial Position, Coverage Spatial Part, Coverage Spatial Locality, Creator, Date, Format, Source, Type, Position Latitude, Position Longitude and Position Ellipse.

3.5.1.3. Overview

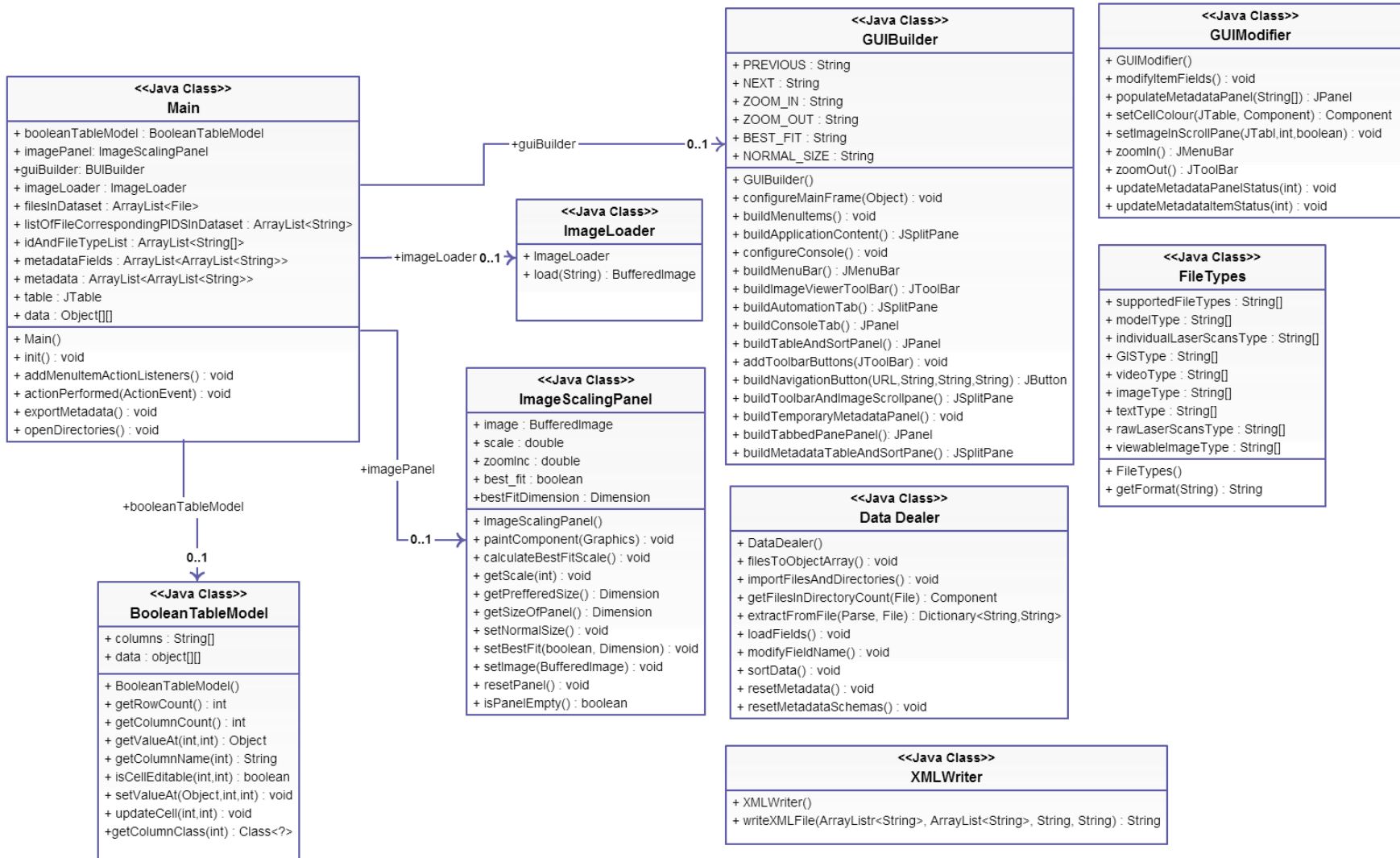


Figure 3.11: Metadata Management Tool core functionality Class Diagram

Model View Controller Pattern

The Metadata Management Tool makes use of a Model View Controller (MVC) Pattern [15]. An MVC pattern was chosen as the tool will make use of a GUI which is compliant to the MVC Pattern. This pattern separates the system into three parts: Model, View and Controller.

- **Model** – Contains data of the application and logic, methods and functions used to manipulate it.
- **View** – A means of presenting the data.
- **Controller** – Maps user interactions from the view in order to update the Model.

The Model of the program is the zamani.metafy.core.data package, which contains the DataDealer Class and other data related helper Classes. The Main class is the main application view and serves as the View of the program. Additionally, the Main class's table selection mechanism, ActionListeners and the GUIModifier Class serve as the Controllers of the application. When a user action is executed, the controller updates the view and model with the necessary changes. The separation of the controller and GUI leads to system changes generally only affecting one section of the MVC.

Array vs. ArrayList

Throughout the development of the Metadata Management Tool, different array type data structures were used. Arrays were used where no insertions or deletions were expected. ArrayLists were used where additions and/or deletions were expected and frequent random access.

3.5.1.4. Detailed Overview

Class Definitions and Methods

1. *Main* – The Main Class is a GUI based Java Applet. It deals with applet initialization, continuation, user interaction and stores frequently used variables statically.

Main Static Variables:

- basicMetadataSchema
 - ArrayList containing a list of the Core Metadata Schema field names that applies to all of the different types of files.

All of the below static variables relate to the metadata of the current dataset, all of which are ArrayLists with corresponding indices. Corresponding indices are used due to the orientation of the data structures. All of the data structures relate to digital objects in the dataset, by using corresponding indices, all of the required information can be acquired for a digital object by accessing the various variables using the same index.

- filesInDataset
 - ArrayList of Files in the current dataset.
- listOfFilePathsInDataset
 - ArrayList of file paths in the current dataset.
- listOfFileCorrespondingPIDSInDataset
 - ArrayList of PIDS in the current dataset.
- idAndFileTypeList
 - ArrayList of String Arrays containing: PID, file extension, index (Index of metadata item at creation), status (Percentage of metadata field values that are complete) and camera model (only relevant to image types).
- metadataFields
 - ArrayList containing an ArrayList of field names for each metadata item in the current dataset.

- metadata
 - ArrayList containing an ArrayList of corresponding field values for each metadata item in the current dataset.

Methods:

- `init()`
 - Applet initialization, called when applet loaded into browser.
 - Uses `GUIBuilder` Class to create the initial GUI.
- `actionPerformed()`
 - Deals with all action events for button presses, calls the associated Action Event Method.

Action Event Methods:

- `openDirectories()`
 - Uses the `DataDealer` Class to import the files in the directory.
 - `exportMetadata()`
 - Uses the `XMLWriter` Class to write all metadata to `FOXML`.
2. *GUIBuilder* – The `GUIBuilder` Class contains GUI functions relating to the construction of the GUI.
 3. *GUIModifier* – The `GUIModifier` Class contains GUI functions relating to the dynamic modification of the GUI.

Methods:

- `getSelectedTableItemIndexes()`
 - Get a list of the indices of the selected table items.
 - `populateMetadataPanel(String[] labels)`
 - Populate the metadata Panel with the set of field labels and their associated metadata.
 - `setImageInScrollPane()`
 - Sets the current preview image to the newly selected table row.
 - `zoomIn()`
 - Zoom into the image in the image scaling panel by increasing the image scale.
 - `zoomOut()`
 - Zoom out of the image in the image scaling panel by reducing the image scale.
4. *DataDealer* – The `DataDealer` Class deals with the majority of data calculations. It contains functions relating to file input/output, metadata calculations, reset methods and the core sort function.

Methods:

- `importDirectories()`
 - Imports the files in a directory and stores created metadata in the necessary data structures (Main Class's static variables). This method recursively iterates through a directory and its sub-directories.
- `filesToObjectArray()`
 - Performed after files have been imported. Calculates an `Object[][]` containing the metadata and sets the table models data to it.
- `extractFromFile(Parser parser, File file)`

- Extracts the Camera Model from a file using the Apache Tika Parser and returns a dictionary containing the extracted metadata. This method can be extended to extract a variety of metadata from a file.
 - `sortData()`
 - This method is used to sort the metadata after a filter has been selected. Each filter has an individual sort function that sorts the `idAndFileTypeList`. Each element of the `idAndFileTypeList` is an Array where the third index value is equal to its index on creation. When the `idAndFileTypeList`'s elements are sorted in ascending order by their third index value, they are sorted by Identifier. This function is used to sort all of the other metadata data structures after the `idAndFileTypeList` has been sorted by its respective filter. This is done because the Main Class's static variables, specifically arrays and lists relating to the metadata, use corresponding indices, thus all of them need to be normalized to ensure correct functioning after sorting.
 - `resetIndexesOfSortedData()`
 - This method is used to reset the indices of the `idAndFileTypeList` after the `sortData` method has been called. Resetting the indices involves ordering the `idAndFileTypeList`'s list elements in ascending order by the value contained within their third index. The third index contains the elements index on creation. By resetting the indices of the `idAndFileTypeList`, additional sorts can be accomplished.
 - `resetMetadata()`
 - Reset all metadata data structures to null. This method is used if there is data in the metadata data structures and the Open Files or Open Directories features are used.
 - `resetMetadataSchema()`
 - Resets the basic metadata schema by reading in its contents from the `basicMetadataSchema.txt` setting file. The `basicMetadataSchema` contains lines of the Basic Metadata Schema fields (See section 3.4.1.2).
5. *FileTypes* – The `FileTypes` Class deals with the accepted file types and their relevant Mime types. It consists of a number of string arrays of the various type categories. A list of the various type categories are as follows: GIS, Image, Individual Laser Scan, Model, Raw Laser Scan, Supported Files, Text Type, Video and Viewable Image.

Methods:

- `getFormat(String fileExtension)`
 - Get the mime type of a file from its file extension. Only supports mime types of the supported file types that are contained in the `FileTypes` Class.
6. *ImageScalingPanel* – The `ImageScalingPanel` Class extends `JPanel` and serves as the renderer of a metadata items image preview as well as deals with manipulation of the preview.

Methods:

- `paintComponent(Graphics g)`
 - Custom `JPanel` paint component used to render the image preview on the `JPanel`.
- `getSize()`
 - Get the size of the `ImageScalingPanel`.
- `resetPanel()`
 - Reset the `ImageScalingPanel` to render no preview image.
- `calculateBestFitScale()`
 - Calculate the scale that best fits the `ImageScalingPanel`.
- `setBestFit()`
 - Set the Image Scaling Panel to render the image preview with the newly calculated scale.
- `setImage(BufferedImage image)`
 - Set the image that the `ImageScalingPanel` should render.

- setNormalSize()
 - Set the normal scaling panel to its normal size.
 - setScale()
 - Set the scale of the ImageScalingPanel.
7. *BooleanTableModel* – The *BooleanTableModel* Class is a custom table model that renders Boolean data as check boxes and only enables the first column in the table to be edited.

Methods:

- isCellEditable(int row, int col)
 - Check to see if a cell is editable, only the first column is regarded as editable. Returns true if a cell is editable and false if not.
 - updateCell(int row, int col)
 - Update the specified row by firing a table cell updated event.
 - setValueAt(Object value, int row, int col)
 - Modify the model's data array and update the cell
 - getValueAt(int row, int col)
 - Get the model's data value at the specified row and column
 - getColumnClass(int columnIndex)
 - Method used by the model to define default renderer/editor for each cell. Used to render Boolean data as a check box, returns the column Class at the specified column index.
8. *XMLWriter* – The *XMLWriter* Class deals with the writing of metadata to an XML file.

3.5.1.5. Features

A number of features were developed from the requirements of the first iteration (See Figure 3.3), below is a list of the features that were developed during the first design iteration.

Importing of Metadata

The Import Metadata feature is present in the Metadata Management Tools 'File' menu. Importing metadata consists of prompting the user for a dataset directory that is then used to facilitate the creation of the datasets associated metadata.

The *DataDealer* Class's *importDirectories* method is used to prompt the user for a dataset directory and then import all of the supported files in the directory. Once the dataset directory has been acquired from the user, all of the files in the dataset that are of a supported file type are processed. In order to process all of the supported file types in the given dataset directory, the dataset directory is recursively traversed and each file in the directory is only processed if its file type is contained within the *FileTypes* Class's *supportedFileTypes* list.

Each supported file is then processed as follows:

- i. A new basic metadata schema is created for the file using the *Main* Class's *basicMetadataSchema* list. The *basicMetadataSchema* list will contain a number of creator fields if multiple creators have been added to the *Creator* field in the automations field dictionary.
- ii. A temporary metadata array that is the size of the *basicMetadataSchema* list is created for the file.
- iii. The index of the *Identifier* field in the *basicMetadataSchema* list is then used to set the corresponding identifier index value in the temporary metadata array. The identifier value consists of the identifier prefix present in the *Automation Fields* combined with a

- semicolon and unique identifier. The unique identifier consists of seven digits, it is created from a zero padded version of the file index (e.g. index 1 has a unique identifier of 0000001). This format was determined by the PID format that Fedora required.
- iv. The index of the first Creator field in the basicMetadataSchema is then used to set the creator field values (if more than one) in the temporary metadata array.
 - v. The index of the Format and Date fields in the basicMetadataSchema are then used to set the extracted values respectively.
 - vi. The Source, Coverage Spatial Country and Coverage Spatial Name field values from the Automation Fields are then used to set their respective values in the temporary metadata array.
 - vii. An ArrayList is created with the temporary array contents, this ArrayList represents the files metadata.
 - viii. The new basic metadata schema list is then added to the Main Class's metadataFields list and the ArrayList containing the temporary array contents is added to the Main Class's metadata list.
 - ix. The Main Class's filesInDataset, idAndFileTypeList, listOfFilePathsInDataset and listOfFileCorrespondingPIDSInDataset lists are also added to with their respective data.

Once all supported files in the dataset directory have been processed the DataDealer Class's filesToObjectArray method is called. This creates an Object[][] from the newly created Main Class's metadata lists. The populated Object[][] is then used to set the data of the Main Class's table model. The model of the Main Class's table is then set to the newly modified table model using the JTable setModel method. Once the table model of the table has been set the fireTableDataChanged method is called on the Main Class's booleanTableModel to ensure that the new table data is displayed to screen.

Validation is required for this feature as Fedora insists that PIDs follow a specific format. When importing a dataset with an incorrectly formatted Identifier Prefix (Automation Field), feedback is presented to the user via the console. The Identifier Prefix is compared against the following regular expression:

```
([A-Za-z0-9]/-/\.)
```

If the identifier prefix matches the regular expression import proceeds.

Exporting of Metadata

The Export Metadata feature is present in the Metadata Management Tools 'File' menu. Exporting metadata consists of prompting the user for a dataset directory that is then used as the export location. During this iteration, the goal was to begin outputting an XML file for each metadata item in the dataset. Furthermore, the XML file produced needed to follow the structure of the FOXML schema. This iteration focused on the creation of the Dublin Core data stream, containing nine fields of use, namely: Description, Identifier, Title, Coverage, Creator, Date, Format, Source and Type. Additionally, the 'Publish' column in the metadata table (See Figure 3.12 on the following page) is used to determine if a FOXML file should be created for the metadata item.

An example FOXML file produced during the first iteration is illustrated in Appendix A2. Each digital object is required to have its own unique identifier, the PID field is used to represent the unique identifier. Fedora PIDs are required to be in a set format that conforms to the following regular expression:

```
([A-Za-z0-9]/-/\.)+:(([A-Za-z0-9]/-/\./~/_/(\%[0-9A-F]{2}))+
```

During the import process, PID's are produced that conform to the above regular expression.

The XMLWriter Class's writeXMLFile method is used to write a FOXML file containing the metadata pertaining to a single file.

Sorting the Metadata Set

The ability to sort the metadata set is of great importance as it optimizes the navigation through the metadata set. Additionally, sorting the metadata set empowers other functionality which rely on using the Table Selection Mechanism to select multiple metadata items. It empowers such functionality by making it easy to group the metadata into categories before multiple selection occurs. As sorting the metadata set is expected to be a frequently used feature and it directly effects the ordering of metadata items in the table, it appears above the table. The metadata set can be sorted in four different ways, namely by: Identity, File, Tag and Contains.

Identity

Orders the metadata in ascending order by PID.

File

Orders the metadata in ascending order by file extension

Tag

Sort by an inputted value in a specific field contained within the metadata items. For a metadata item to match the value of the specific field, the metadata's value must equal the inputted value. Orders the matched metadata items first highlighted in grey and the rest of the metadata items afterwards. This is illustrated below in Figure 3.12.



Sort by			
<input type="radio"/> Identity	<input type="radio"/> File	<input checked="" type="radio"/> Tag	<input type="radio"/> Contains
Identifier	uctnew:0000007		
Pub	Metadata		Status
<input checked="" type="checkbox"/>	uctnew:0000007 for [Photogrammetric Submit/SUD_7211_s.jpg]		57.1%
<input checked="" type="checkbox"/>	uctnew:0000001 for [FINAL/temple_100_3MII_Faces.ply]		61.9%
<input checked="" type="checkbox"/>	uctnew:0000002 for [FINAL/temple_200_3MII_Faces.ply]		61.9%

Figure 3.12: Tag Sort

Contains

Sort by an inputted value in a specific field contained within the metadata items. For a metadata item to match the value of the specific field, the metadata's value must contain the inputted value. Orders the matched metadata items first highlighted in grey and the rest of the metadata items afterwards.

The identity and file sorts were performed using the Java Collections sort method. A custom comparator was needed for both sorts in order to compare the elements in the Main Class's idAndFileTypeList. This was due to the idAndFileTypeList containing String Arrays. This was achieved by creating a new Comparator and overriding its compare method to compare the first indexes of the String Arrays for the identity sort and the second indexes for the file sort. The tag and contains sorts required the Main Class's metadataFields and metadata lists to be traversed and added to two separate lists, namely matched and unmatched. Thereafter they were concatenated to form the sorted metadata lists.

Table Selection Mechanism

The core mechanism for metadata item selection is table selection. Table selection is a fundamental feature of the Metadata Management Tool. When a metadata item in the table is selected, its corresponding metadata and thumbnail image are displayed. Thus it is the core mechanism involved in navigating through metadata items. The table selection mechanism also needed to facilitate selection while checking and unchecking a metadata items publish check box. Additionally, the table selection mechanism needed to support multiple selection of metadata items, as it would be used by other features in the next iteration of development. Furthermore, table selection needed to be limited to rows as a row would contain information pertaining to one metadata item.

In order to facilitate all of the table selection requirements, the approach involved creating a BooleanTableModel Class that extended the AbstractTableModel. This would enable the use of a custom Table Model that could be used to achieve the required functionality. A Table could then be created using the BooleanTableModel as follows:

```
BooleanTableModel() booleanTableModel = new BooleanTableModel();  
JTable table = new JTable(booleanTableModel);
```

Row selection could then be enabled by setting `table.setRowSelectionAllowed(true)` and column selection could be disabled by setting `table.setColumnSelectionAllowed(false)`. By default, Java tables support multiple selection of columns and rows, with column selection disallowed, multiple selection would only work with rows, as required. This is illustrated below in Figure 3.13.



Pub	Metadata	Status
<input checked="" type="checkbox"/>	uctnew:0000001 for [FINAL_temple_100_3Mil_Faces.ply]	61.9%
<input checked="" type="checkbox"/>	uctnew:0000002 for [FINAL_temple_200_3Mil_Faces.ply]	61.9%
<input checked="" type="checkbox"/>	uctnew:0000003 for [FINAL_temple_300_0.7Mil_Faces.ply]	61.9%

Figure 3.13: Table Selection Mechanism

Editing Metadata

Editing metadata is one of the manual tasks users will need to perform to complete the remaining metadata field values. When a metadata item is selected using the Table Selection Mechanism, its corresponding metadata is displayed on the right hand side of the screen in the Overview tab. The corresponding metadata is displayed in metadata field name and value pairs. The metadata values are editable and provide instant feedback via the table Status column that displays the percentage of complete metadata field values for a metadata item (See Figure 3.13).

The metadata that is displayed in the Overview tab (See Appendix D1) is dynamically created when a row in the table is selected, this is accomplished with the use of the GUIModifier Class's populateMetadataPanel method. The table row index is used to lookup the metadata items associated metadata in the Main Class's metadataFields and metadata lists in order to dynamically populate the metadata panel.

Metadata Item Image Preview

When selecting a metadata item using the Table Selection Mechanism, if the metadata item is an image type or contains an associated preview image the image is displayed in the bottom left corner of the Main application screen. Preview images for types of data such as Individual Laser Scans are assumed

to be located next to the data item itself. Preview images are named using the actual digital objects name with the addition of appending “_s” to the filename and making use of the .jpg file extension.

This feature includes an image preview manipulation toolbar. The toolbar contains a number of image preview manipulation features, namely: Previous Image, Next Image, Zoom-in, Zoom-out, Best-fit and Normal-size. These features have icons associated with them that relate to the function of the manipulation. Additionally, a hover tooltip was added to each of the features that provides the user with feedback of the manipulation function. The Image preview and tooltip feedback is illustrated below, in Figure 3.14.



Figure 3.14: Metadata Item Image Preview – Tooltip Feedback

Console

In order to provide the user with additional feedback throughout their use of the Metadata Management Tool console feedback is provided. The console is located in a scrollable pane on the right side of the screen in the Console tab. An illustration of typical console feedback is provided below in Figure 3.15.

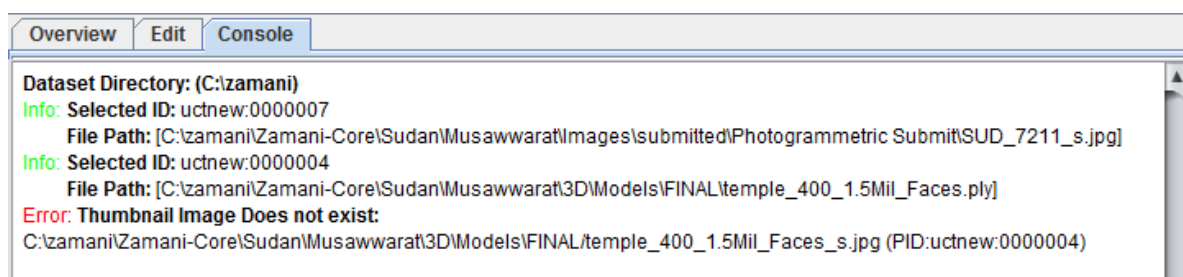


Figure 3.15: Console Feedback

Automated Metadata Extraction

The automatic extraction of metadata from digital objects is fundamental as it requires no interaction from the user. There will be a large number of digital objects that need to have associated metadata, automating the extraction of as many of the needed metadata fields as possible is essential.

Unfortunately, out of the 17 core metadata schema fields only two of the fields values could be automatically extracted from a digital objects metadata, namely: Date and Format. Both of these fields are extracted from a digital object using Java’s File Class. The date can be extracted with the help of the File Class’s lastModified() method. The format being the mime type of the object can be calculated by getting the digital objects file name using the File Class’s getName() method. By extracting the extension from the file name, the mime type can be calculated with the use of the FileTypes Class’s getFormat method.

Set Automation Fields & Field Dictionary

Overview	Edit	Console
Automate		
Identifier Prefix	uctnew	
Source	University of Cape Town, Geomatics Department	
Coverage Spatial Country	Sudan	
Coverage Spatial Name	Musawwarat es-Sufra	
Coverage Spatial Structure	Great Enclosure	
Fields	Creator	
Items	Ruther, Heinz	
	[UPDATE] [DELETE]	
	[ADD]	

Figure 3.16: Preset Automation Fields

Due to the small degree of useful metadata that could be automatically extracted from digital objects an alternative approach, namely preset automation fields was devised (See Figure 3.16). An ‘Edit’ tab on the Main application screen was created to house the automation fields. Due to the frequency of use of the preset automation fields, it was decided that for the sake of accessibility, the automation fields would be located in a tab on the Main application screen (See Figure 3.16).

The automation fields consist of five of the 17 fields contained in the Core Metadata Schema. Additionally, the “Identifier Prefix” field is used as a prefix for automatically generated Identifiers for the metadata items. Furthermore, the feasibility of an automation dictionary was explored. The dictionary consisted of the remaining fields from the Core Metadata Schema that were not used for the automation fields, along with their associated dictionary values. The dictionary values for a specific field are contained within the Items dropdown. When a field is selected in the Fields dropdown, the Items dropdown is updated with the field’s associated dictionary values. The dictionary values for each field can be updated, deleted or added to. Additionally, the dictionary needed to be stored as a text file internally in the application so that the dictionary remains persistent throughout sessions.

The purpose of the automation dictionary was to enable a faster mechanism than manually inputting the same field values for a large number of metadata items in the dataset. It was identified that for some of the Core Metadata Schema fields the values inputted while editing a collection of metadata would consist of only a few combinations. An example of one such field is the Type field. The Zamani dataset consists of a number of types that are not added to often. Thus, when editing a metadata item, the fields that have values associated with them in the automation dictionary should be displayed as an editable dropdown list as opposed to the standard input box. There is however one exception, for the Creators field, if there are multiple creators in the automation dictionary, the datasets metadata items should be created with multiple creator input boxes that are populated with the creators from the automation dictionary. During this iteration, due to time constraints, only the exception was implemented, thus only the Creator field in the field dictionary is effectively being used.

Thus seven of the 17 fields of a metadata item could be automatically set by pre-setting the respective automation fields, automation dictionary and using the field values during the import metadata process. This does not take into account a field dictionary that contains multiple creators. With the combination of this and the Automated Metadata Extraction, a total of 9 out of the 17 Core Metadata Schema fields could automatically be set during the importing of metadata. Thus 53% of a dataset’s metadata could be created automatically, this was a considerable increase over the 11.8% produced by the Automated Metadata Extraction alone. The features lacking from the proposed field dictionary

would be implemented in the second design iteration to enable easier modification of frequently occurring metadata field values.

Java Applet Security Constraints

Applets have a security model that influences how the application runs. There are three different security models for applets: signed applets, unsigned applets and self-signed applets. A signed applet, once approved by a user has full access to the users file system. However, signing an applet requires a certificate from a Certificate Authority Server, which is often expensive. Self-signed applets, applets that are signed by the developer, provide full access to user's file systems but may potentially pose a security risk. Due to the potential security risk, the user is provided with a warning when a self-signed applet is requesting authorization. Additionally, the user needs to add the applet website to their list of accepted sites by configuring java.

Due to the applet security constraints, a self-signed applet was used to demonstrate the feasibility of using a Java applet to run the Metadata Management Tool in a Web browser.

The process of imbedding a Java applet in an HTML document and avoiding local file system security constraints involved:

- i. Manifest File Creation
- ii. JAR File Creation
- iii. Self-signed JAR Creation
- iv. Imbedding a self-signed JAR containing a Java applet in an HTML Document
- v. Configuring Java

The process outlined above is explained in detail in Appendix A1.

3.5.2. Evaluation

An evolutionary prototype was produced to demonstrate the feasibility of a Web-based Metadata Management Tool. The prototype included the core requirements of the Metadata Management Tool as well as any functionality that was identified as having technical challenges associated with it (See section 3.4.1). The evolutionary prototype was demonstrated to the project supervisor (Hussein Suleman) and second reader (Maria Keet). The prototype was additionally demonstrated to the Zamani team to demonstrate the feasibility of the system as well as gather additional requirements and any design issues that were present. The prototype illustrated that a Web-based Metadata Management Tool was feasible and that all of the requirements could be implemented. Additionally it was valuable to develop a more in depth understanding of the problem.

A number of issues were identified and are as follows:

Look & Feel

The current GUI look and feel is bland and lacks aesthetic appeal.

Tab Naming Convention

The tab names were found to incorrectly describe their functions.

Tag & Contains Sort Input Fields

The input fields required for Tag and Contain sorts should only be enabled when the Tag or Contains sort radio button is selected.

A number of additional tasks would need to be performed by the Zamani team, an additional subset of functionality was identified and is as follows:

Sort by Status

The ability to sort by status was identified as an important sort filter.

Add Files & Directories

It is fundamental that the current metadata set can be added to.

Open Files & Directories

There is also the need to replace the current metadata set with a new set of metadata.

Persistence

The current metadata set needs to be able to be saved and metadata sets that have been saved need to be able to be re-loaded.

Associate Files

The system needs to be able to deal with digital objects in a metadata set being moved to a different directory. Thus, the system needs to be able to associate the newly moved files with the previously created metadata.

Import Camera Calibration Settings

The ability to import Camera calibration settings from a text file and add them to the metadata of one or more selected metadata items. The camera calibration settings should only be added to metadata items that are associated with image files.

Remove Camera Calibration Settings

Camera calibration settings need to be able to be removed from one or more metadata items that contain camera calibration settings that were accidentally added to.

Import Co-ordinates

Import metadata position data from a text file consisting of lines of space separated triplets (filename latitude longitude). Any metadata items with filenames that match a triplet from the metadata position data should be automated to contain the position data.

Transform Aluka Metadata

The metadata that the Zamani team compiled for Aluka needs to be able to be transformed into the new Metadata Management Tool save file that can be used to re-load such metadata into the tool.

3.6. Iteration 2: Metadata Management Tool Design

The second iteration uses the first iteration as a base to build upon. The objective of the second iteration is to refine the first iteration and add an additional subset of functionality.

3.6.1. Design and Implementation

The second iteration follows a more rigorous process of design, building on the first iteration. The primary goal is to refine and extend the Metadata Management Tool with the additional functionality presented in Section 3.4.2. Additionally, this iteration aims to improve on the usability issues presented in the previous iteration

The additional functionality was identified and added to the core functionality of the Metadata Management Tool, this is illustrated in the use case diagram in Figure 3.5.

3.6.1.1. Revised Core Metadata Management Tool Requirements

The core requirements of the Metadata Management Tool were revised and are illustrated in the use case diagram in Figure 3.17. The use cases outlined in red represent the additional functionality identified during the first iteration.

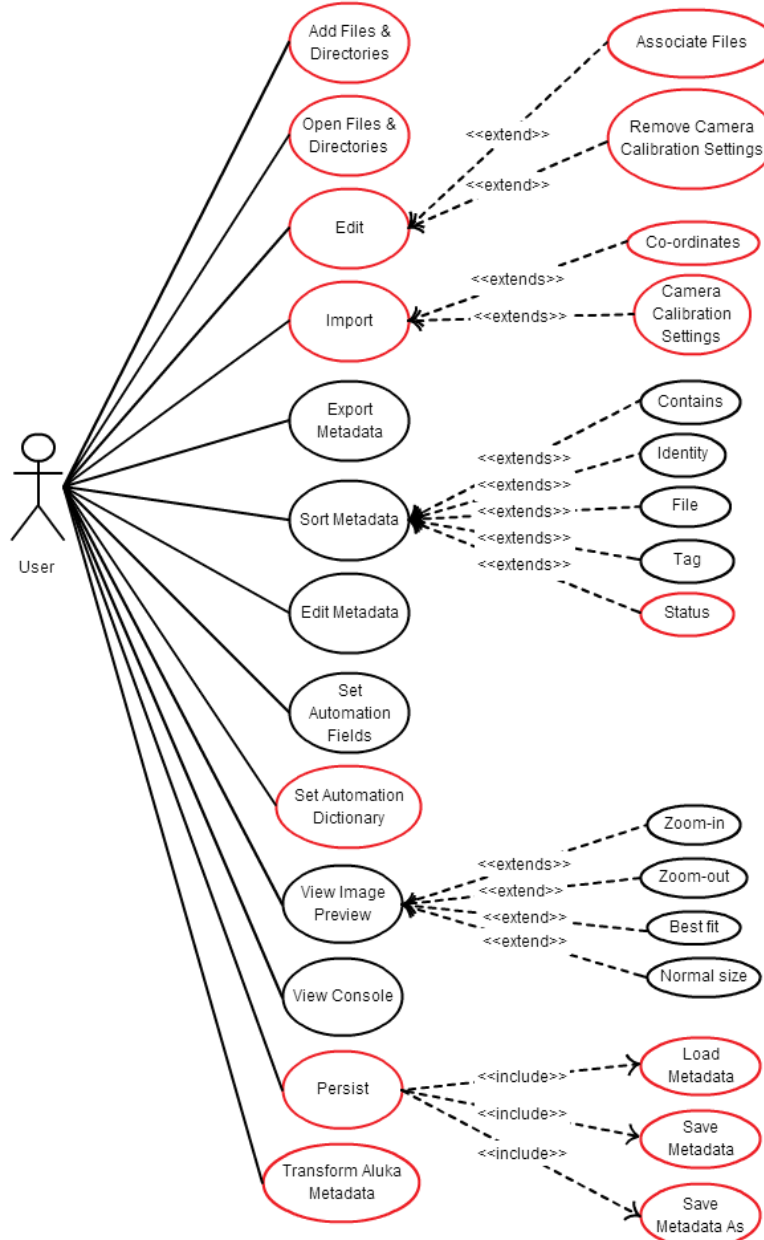


Figure 3.17: Use case diagram of revised Metadata Management tool functionality

3.6.1.2. Detailed Overview

During the second iteration modifications were made to the Main, DataDealer and XMLWriter Classes, this is illustrated in Figure 3.18 in blue. The detailed overview is provided below the illustration.

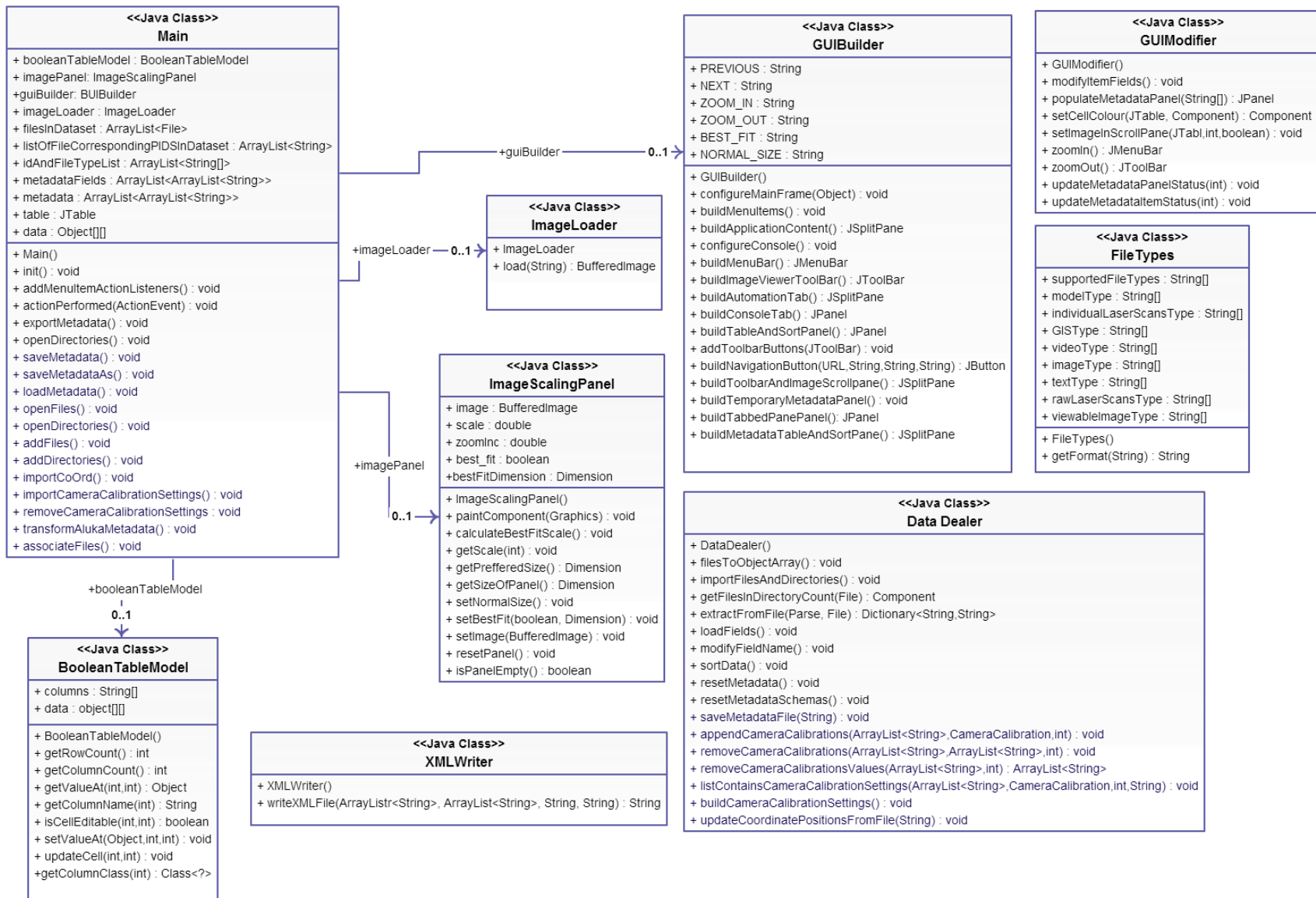


Figure 3.18: Metadata Management Tool revised functionality Class Diagram

Class Definitions and Methods

1. Main

Action Event Methods:

- openFiles()
 - Uses the DataDealer Class's importFilesAndDirectories method to import the selected files. Uses an isDirectory parameter value of false and reset value of true. File indices and counts are reset prior to calling the method.
- openDirectories()
 - Uses the DataDealer Class's importFilesAndDirectories method to import the files in the directory. Uses an isDirectory parameter value of true and reset value of true. File indices and counts are reset prior to calling the method.
- addFiles()
 - Uses the DataDealer Class's importFilesAndDirectories method to import the selected files. Uses an isDirectory parameter value of false and reset value of false.
- addDirectories()
 - Uses the DataDealer Class's importFilesAndDirectories method to import the files in the directory. Uses an isDirectory parameter value of true and reset value of false.
- importCoords()
 - Uses the DataDealer Class's updateCoordinatePositionsFromFile method to import coordinate data.
- importCameraCalibrationSettings()
 - Uses the DataDealer Class's appendCameraCallibrations method to add camera calibration settings to the selected table items.
- removeCameraCalibrationSettings()
 - Uses the DataDealer Class's removeCalibrationSettings method to remove camera calibration settings from the selected table items.
- transformAlukaMetadata()
 - Creates a transform Aluka metadata popup window using the PopupWindow Class's popupTransformAlukaMetadataWindow method.
- associateFiles()
 - Creates an associate files popup window using the PopupWindow Class's popupAssociateFilesWindow method.
- saveMetadata()
 - Checks if the metadata set has been saved, if it has, the previously saved file is overwritten. If not, the user is prompted for a new file path and a new .metafy file is created. Uses the DataDealer Class's saveMetadataFile method to save a serialized .metafy file.
- saveMetadataAs()
 - Prompts the user for a file path and then uses the DataDealer Class's saveMetadataFile method to save a serialized .metafy file.
- loadMetadata()
 - Creates a load metadata popup window using the PopupWindow Class's popupLoadMetadataProgressWindow method to provide feedback of the current load progress. The Actual loading of the file is performed by the BackgroundWorker Class's nested LoadMetadataWorker Class.
- exportMetadata()
 - Creates an export metadata popup window using the PopupWindow Class's popupExportMetadataWindow. Uses the XMLWriter Class to write a collection of FOXML files.

2. *DataDealer*

Methods:

- `importFilesAndDirectories(bool isDirectory, bool reset)`
 - Imports the files in a directory or selected files and stores the created metadata in the necessary data structures (Main Class's static variables). Sets the selection mode of the file chooser to directories or files depending on `isDirectory`'s value. Resets the metadata data structures if `reset` is true. This method recursively iterates through a directory and its sub-directories.
- `saveMetadataFile(String filePath)`
 - The `Serializer Class` is used to serialize a `DataStore` object that stores data structures that are needed for persistence. The serialized object is written to disk using the specified file path.
- `appendCameraCalibrations()`
 - Checks to see if an item is an accepted calibration image type, if it is, the camera calibration settings are appended to the metadata item. Appending the camera calibration settings involves adding the camera calibration field names to the item in the `metadataFields` list and adding the corresponding values from the `CameraCalibration` object to the metadata list.
- `listContainsCameraCalibrationSettings(ArrayList<String> list)`
 - Checks to see if the given list contains the first camera calibration setting, namely "Calibration Date". Returns true if contained and false if not.
- `removeCameraCalibrations(int indexToRemove)`
 - The item at the specified index to remove is checked to determine if it contains camera calibration settings using the `listContainsCameraCalibrationSettings` method. If it does contain camera calibration settings the index of the first calibration setting is found in the `metadataFields` list. There are sixteen camera calibration settings. Due to the data structure being an `ArrayList`, the calibration settings are removed from the `metadataFields` and `metadata` lists by removing the index of the first calibration setting sixteen times.
- `updateCoordinatePositionsFromFile(String coordFile)`
 - Creates a `FileCoordinatesList` object from the given `coordFile`. All of the files in the `filesInDataset` list are then iterated through, comparing the filenames to those contained within in the `FileCoordinatesList`. This is done with the use of the `FileCoordinatesList Class`'s `fileNameSearch` method that returns the index if the filename is contained within the list of file coordinates, else it returns -1. If the index returned is not -1, its corresponding position data (latitude and longitude) are set for the metadata item that was matched.

3. *XMLWriter* – The `XMLWriter Class` outputs a site hierarchy of FOXML files using `writeXMLFile`, `writeXMLCountryCollectionFile` and `writeXMLSiteCollectionFile`.

Methods:

- `writeXMLFile(ArrayList<String> metadataFields, ArrayList<String> metadata, String outputPath, String collectionPID, String filePathServer, String fileSize)`
 - Writes a metadata item FOXML file to the specified `outputfilePath` location, using an `RDF isMemberOf` relationship using the given `collectionPID`. The FOXML file contains the `POSITION` and `FILE` datastream for the metadata item, conditionally if the item is an image type it may also contain a `CALIBRATION` datastream.
- `writeXMLCountryCollectionFile(String filePath, String collectionPID, String country)`
 - Writes a country collection FOXML file to the specified `filePath` location, using an `RDF isMemberOf` relationship with the `Zamani Project collection PID` (`zamaniProjectCollection:1`) and an `isCollection true` attribute.

- writeXMLSiteCollectionFile(String filePath, String collectionPID, String countryCollectionPID, String siteName, String siteLatitude, String siteLongitude, String country)
 - Writes a site collection FOXML file to the specified filePath location, using an RDF isMemberOf relationship with the given countryCollectionPID and an isCollection true attribute. The FOXML file contains a SITE datastream to store the country, name and position of the site.

During the second iteration a number of Classes were created. The BackgroundWorker and PopupWindow are two notable Classes. The relationship between the two Classes is illustrated in Figure 3.19. A detailed overview of the Classes is provided after the illustration

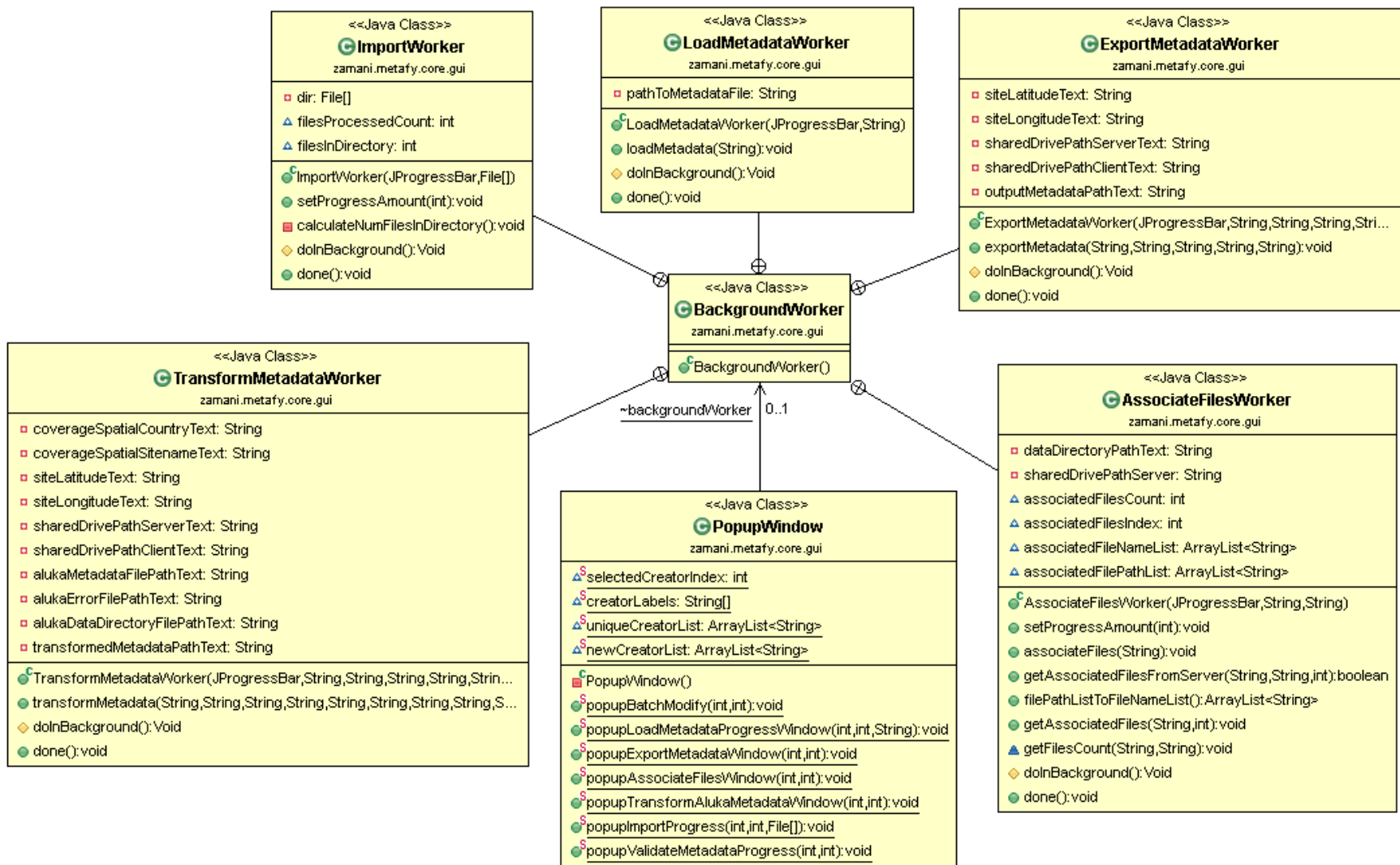


Figure 3.19: Class diagram of BackgroundWorker Class and PopupWindow relationship

The BackgroundWorker Class contains a number of nested Classes, namely: ImportWorker, LoadMetadataWorker, ExportMetadataWorker, AssociateFilesWorker and TransformMetadataWorker. As the names imply, these Classes are background worker Classes for some of the functionality that is computationally expensive. The PopupWindow has an instance of the BackgroundWorker Class and instantiates the background worker Classes when appropriate.

Class Definitions

1. *BackgroundWorker* – The BackgroundWorker Class is a container Class that has several nested Classes that all extend SwingWorker and run on background threads.

Nested Classes:

- 1.1. *ImportWorker* – The ImportWorker Class is used to import the associated metadata of files. It starts by calculating the number of files in the directory so that feedback can be provided by a relatively accurate progress bar when files are being processed. It then processes all of the files or directories that were selected with the use of the DataDealer Class's importFilesAndDirectories method. Once done the imported files are converted to an object array with the use of the DataDealer Class's filesToArray method. The table's model is set to the newly updated table model and the fireTableDataChanged method is called on the booleanTableModel.
- 1.2. *LoadMetadataWorker* – The LoadMetadataWorker Class is used to load a .metafy file containing a serialized DataStore object. A Deserializer object is used to deserialize the .metafy file into a MetadataStore object. The Main Class's static variables that are metadata related are then loaded to their previous state using the MetadataStore object. The table's model is then set to the loaded table model and the fireTableDataChanged method is called on the booleanTableModel.
- 1.3. *ExportMetadataWorker* – The ExportMetadataWorker Class is used to export a collection of metadata. Firstly, a site and country collection FOXML file is created with the use of the XMLWriter Class's writeXMLCountryCollectionFile and writeXMLSiteCollectionFile methods. It then iterates through all of the metadata items and makes use of the XMLWriter Class's writeXMLFile method to create a FOXML file for each metadata item.
- 1.4. *AssociateFilesWorker* – The AssociateFilesWorker Class is used to associate a new directory of files with a metadata set. It compares the file names present in the given directory to those of the metadata set, if there is a match the metadata item in the metadata set is associated with the new file location. As associating files is a computationally expensive task, the count of the given directory to associate is calculated first so that accurate feedback can be provided via a progress bar.
- 1.5. *TransformMetadataWorker* – The TransformMetadataWorker Class is used to transform metadata created for Aluka to a .metafy file that can be re-loaded. It makes use of an ErrorFormatter object to extract a file path list and PID list from the supplied Aluka error file. Additionally, an XMLReader object is used to extract three lists, namely: metadataFields, metadata and idAndFileTypesList. The metadata list is then iterated through and the PIDs from the PID list and metadata item PIDs are compared. When a matching PID is found its new file location is calculated using the supplied Aluka data directory path and the path found in the PID lists associated file path list.

2. *PopupWindow* – The `PopupWindow` Class contains the popup window methods used for the various nested `BackgroundWorker` Classes.

Methods

- `popupImportProgress`
 - Creates a Popup window to display the import progress bar and instantiates an `ImportMetadataWorker` object and calls its `execute` method.
- `popupLoadMetadataProgressWindow`
 - Creates a Popup window to display the load metadata progress bar and instantiates a `LoadMetadataWorker` object and calls its `execute` method.
- `popupExportMetadataWindow`
 - Creates a Popup window with a number of input fields and an ‘Export Metadata’ button. The input fields are namely: Site latitude, longitude, path to shared drive (server), path to shared drive (client) and the output directory path. The ‘Export Metadata’ button’s action listener provides validation for the input fields to ensure that all of the input fields have been filled out and that the paths are valid before exporting takes place. If the validation passes, the export metadata progress bar is made visible and a `LoadMetadataWorker` object is instantiated and its `execute` method is called.
- `popupAssociateFilesWindow`
 - Creates a Popup window with two input fields and an ‘Associate’ button. The input fields are namely: path to shared drive (server) and the associate files directory path. The ‘Associate’ button’s action listener provides validation for the input fields to ensure that both of the input fields have been filled out and that the paths are valid before association occurs. If the validation passes, the associate files progress bar is made visible and an `AssociateFilesWorker` object is instantiated and its `execute` method is called.
- `popupTransformAlukaMetadataWindow`
 - Creates a Popup window with a number of text-based input fields and a ‘Transform’ button. The input fields are namely: coverage spatial country, coverage spatial name, site latitude and longitude. Additionally, a number of paths are required via text-based input or the browse button, namely: shared drive (server), shared drive (client), Aluka metadata file, Aluka error file, Aluka data directory and transformed metadata. The ‘Transform’ button’s action listener provides validation for the input fields to ensure that text has been inputted and that the paths are valid before the Aluka metadata is transformed. If the validation passes, the transform Aluka metadata progress bar is made visible and a `TransformMetadataWorker` object is instantiated and its `execute` method is called.

Three helper Classes were needed for the Import features, namely: FileCoordinatesList, FileCoordinates and CameraCalibration. The three Classes are illustrated in Figure 3.20. A detailed overview of the Classes is provided after the illustration.

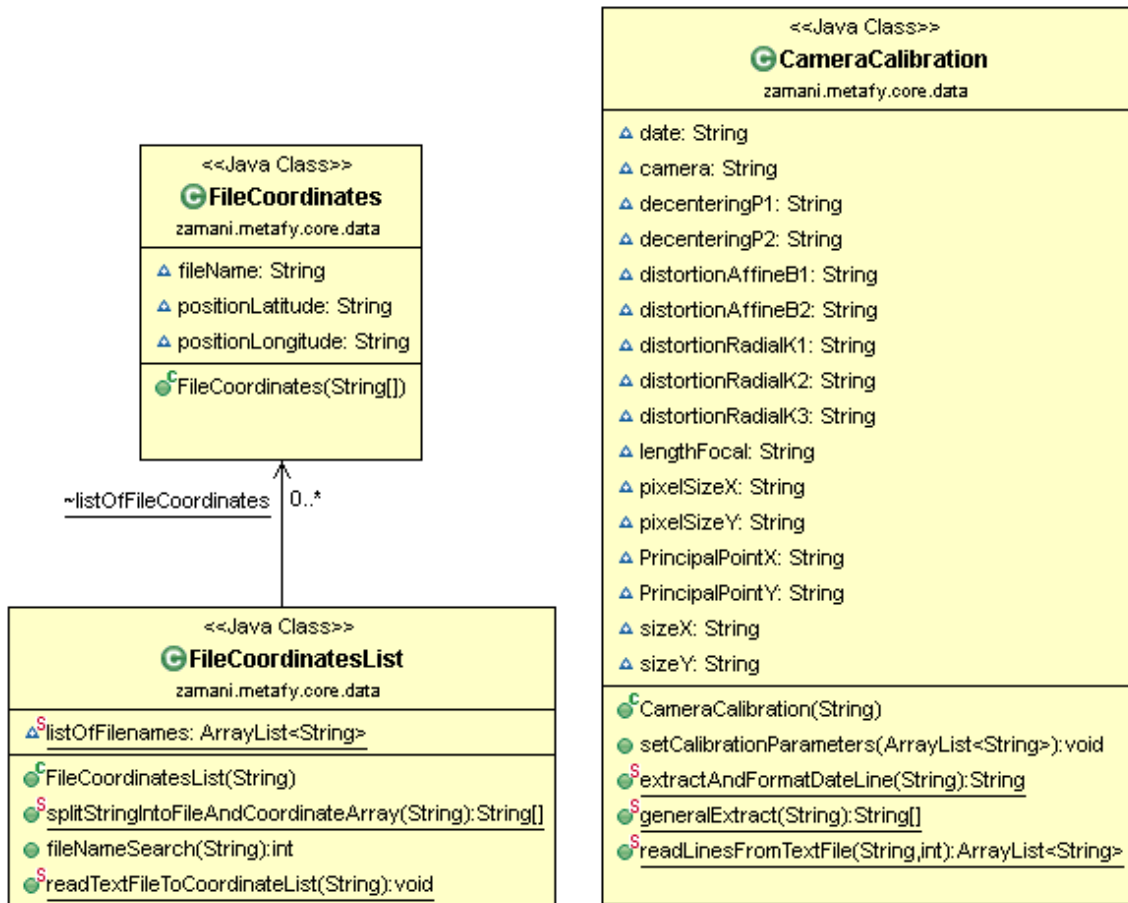


Figure 3.20: Class diagram of helper Classes used in Import features

Class Definitions

1. *FileCoordinateList* – The FileCoordinateList Class is a helper Class that reads a text file of space separated co-ordinate triplets, consisting of file name, latitude, and longitude into a FileCoordinateList object containing a list of the co-ordinate triplets.
2. *FileCoordinates* – The FileCoordinates Class is a helper Class that is used to store co-ordinate triplets in a FileCoordinates object.
3. *CameraCalibration* – The CameraCalibration Class is a helper Class that is used to read camera calibration settings from a text file and store them in the CameraCalibration object.

Two helper Classes were needed for the TransformMetadataWorker, namely: XMLReader and ErrorFormatter. The two Classes are illustrated in Figure 3.21 and a detailed overview of the Classes is provided after the illustration.

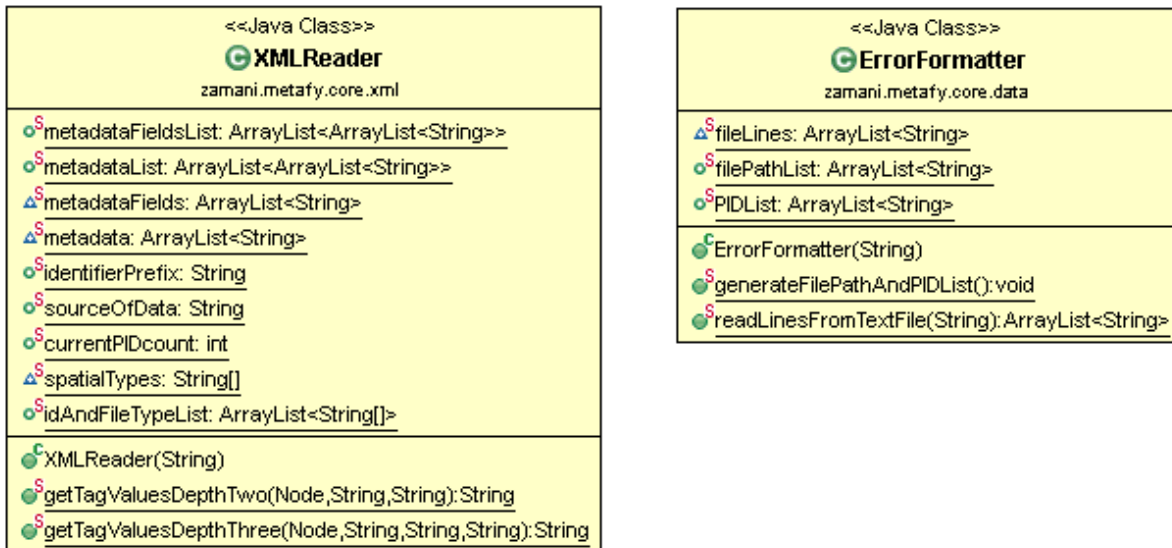


Figure 3.21: Class diagram of helper Classes used to Transform Aluka metadata

Class Definitions

1. *XMLReader* – The XMLReader Class is a helper Class that is used in the TransformMetadataWorker Class. It serves to read an XML file produced for Aluka into data structures that follow the same structure as the Main Class’s metadata variables.
2. *ErrorFormatter* - The ErrorFormatter Class is the second helper Class that is used in the TransformMetadataWorker Class. It serves to read an Aluka error file into two lists, namely: filePathList and PIDList.

3.6.1.3. Usability Issues

During the first iteration a number of usability issues were identified, the system was altered to improve on these issues as follows:

Look & Feel

The most noticeable visual improvement to the system was the change in look and feel. A hybrid look and feel was designed to incorporate look and feel styles from the Nimbus and default Java look and feel. This was achieved by storing a set of the default look and feel settings, setting the look and feel to Nimbus and then modifying certain styles by replacing them with those present in the stored set. The main style components that were modified were: ToolBar, SplitPaneDivider and Table. The altered look and feel makes the tool more appealing (See Figure 3.22).

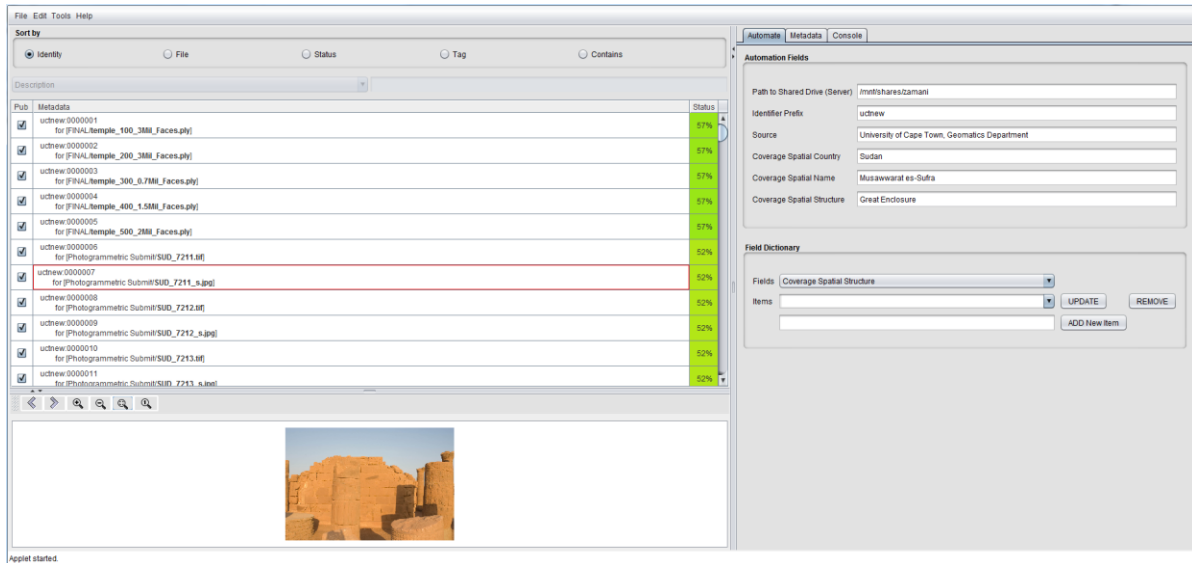


Figure 3.22: Look and Feel

Tab Naming Convention

In the first iteration the Tabbed Pane present on the right hand side of the screen was identified as containing tab names that incorrectly described their purpose. The tab names present in the first iteration were: Overview, Edit and Console. Firstly, the Overview tab served as the container for an items metadata. Secondly, the Edit tab was used to set Automation Fields and the Field Dictionary. Lastly, the Console tab was used to provide console oriented feedback. Due to the functions of these tabs not changing their names were changed to represent their purpose. The Overview tab was renamed to the Metadata Tab as it is the container for an items metadata. The Edit tab was changed to Automate as it houses the Automation Fields and Fields Dictionary that are used to further automate the process of metadata creation. The Console tab was not changed as it represents its function. Additionally, the Automate tab was formatted into two distinct categories: Automation Fields and Field Dictionary. These changes are illustrated in Figure 3.22.

Tag & Contains Sort Input Fields

It was identified that the input fields used for the Tag and Contain sorts should only be enabled when either one of them is selected and on the contrary should be disabled when one of the other sort methods is selected. This ensured that the user is only able to use the given functionality when it is able to work. This is illustrated in Figure 3.23 and Figure 3.24 below.

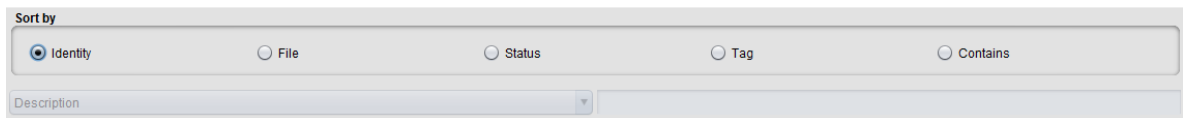


Figure 3.23: Disabled Tag and Contain Sort Input Fields

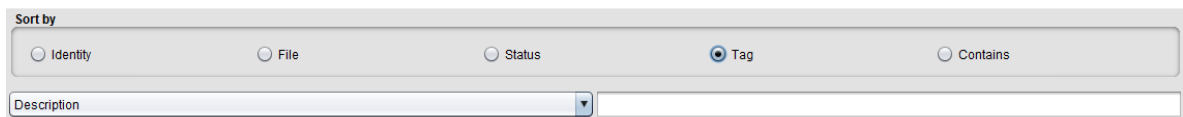


Figure 3.24: Enabled Tag and Contain Sort Input Fields

Menu

A number of changes were made to the menu, including adding a number of additional menus to the menu bar, the menu bar was altered to include four menu categories: File, Edit, Tools and Help. Additionally, descriptive menu icons were added to menu items and ellipses were used to denote that more information is needed to perform the action (See Figure 3.11).

The various functions of the menu items are discussed in Additional Functionality section to follow.

The File's Import menu item present in the first iteration was split into four menu items that followed a naming convention that the Zamani team were familiar with, namely: Open Files, Open Directories, Add Files and Add Directories. This is illustrated in Figure 3.25.

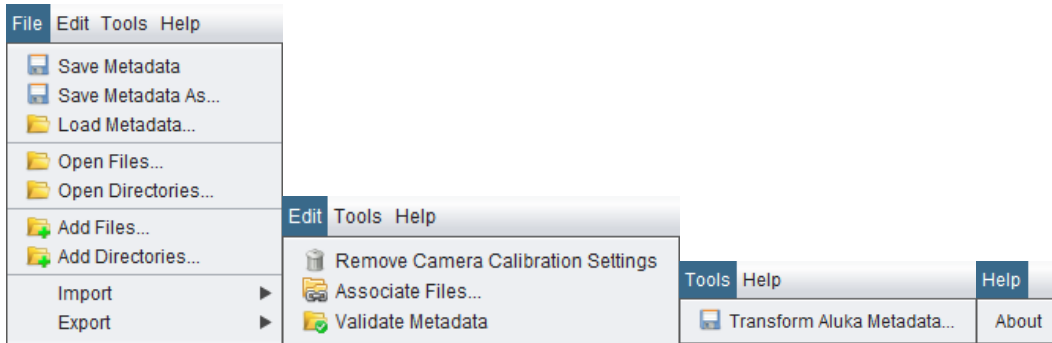


Figure 3.25: Menu Categories and items

3.6.1.4. Additional Functionality

A number of features were developed from the additional tasks the Zamani team required, below is a list of the features that were developed during the second design iteration.

Sort by Status

In the first iteration it was identified that sorting by status would be a useful addition to the other sort methods, namely by: Identity, File, Tag and Contains. Sorting by status would entail ordering the metadata in ascending order by their status. The status of a metadata item is represented by the percentage of its complete metadata fields.

Similarly to the identity and file sorts the status sort is performed using the Java Collection sort method. The sort's comparator is overridden to compare metadata based on the third index of the String Arrays contained within the idAndFileTypeList. The third index contains an up to date representation of the metadata items status.

Add Files and Directories & Open Files and Directories

The Add Files, Add Directories, Open Files and Open Directories features are present in the File menu. As the import functionality from the first iteration is identical to the required functionality for the Open Directories feature, the DataDealer class's importDirectories method was modified to importDirectoriesAndFiles and two Boolean parameters, namely isDirectory and reset were added. Depending on the value of isDirectory the user is prompted to select directories or files. Additionally, the reset value is used to determine if the DataDealer class's resetMetadata method should be called. Thus, the importDirectoriesAndFiles method can be used to cater for all four features.

Additionally, it was found that the importDirectoriesAndFiles method could take some time, thus it was decided to provide feedback of the import progress via a popup progress bar. In order to provide

accurate progress, the number of files to be processed needed to be calculated first, the `DataDealers` `getDirectoryCount` method is used for this purpose. The `BackgroundWorker` class's `ImportWorker` Class, a nested class that extends `SwingWorker` is used to house the `importDirectoriesAndFiles` method on a background thread. Additionally, it is used to update the progress bar that is created using the `PopupWindow` class's `popupImportProgress` method.

Persistence

Persistence is essential in order to ensure that metadata sets can be saved and later re-loaded, modified and added to. The `Save Metadata`, `Save Metadata As` and `Load Metadata` features are present in the `File` menu. The `Save Metadata` feature is used to save over a previously saved metadata set during the same session. The `Save Metadata As` feature is used when a metadata set is saved for the first time during a session, the user is prompted for a path and filename. The user is required to specify `.metafy` as the file extension, if an incorrect file extension is used, the user is provided with feedback via the `Console` tab. In order to reduce confusion, the `Save Metadata` feature functions similarly to the `Save Metadata As` feature if the current metadata set has not yet been saved. Additionally, once a metadata set is successfully saved, feedback is provided in the top right corner of the screen (See Figure 3.26). The `Load Metadata` feature prompts a user for a path to a previously saved (`.metafy`) file and then proceeds to deserialize the file. Additionally, as deserializing a large metadata set can take some time, it is run in a background thread and feedback is provided via a popup progress bar.

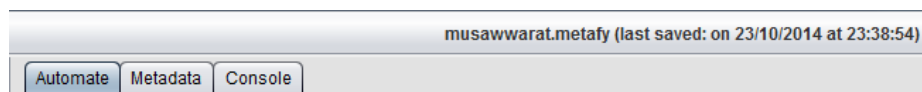


Figure 3.26: Save Feedback

All of these features make use of the `zamani.core.persistence` package consisting of three Classes, namely: `MetadataStore`, `Serializer` and `Deserializer`. The `MetadataStore` implements `Serializable` and is used to store any metadata related data structures that need to persist across sessions. In order to save metadata, a `MetadataStore` object is created and written to an `ObjectOutputStream`. Loading the metadata file is accomplished by reading the serialized `MetadataStore` object, using an `ObjectInputStream` and casting it to a `MetadataStore` object.

Associate Files

The `Associate Files` feature is present in the `Edit` menu. This feature is essential to ensuring the longevity of created metadata. It accomplishes this by its ability to associate a new directory of files with a previously created metadata set. This is needed in situations where some or all of the digital objects in a previously created metadata set have been moved to a different directory. Additionally, it ensures the re-use of the `.metafy` files produced from the `Transform Aluka Metadata` feature.

When the `Associate Files` menu item is selected, an `associate files` popup window is created. The path to the shared drive on the server is preset to its current location but is currently not made use of during this iteration. The user is required to browse for the directory to be associated, if a directory is not selected or invalid, feedback is provided below the input fields (See Figure 3.27). While files are being associated, feedback is provided via a progress bar and the `associate files` button is disabled (See Figure 3.28). Additionally, console feedback is provided for each metadata item that is associated, detailing the PID of the metadata item, the old file path and the newly associated file path.

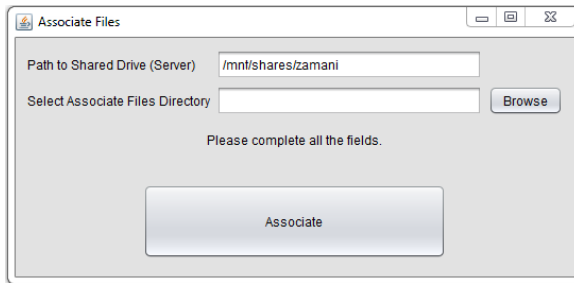


Figure 3.27: Validation Feedback

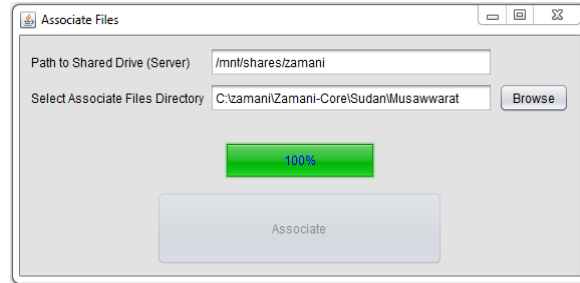


Figure 3.28: Progress Feedback

Associating a large directory of files is a computationally expensive task, thus it is run on a background thread. The `BackgroundWorker` class's `AssociateFilesWorker` class, a nested class that extends `SwingWorker` is used to associate files. Additionally, it is used to create and update the progress bar.

Import Camera Calibration Settings

The Import Camera Calibration Settings feature is present in the Import File menu's sub-menu. It is used to 'attach' camera calibration settings to metadata item/s that have been selected using the Table Selection Mechanism. When the menu item is selected, the user is prompted to browse the local file system for a text file containing camera calibration settings. The `CameraCalibration` Class is then used to create a `CameraCalibration` object containing the camera calibration settings. This object is then used to add the camera calibration settings to one or more selected metadata items. This is done using the `DataDealer`'s `appendCameraCallibrations` method. However, camera calibration settings are only relevant to image types, and thus are only added to metadata items that are of an image type. Additionally, if this feature is used on metadata item/s that already contain camera calibration settings the settings are replaced with the newly supplied text file.

Remove Camera Calibration Settings

The Remove Camera Calibration Settings feature is present in the Edit File menu. It is used to remove camera calibration settings from metadata item/s that have been selected and contain camera calibration settings.

Import Co-ordinates

The Import Co-ordinates feature is present in the Import File menu's sub-menu. It is fundamental as it automates the process of having to manually set metadata items position data. It prompts the user to browse the local file system for a text file containing lines of space separated triplets (filename latitude longitude). The `DataDealer`'s `updateCoordinatePositionsFromFile` method is used to update the position data of metadata item/s. It compares their associated file names with those contained within the `FileCoordinatesList` class's `listOfFileNames` list, if a match is found the associated metadata items position data is set to the matched triplet.

Transform Aluka Metadata

The Transform Aluka Metadata feature is present in the Tools menu. In order to re-use the metadata the Zamani team produced for Aluka the Transform Aluka Metadata feature is used. This feature was seen as a high priority feature by the team as a significant amount of time was spent creating metadata for a number of the team's documented cultural heritage sites.

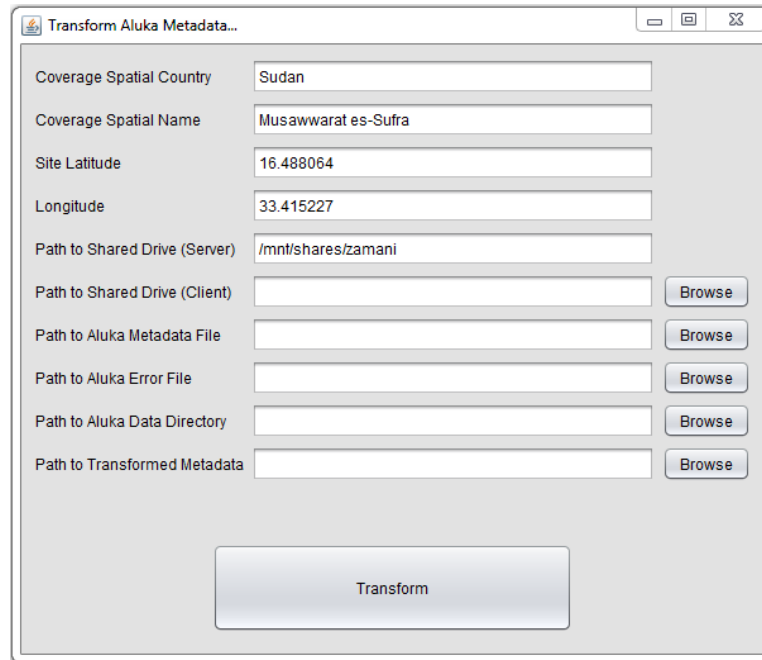


Figure 3.29: Transform Aluka Metadata Popup

When the Transform Aluka Metadata menu item is selected, a Transform Aluka Metadata popup window is created (See Figure 3.29). A number of the required fields are needed to gather information about the site, preset values are used and serve as placeholders to aid the user. Additionally, the user is required to set a number of paths: Shared Drive (Server), Shared Drive (Client), Aluka Metadata File, Aluka Error File, Aluka Data Directory and Transformed Metadata.

The path to the Shared Drive (Server) and Shared Drive (Client) is the path to the shared drive on the server and client respectively (See Section 3.2.1). The shared drive is mounted on the server, thus it is currently pre-set to the currently mounted directory for ease of use. The path to the Aluka Metadata File is the path to the XML file containing one site's metadata set. The Aluka Error File is a text file containing lines of errors (See Figure 3.30) produced by Metty-D when loading a site's metadata set that has file paths that no longer exist. The Aluka Data Directory path is the location of the site's files and the Transformed Metadata path is the output location.

```
ERROR error_message_box: The file 'F:\Sudan\Musawwarat\images\Photogrammetric Lion Temple Submit\SUD-8066.tif'
(F:/Sudan/Musawwarat/images/Photogrammetric Lion Temple Submit/uctsud0000001.tif) does not exist.
```

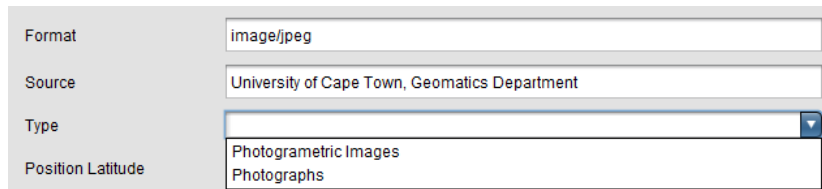
Figure 3.30: A line in the Aluka Error File

Similarly to the Associate Files feature validation is provided to ensure that all of the fields have been filled out and that the paths exist (See Figure 3.29). Additionally, a progress bar is used to display the progress of the transformation (See Figure 3.30).

The BackgroundWorker class's TransformMetadataWorker class, a nested class that extends SwingWorker is used to transform the Aluka metadata using the provided information. Furthermore, it creates and updates the progress bar on a background thread and outputs a .metafy file containing the metadata set contained within the Aluka XML file. Thus the previously created Aluka metadata can be loaded into the tool with the newly created .metafy file.

Field Dictionary

During this iteration the field dictionary created during the first iteration was expanded on. Due to some fields in the metadata sets only consisting of a few combinations of different values, the Field Dictionary was extended to enable a faster mechanism than manual text input. This was done by extending the Edit Metadata Feature, by modifying the GUIModifier class's populateMetadataPanel method. Thus, when editing a metadata item, the fields that have values associated with them in the automation dictionary are displayed as an editable dropdown list as opposed to the standard input box (See Figure 3.31).

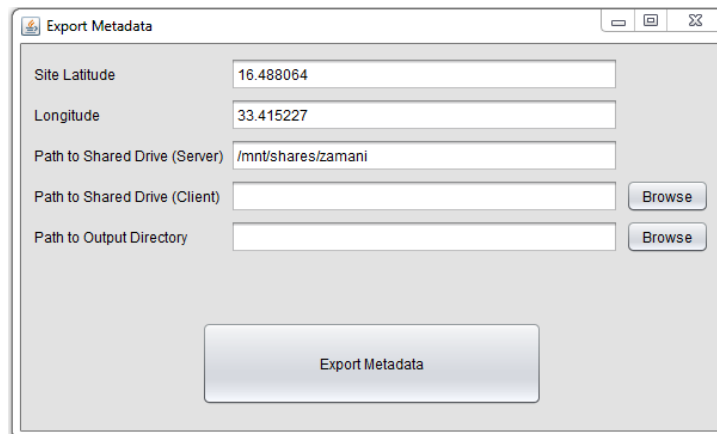


The screenshot shows a form with four fields. The 'Format' field contains 'image/jpeg'. The 'Source' field contains 'University of Cape Town, Geomatics Department'. The 'Type' field is a dropdown menu with 'Photogrammetric Images' selected and 'Photographs' visible below it. The 'Position Latitude' field is empty.

Figure 3.31: Edit Panel making use of Revised Field Dictionary

Export Metadata

During this iteration the Export Metadata feature produced during the first iteration was expanded on. During the first iteration simple FOXML files containing a single Dublin Core data stream were created for each metadata item of a collection (metadata set). This iteration expands on the simple FOXML files produced by adding a number of different data streams as well as creating a hierarchy of links amongst countries, sites and their respective data sets. This is fundamental in ensuring that the other system components have the information needed to provide their required functionality.



The screenshot shows a window titled 'Export Metadata'. It contains five input fields: 'Site Latitude' with the value '16.488064', 'Longitude' with '33.415227', 'Path to Shared Drive (Server)' with '/mnt/shares/zamani', 'Path to Shared Drive (Client)', and 'Path to Output Directory'. There are 'Browse' buttons next to the Client and Output Directory fields. A large 'Export Metadata' button is at the bottom.

Figure: 3.32: Export Metadata Popup

When the Export Metadata menu item is selected, an Export Metadata popup window is created (See Figure 3.32). A number of the required fields are needed to gather information about the metadata set of the site to be exported. The sites latitude and longitude are required for the search interface's map-based Collection Search feature explained in the third iteration and are stored in the SITE data stream of the FOXML file representing the site. The path to the shared drive on the server and client are used to calculate server side file paths for all of the metadata items associated files. The server side path of a file associated with a metadata item is stored in the FILE data stream. A CALIBRATION data stream is used to store camera calibration settings, and is added to a metadata item's FOXML file if the item's metadata contains calibration settings. Additionally, a POSITION data stream is used to store the position data of the metadata item.

Similarly to the Associate Files feature validation is provided to ensure that all of the fields have been filled out and that the paths exist. Additionally, a progress bar is used to display the progress of the export. The BackgroundWorker class's ExportMetadataWorker class, a nested class that extends SwingWorker is used to export a collection. Furthermore, it creates and updates the progress bar on a background thread and outputs the collections FOXML files to an output folder at the specified output location. The output folder can then be ingested with the use of the Archival Tools housed in the Zamani Archive Backend.

3.6.2. Evaluation

The second iteration was evaluated with the help of the Zamani Project team and the project supervisor. The evolutionary prototype was once again demonstrated. This evaluation served as a critical analysis of the Metadata Management Tool, with the sole purpose of identifying any major functional issues, design flaws and/or additional useful functionality. The prototype illustrated a functional Metadata Management Tool that could provide the Zamani team with an easy to use tool that fulfilled the tasks that needed to be accomplished.

Two additional features were identified that could provide the tool with further automation capabilities, namely:

Automated Title Generation

The ability to automatically generate titles for a metadata set. The title would be generated based on the values contained within the metadata item.

Batch Modify

The ability to modify the values of multiple metadata items simultaneously.

3.7. Iteration 3: Search Interface Design

The third iteration focused on the design and implementation of the Web-based Search Interface. The goal of this iteration was to design and implement the core Search Interface requirements. During this iteration, a significant amount of time was spent analyzing Aluka's search interface. This phase produced a fully-fledged Web-based Search Interface and Search Interface API for the Zamani Data Archive.

3.7.1. Design and Implementation

The focus of this iteration was to enable the core functionality of the Search Interface. The Search Interface would be designed to be functional and aesthetically appealing as it would serve as the home page of the Public Portal. The core functionality was identified as: text-based search, faceted search, collection-based search and Text-based search auto-complete. Additionally, an in depth view of a single search result is required, namely: Show Metadata. The Show Metadata view should also enable the user to backtrack to the collection or parent collection of the item using the Show Collection or Show Parent Collection features. The core functionality is outlined by the use case diagram in Figure 3.33.

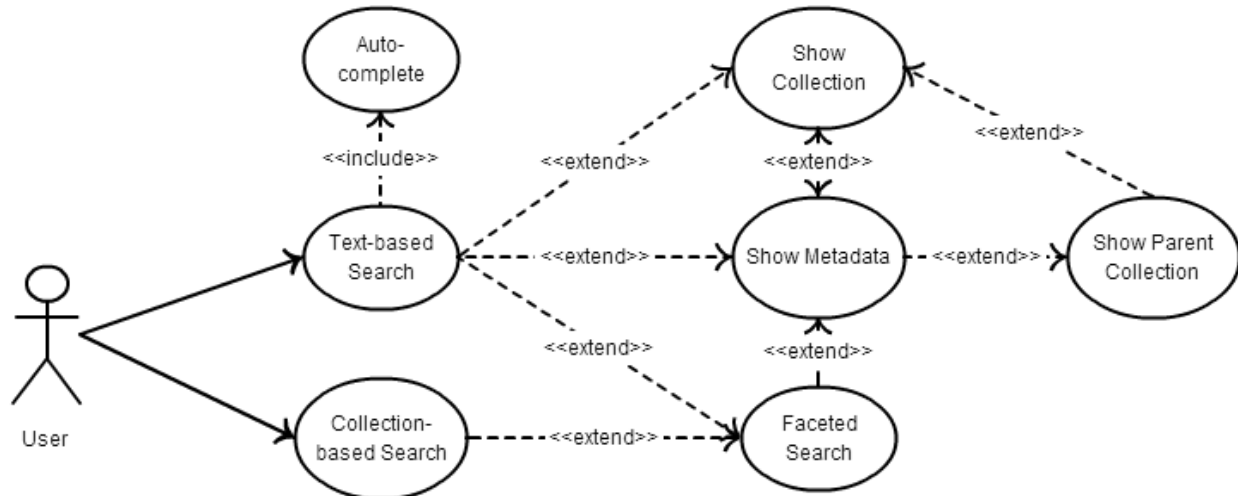


Figure 3.33: Use case diagram of core functionality of the Search Interface

The Search Interface was created using PHP, JavaScript and JQuery. In order to facilitate the implementation of the outlined functionality a Search Interface API was created. The API consists of a number of methods that are used for querying: Fedora, RISearch and Solr.

3.7.1.1. Search Interface API

The Search Interface consisted of a number of PHP scripts that were accessed with the use of HTTP GET requests. The data is generally returned by the API in JSON to ensure lightweight exchange of the data. Once the API had been created, the Search Interface was set up to interface with the API.

API Methods

Fedora Digital Object Datastream Query

URL Syntax

http://nala.cs.uct.ac.za/zamaniProject_archive/inc/datastream.php?{PID}/datastreams/{DATASTREAM}/content

HTTP Method

GET

HTTP Response

Specific object datastream returned in XML. Empty XML document returned if PID or DATASTREAM not found.

Parameters

Name	Description
{PID}	The persistent identifier of the digital object
{DATASTREAM}	The digital objects datastream to query

Solr Query

URL Syntax

http://nala.cs.uct.ac.za/zamaniProject_archive/inc/solr.php?q={QUERY_STRING}&start=0

HTTP Method

GET

HTTP Response

Response from Solr for the given query returned in JSON.

Parameters

Name	Description
{QUERY_STRING}	Query string for Solr

RI Search Tuple Query

URL Syntax

http://nala.cs.uct.ac.za/zamaniProject_archive/inc/risearch.php?type=tuples&lang=itql&format=json&query={QUERY_STRING}

HTTP Method

GET

HTTP Response

Response from RI Search for the given tuple query, returned in JSON.

Parameters

Name	Description
{QUERY_STRING}	ITQL based tuple query

Solr Suggest Query

URL Syntax

http://nala.cs.uct.ac.za/zamaniProject_archive/inc/suggest.php?q={QUERY_STRING}

HTTP Method

GET

HTTP Response

Response from Solr Suggest for the given query string, returned in JSON.

Parameters

Name	Description
{QUERY_STRING}	Query string for Solr Suggest

Get Additional Metadata

URL Syntax

http://nala.cs.uct.ac.za/zamaniProject_archive/inc/getAdditionalMetadata.php?pid={PID}

HTTP Method

GET

HTTP Response

Response from Get Additional Metadata, consists of file size from SIZE datastream, position data from POSITION datastream and conditionally camera calibration settings from the CALIBRATION datastream for the given PID. The response is returned in HTML that can be directly used by the Show Metadata feature.

Parameters

Name	Description
{PID }	The persistent identifier of the digital object

Get Scaled Image

URL Syntax

`http://nala.cs.uct.ac.za/zamaniProject_archive/inc/getScaledImage.php?w={WIDTH}&h={HEIGHT}&mimeType={MIME_TYPE}&pid={PID}`

HTTP Method

GET

HTTP Response

Response from Get Scaled Image, consists of an image of the specified width and height. There is however a maximum dimension of 500 pixels. Additionally, if the PID does not exist or does not have an image associated with it, an image containing the text “Image Not Available” is returned.

Parameters

Name	Description
{WIDTH}	The width of the image
{HEIGHT}	The height of the image
{MIME_TYPE}	The mime type of the digital object
{PID }	The persistent identifier of the digital object

3.7.1.2. Home Page Design

In order to create an aesthetically appealing Home Page that could easily be integrated with the Zamani Projects current website (<http://www.zamaniproject.org/>), the theme of the Zamani Project website (See Figure 3.34) was incorporated into the design (See Figure 3.35). The Zamani Project colour scheme (consisting of two shades of grey and orange) and banner were used to create a link between the Public Portal of the Zamani Data Archive and the Zamani Project website. The ‘Archive’ navigation button is highlighted on the home page, attracting attention to the purpose at hand. Additionally, the text-based search is designed in accordance to the principle of least astonishment [31]. The input bar for the text-search is large and positioned near the top of the screen. The law of least astonishment is followed by adding a well-known search icon to the search bar and adding placeholder text. The two main search features follow Fitt’s Law [16] as they are easily accessible on the home page and are of a relatively large size. Fitt’s Law states that the duration of time taken to use a given item is a function of the size of the item and distance to it. As the function of the collection-based search using the map could easily be confused, a ‘Search by Collection’ header is added. Bootstrap [6] was used as the HTML, CSS and JavaScript framework to develop the Search Interface. Bootstrap was chosen as it speeds up the development of Web-based applications. Additionally, it is easy to use, highly scalable and results in a highly responsive and aesthetically appealing design.

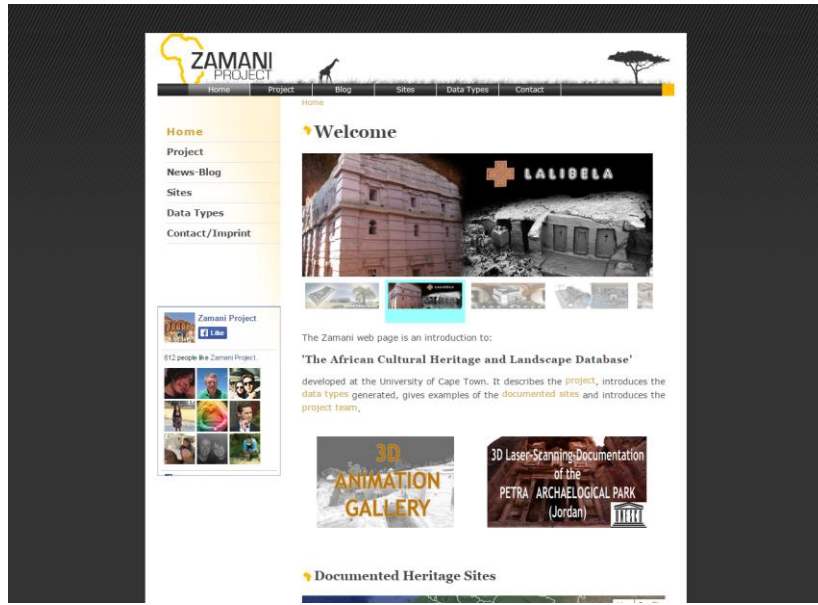


Figure 3.34: Zamani Project website

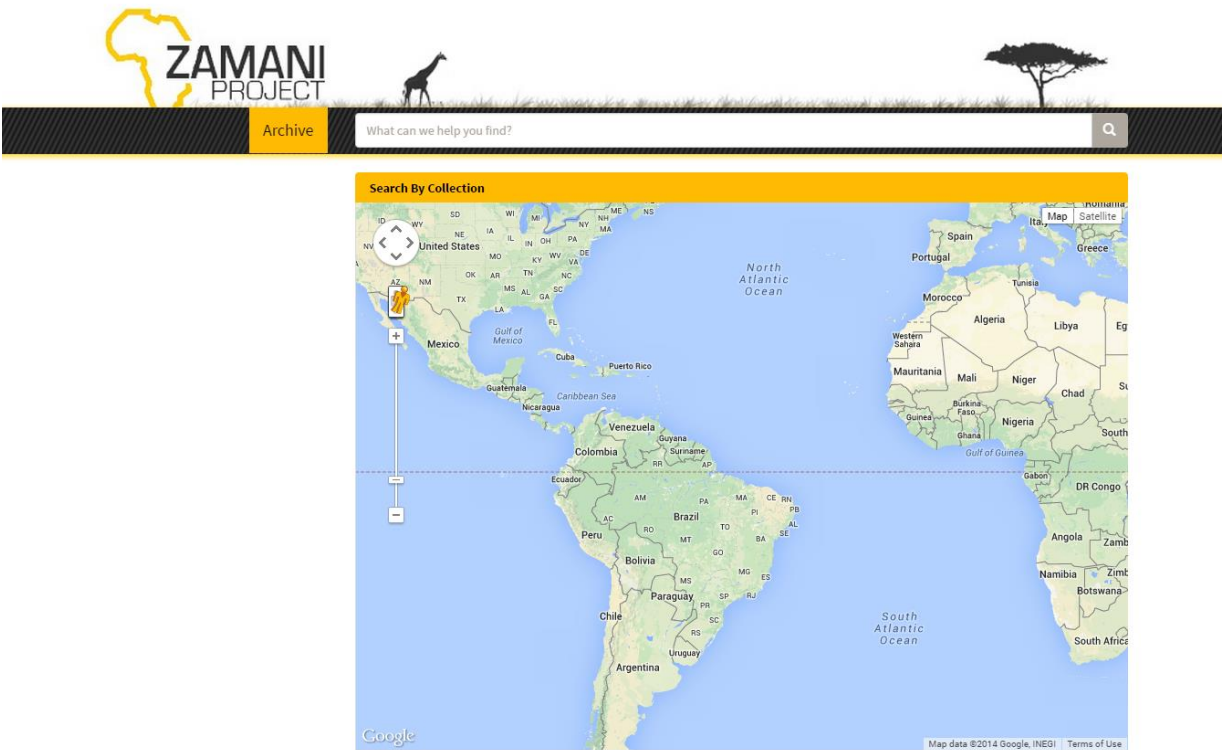


Figure 3.35: Zamani Data Archive - Home Page

When designing and laying out the Home Page, a number of content areas were set up and linked to unique identifiers (See Figure 3.36). This was done to facilitate the ease of integration of the proposed functionality.

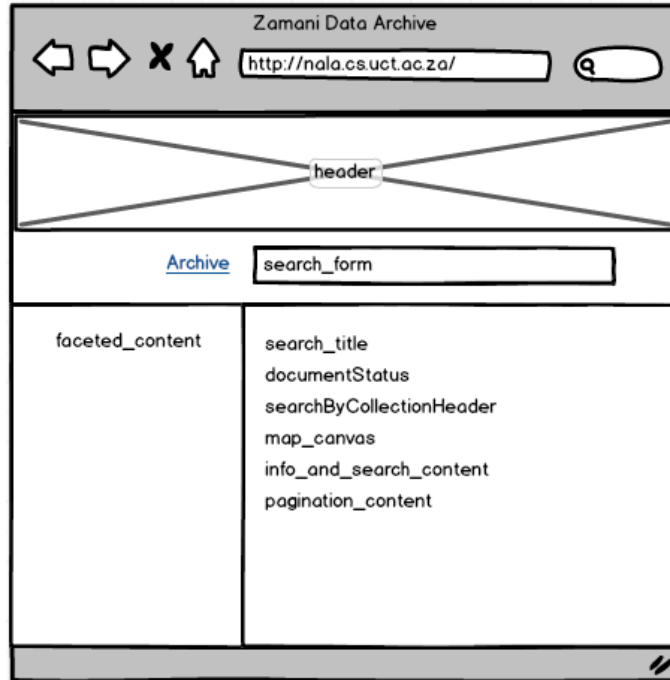


Figure 3.36: Content Area IDs

3.7.1.3. Interfacing with the Search Interface API

Once the Search Interface API and Home Page structure had been created, the interfacing with the Search Interface API commenced. In order to interface with the Search Interface API, JQuery's HTTP GET method was used, this is illustrated in Figure 3.37.

```
$.get( "Search Interface API URL", function( data ) {
    // Parse and use returned data from Server
});
```

Figure 3.37: Interfacing with the Search Interface API

3.7.1.4. Important Functions

A number of JavaScript functions were needed to facilitate the implementation of the proposed Search Interface functionality of the Zamani Data Archive, the notable functions are outlined below:

getQueryVariable(variable)

Returns the given query variable value from the clients URL.

populateAndInsertMap()

This function is used to populate the collection-based map search. A new Google map is created specifying the map_canvas as the content container. A RISearch Tuple Query is made with the use of HTTP GET, the query selects all members of the root (Zamani Project collection). Members of the root collection are country collections. All special characters in the query string are escaped as they can lead to query related syntax errors. An example URL for the RISearch Tuple Query Method is illustrated in Figure 3.38.

```
http://nala.cs.uct.ac.za/zamaniProject_archive/inc/risearch.php?type=tuples&lang=itql&format=json&q
uery=select%20%24member%20from%20%3C%23ri%3E%20where%20%24member%20%3Cfedora-
rels-ext:isMemberOf%3E%20%3Cinfo:fedora/zamaniProjectCollection:1%3E
```

Figure 3.38: URL example for RISearch Tuple Query Method

The results returned are then used to perform an additional HTTP GET request to get the members (site collections) from the returned Zamani collection members (country collections).

Once all of the site collections have been retrieved, HTTP GET requests are made to request the SITE datastream of all of the retrieved collections. The SITE datastream contains the country, site name and position of the site, all of which are needed to produce an informative collection-based map search. For each SITE datastream that is retrieved a new Google map marker is plotted on the created map. Additionally, an on-click listener is added to the marker that makes use of the onMarkerClick method that takes the site collections PID as its parameter. This ensures that collection-based searches can be accomplished on the click of a marker.

[createCollectionMap\(collectionCanvasID, siteColID, latitude, longitude, siteName\)](#)

This method is used to create a single collection-based search map for a site, it acts as the thumbnail image of a site collection.

[escapedPID\(pid\)](#)

Returns an escaped PID from the given pid. Escaped PIDs are needed to ensure that Solr queries do not return syntax related errors.

[getAndAppendSearchResults\(query, startPosition, faceted\)](#)

The startPosition refers to the position at which the documents to be queried should start from. The faceted parameter is set to true when the user has faceted the query. The query is used to make an HTTP GET request to the Solr Query method. The returned documents are iterated through. General search results are displayed as a list of result content areas (one for each document) and are limited to ten results per page. A general search result is considered as an item of a collection and not a collection itself. General search results are displayed with an image preview using the Get Scaled Image method and a number of the Dublin Core Datastream fields, namely: Title, Creator, Date, Type and Source. The Title is a hyperlink and is used to navigate to the Show Metadata view of the item. Site collections are displayed differently to general search results, making use of the createCollectionMap method to display a map in the image preview content area. Additionally, the title of the returned result is a hyperlink and is used to navigate the user to the Show Collection view of the collection. Country collections are also displayed differently to general search results, a horizontal list of the countries' contained sites is displayed. This is illustrated in Appendix C6, however there is only one site collection associated with the country collection. The contained sites are displayed similarly to how a site collection is displayed. Additionally, collections that are returned in search results are displayed at the top of the search results.

This function also handles faceted queries. If a user has faceted the query, the page is re-paginated and getAndAppendSearchResults is called. Additionally, this method handles pagination, with the help of the Simple Pagination JQuery plugin [22] that handles the pagination functionality (See Appendix C3). When a page number is selected, the getAndAppendSearchResults method is called, the startPosition parameter is used to shift the returned documents to the correct page.

Lastly, in order to ensure that the search results are displayed correctly and that the rest of content containers are hidden, the `map_canvas` and `searchByCollectionHeader` are hidden. The `info_and_search_content` and `pagination_content` are shown. This is accomplished with the use of the JQuery `hide` and `show` methods respectively.

`appendFacetsAndPaginatedSearchResults(inputQuery)`

Starts by using the `getAndAppendSearchResults` method with the following parameters: `inputQuery`, `start` being 0 and `facet` being false. Then uses the Solr Query method to do a facet based query as follows:

```
http://nala.cs.uct.ac.za/zamaniProject_archive/inc/solr.php?q="{ QUERY_STRING }"&facet=true&facet.limit=10&facet.field=dc.type&facet.field=dc.format
```

Figure 3.39: Faceted query using the Solr Query method

As illustrated in Figure 3.39, the faceted query is used to facet the given query string by Type and Format. The response is parsed and used to create the contents of the `facet_content` content area. This is done by creating an HTML list for each facet category, the category name is present in the list's header. The list content contains a number of check boxes and the facet filters themselves (See Appendix C5). The JavaScript `onClick` event of the input check boxes is used to house the `applyFilters` method. For the Type check boxes a parameter of false is used and true for the Format check boxes. This ensures that when a Type facet filter is selected the Format facet filters are updated accordingly. The Date range facet is managed with the use of the JQuery `click` function to determine when the date range submit button is pressed. The click event triggers a function that is used to validate that the start and end dates are valid using the following regular expression: `"([1-2][0-9][0-9][0-9])"`. Additionally, it was ensured that the end date is greater than or equal to the start date. Validation for incorrect date ranges is provided below the date range input fields. If the date range is valid it is used to construct a date range facet query that is appended to the query returned from the `getCurrentFilter` method. The newly constructed query is then used along with a `start` of 0 and `facet` of true as the parameters for the `getAndAppendSearchResults` method.

`getSelectedFacets()`

Returns an Array containing a list of the selected facets. The selected facets are calculated based on whether they are checked or not. A checked facet is considered a selected facet.

`applyFilters(isFormatFilter)`

This method makes use of the `getSelectedFacets` method to construct a query string based on the selected facets. The `getAndAppendSearchResults` method is then called with the newly created query string. Additionally if the supplied `isFormatFilter` is false, the format filters are updated with the use of the `updateFormatFilter` method.

`getCurrentFilters`

This method returns a query containing all of the current facet filters using the `getSelectedFacets` method.

`showCollection(collectionPID)`

This method constructs a query that returns all of the children of the given parent collection. The query is used as the parameter for the `appendFacetsAndPaginatedSearchResults` method.

`showChildrenCollection(parentCollectionPID)`

The given `parentCollectionPID` is used to make an HTTP GET request to the `RIsearch Tuple Query Method`. The returned results are then used to construct a list of the children collections of the given parent collection. Each collection in the list includes a thumbnail collection map that is created with the use of the `createCollectionMap` method.

The following content areas are then hidden: `pagination_content`, `map_canvas` and `searchByCollectionHeader`. The constructed list is then inserted into the content area of the `info_and_search_content` with the use of the JQuery `html` method. Lastly, the following content areas are shown: `info_and_search_content`, `search_title`.

showMetadata(objectPID)

The given `objectPID` is used to construct a detailed metadata view of the object. This is done by using an HTTP GET request to use the Solr Query method to get the information from the Dublin Core datastream. It then makes an HTTP GET request to use the Get Additional Metadata method to get the information contained with the rest of the object's datastreams. The information returned from these two methods is then parsed and used to construct the Show Metadata view's HTML content, consisting of an image Preview using the Get Scaled Image method and a table containing all of the objects metadata (See Appendix C8). This is accomplished with the use of the JQuery `html` method that is used to insert the HTML content into the `info_and_search_content` content area. The `info_and_search_content` area is then shown using the JQuery `show` method. The Show Metadata view additionally includes hyperlinks to navigate to the Show Collection and Show Parent Collection views. This ensures that an items collection or parent collection can be backtracked from the Show Metadata view.

\$(document).ready(function())

The JQuery document-ready function contains the logic to manipulate the Home Page. The logic is contained within the document-ready function as a page cannot be safely manipulated until the document is ready.

The Search Interface contains four views: Home Page, Show Parent Collection, Show Collection and Show Metadata. Content areas that are not used in the view are hidden. When the document is ready the `getQueryVariable` method is used to check if any of the views are active. Depending on the active view, the necessary show methods are called, this is outlined for all of the views below.

Home Page

If none of the other views are active, the `searchByCollectionHeader` and `map_canvas` are shown. This view is illustrated in Appendix C1.

Show Collection

If the show collection view is active, the `showCollection` method is called.

This view is illustrated in Appendix C2.

Show Parent Collection

If the show parent collection view is active, the `showChildrenCollections` method is called. This view is illustrated in Appendix C4.

Show Metadata

If the show metadata view is active, the `showMetadata` method is called. This view is illustrated in Appendix C8.

The document-ready function not only handles the views but also houses the `autocomplete` method and `search_form` submit method.

Autocomplete Method

The JQuery Autocomplete widget library [21] was used to have access to this method. The method used the Solr Suggest method as its service URL and `search_form` as its form of input. The result from the Solr Suggest method is parsed and returned to the `autocomplete` method. The auto-complete library handles the display of the available auto-complete options below the input field. In this case the input field was the `search_form`, thus the auto-complete functionality appears below the search bar (See Appendix C7).

Search Form Submit Method

The JQuery submit method is used to track the submit event of the search bar. When the search bar is submitted, and there is a value contained within its input field, the value is used as the query string for the `appendFacetsAndPaginatedSearchResults` method.

Key Up Method

The JQuery key-up method was used to track key events in the input field of the search bar. If the input field is made empty at any stage, the view is reset to the Home Page view.

3.7.2. Evaluation

The evaluation phase for the third iteration was rather short. The Search Interface was demonstrated separately to the project supervisor and Zamani team. This served as a demonstration of the Search Interface's core functionality as well as a way to gain insight into other possible features and interface issues.

The Search Interface illustrated a functional and aesthetically appealing interface that could provide the Zamani team with a way of presenting their large collection of cultural heritage sites to the public.

Two additional features were identified:

Search by Type

It was noticed that there was no mechanism in place to search the whole Archive for one or more specific types of data. This was identified as an important use case that should be present on the Home Page View.

Popup Image Viewer

The image present in the Show Metadata View was identified as being too small. Thus the Show Metadata View needed to include a button or hyperlink that could be used to pop a large view of the image.

3.8. Iteration 4: Final Design

The fourth iteration was the final iteration and served as the last round of design, implementation and evaluation. In this iteration, both the Metadata Management Tool and Search Interface components of the Zamani Data Archive were finalized. This entailed adding a small amount of new functionality to each component as well as a rigorous evaluation process discussed in Chapter 4.

3.8.1. Design and Implementation

The final iteration uses the second and third iterations as the basis for the final round of design and implementation. The primary goal was to refine and extend the Metadata Management Tool and Search Interface with the additional functionality presented in Section 3.5.2 and 3.6.2 respectively.

3.8.1.1. Metadata Management Tool Final Refinements

Automated Title Generation

The ability to automatically generate titles for a metadata set was identified as a feature that could save the Zamani Project team a significant amount of time. A Calculate Titles menu item was added to the Edit menu to house this feature. When the feature is used, the metadata set is traversed and a title is created and set for each metadata item.

The title is generated based on the values contained within the metadata item. More specifically, the metadata item's type and spatial data are used to construct the item's title. The title is constructed as follows:

Type of Coverage Spatial Part within the Coverage Spatial Structure in Coverage Spatial Name, Coverage Spatial Country

An example of a constructed title:

Photogrammetric image of Apedemak temple within the Great Enclosure in Musawwarat es-Sufra, Sudan

Batch Modify

Due to a large number of metadata items containing the same field values, the ability to modify the values of one or more selected metadata items simultaneously was essential. The Batch Modify feature was created to this end. A Batch Modify menu item was added to the Edit menu to house this feature. An additional popup window was created that contained all of the metadata fields that could utilize the feature (See Appendix D2). A key was created to ensure that the function of Batch Modify was not misinterpreted. The key contains definitions for CLEAR and IGNORE. To clear a field, it must be left blank or CLEAR must be typed into the field. Alternatively, all of the field values start off as IGNORE, meaning the selected metadata item field values will not be modified for the field. Additionally, a drop down list is populated containing all of the unique creators contained within the selected metadata items. It follows the same design as the Field Dictionary to ensure familiarity. Thus, creators can be updated, removed and added. If the creators are not modified, no modification is made to the selected metadata items. Furthermore, the Publish status is contained within a radio button group. This ensures that the status may only be modified if the 'change' radio button is selected.

3.8.1.2. Search Interface Final Refinements

Search by Type

The Search by Type feature was added to enable users to browse the whole Archive for specific types of data that they are seeking. This feature was added to the Home Page View, Show Metadata View and the Show Parent Collection View. It is inserted into the faceted_content content area with the use of the JQuery html method (See Appendix C9). It was designed similarly to the faceted-search, 'Search by Type' was used as the list header and the available types are listed with check boxes. The number of items contained within the facet are displayed within a badge. Additionally, a 'Search' button was added so that multiple types could be selected and browsed. This is illustrated in Figure 3.40. A searchByType method was added that constructs a faceted query string based on the selected type filters, using the getSelectedFacets method. The constructed faceted query is added to a query string of "*" which is the Solr syntax to search for all documents. The concatenated query is then used as the parameter in the appendFacetsAndPaginatedSearchResults method. Finally, the searchByType method is added to the onClick event of the 'Search' button.

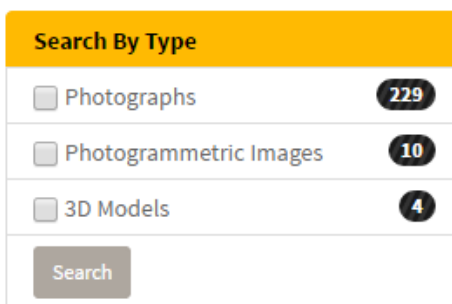


Figure 3.40: Search by Type

Popup Image Viewer

In order to facilitate the Popup Image Viewer a magnifying glass icon and ‘View larger image’ hyperlink was included below the image on the Show Metadata View (See Appendix C10). This demonstrated recognition over recall, as users are likely to identify the function of the magnifying glass [26]. In order to create a popup, the JQuery LeanModal plugin was used. The modal makes use of a defined container and trigger. An image popup container was created and the trigger was set to the identifier of the HTML element containing the hyperlink. Additionally, a new insertImage method was created and added to the onclick event of the same element. The insertImage method makes use of the Get Scaled Image method to insert a larger version of the image into the defined modal container. The onclick event of the element is fired before that of the modal’s trigger, thus when the hyperlink is pressed the modal container is updated and the trigger event then pops up the container on-screen and darkens the surrounding background. The LeanModal also allows for a close button to be specified, but it was decided that a close button would distract from the image. Alternatively, the popup is closed by clicking on the darkened background. The modal container of the image pop-up was designed with a white frame to create added contrast between the pop-up and darkened background (See Appendix C10).

Chapter 4

Evaluation

This section is concerned with determining the applicability of the proposed solution. It describes the various evaluation methods that were carried out in the development of the Zamani Data Archive.

In order to determine the applicability of the system, the system was evaluated on three fronts. Performance testing was performed for the Metadata Management Tool. A Usability Test was performed to test the usability of the system. The system was then evaluated using User Acceptance Testing, with the purpose of demonstrating that the system requirements had been met and that the system could be implemented in a production environment.

4.1. System Testing

System testing was performed throughout the development of the Zamani Data Archive. Three core testing methodologies were used.

4.1.1. Black-box Testing

This method was used to test the system functionality without knowledge of its internal structure and logic. This method proved useful in testing the search interface's API to ensure that correct results were being returned by the API.

4.1.2. White-box Testing

This method was used to test the internal functions of the metadata management tool and search interface. This was fundamental to ensure that components were functioning correctly.

4.1.3. Peer Review

This method was used to ensure that all of the system's core components were correctly integrated. With both developers testing each other's components after integration using a series of test cases, this method proved useful in ensuring minimal integration issues.

4.2. Performance Testing

Performance testing was performed for the Metadata Management Tool as processing a large amount of data in a Java Applet proved to be significantly slow. The processing of data located on a shared drive is regarded as the most expensive process of the Metadata Management Tool.

Tasks that were identified as slow were making use of a recursive directory iterator in order to process data. Due to the processing of data being the source of performance issues, a number of alternative algorithms (See Appendix B1) were considered in an attempt to optimize such tasks. One of the regularly used file directory count methods was examined and compared to these algorithms.

The algorithms were also compared whilst running in two different environments, namely a Java Applet and Java Application. This was done to identify if the cause of the problem was the Java Applet itself.

Additionally, a comparison between using a simple heuristic as opposed to Java's native File class's `isDirectory()` method was evaluated. The simple heuristic consists of a condition that considers digital objects with a "." in their filename as files and the rest as directories (See Appendix B1). This was considered to not be a good solution to the problem as it relies on a defined directory structure. However, the heuristic could be applied to the Zamani dataset as directory names do not contain ".". The Simple heuristic was applied to the Classical Recursion and Directory Stream algorithms.

4.2.1. Algorithm Comparison

This section serves to give readers the necessary background information required to understand the key differences of the compared algorithms (See Appendix B1).

Classical Recursion

Classical recursive directory iteration using Java's standard File Class (`java.io.File`).

Directory Stream

Directory iteration making use of the `DirectoryStream` offered by the Nio File Class (`java.nio.File`).

File Visitor

Directory iteration making use of the `FileVisitor` offered by the Nio File Class (`java.nio.File`).

Get Request

Making use of a GET request to get the returned directory count from the server with the use of an `URLConnection` offered by the Java Net Class (`java.net`).

4.2.2. Methodology

This section describes the techniques and system used to evaluate the performance of the different algorithms.

System Parameters

The experiment was conducted using a Windows 8 machine with an Intel Core i5 1.80 GHz (4 CPUs) processor with 4GB of RAM. The source code was compiled using Eclipse's default settings and run as a Java Application and Java Applet as appropriate.

Procedure

The algorithms were evaluated in terms of the time taken to count the number of files in a directory and its sub directories. The directory to process was a directory located on a shared network drive. All of the algorithms were tested on the same directory containing 1371 files. In order to improve reliability and reduce effects caused by network instability, the algorithms were tested over 1000 iterations, resulting in a mean execution time. Additionally, the PHP script used for the Get Request contained headers to ensure that the results were not cached.

The two headers used were:

```
header("Pragma: no-cache");  
header("Expires: 0");
```

The above procedure was done four times. The algorithms were evaluated once with the heuristic and once without the use of the heuristic, while running in a Java Application and similarly whilst running in a Java Applet.

4.2.3. Results

The results obtained illustrate mean execution times for 1000 iterations:

Non-Heuristic

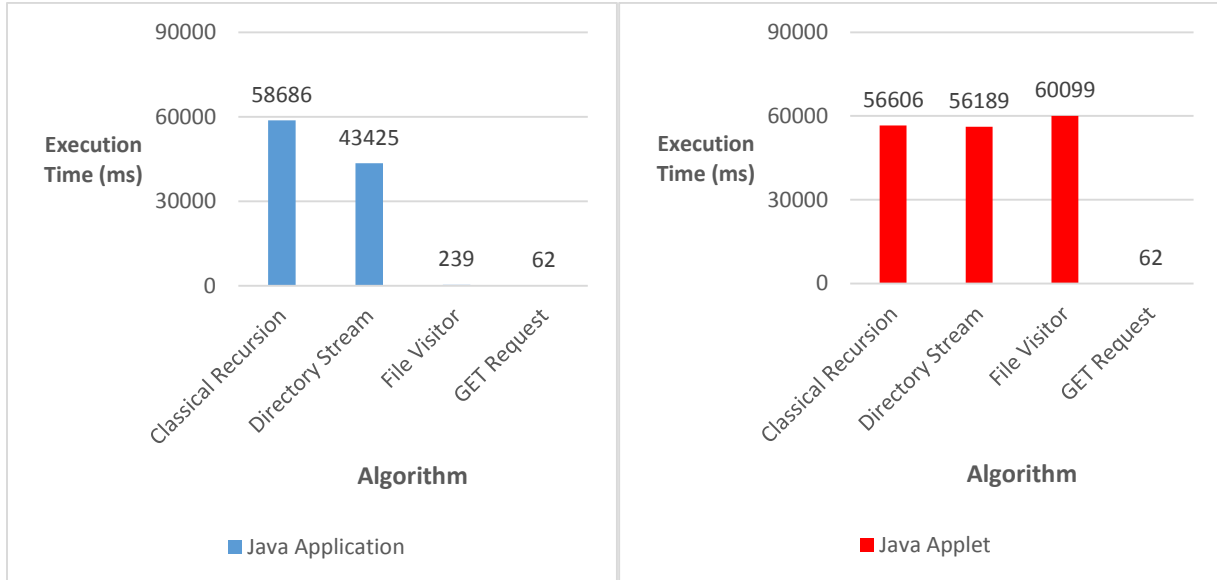


Figure 4.1: Mean Performance Analysis of Algorithms using Non-Heuristic (Java Application & Applet)

Heuristic

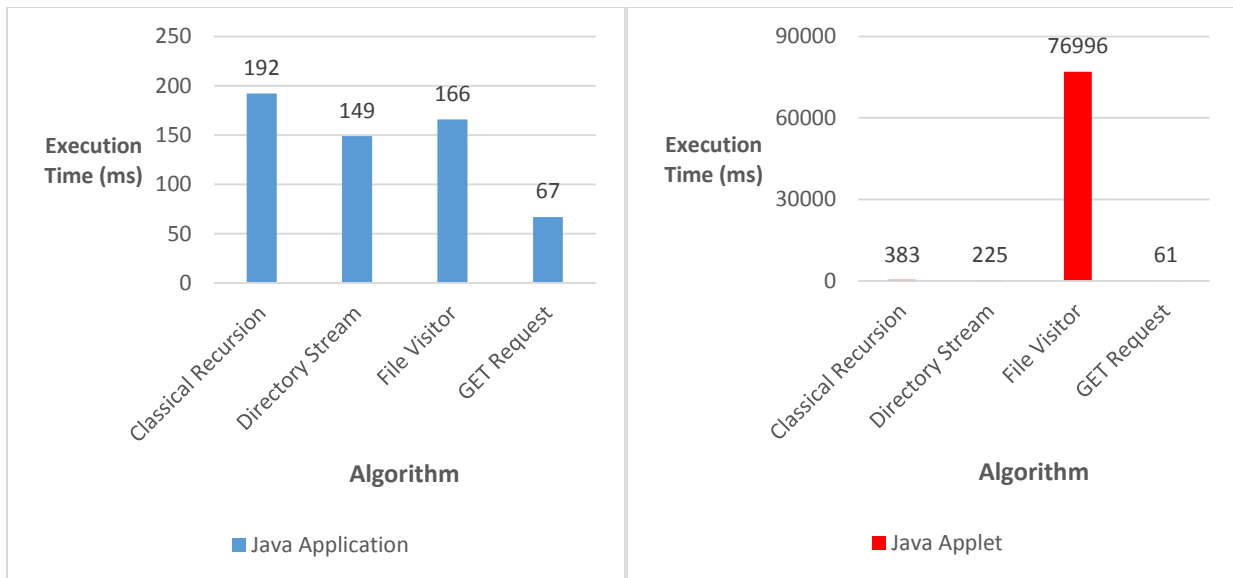


Figure 4.2: Mean Performance Analysis of Algorithms using Heuristic (Java Application & Applet)

$$\frac{53\,424 \text{ files}}{1371 \text{ files}} = 38.97$$

Note: The assumption of a linear increase has been made

$38.97 \times 56\,606$ (execution time of Classical Recursion) = 2205935 ms = 36 minutes

38.97×62 (execution time of GET Request) = 2416 ms = 2.4 seconds

Figure 4.3: Approximate execution times of Classical Recursion and GET Request for a site collection

4.2.4. Discussion of results

The results indicate that using a GET request to facilitate the reading of files is the most efficient and stable method across Java Applications and Applets. The Classical Recursion and Directory Stream show a large reduction in execution time from the non-heuristic to heuristic results for both the Java Application and Applet. The reduction in execution time is attributed to the fact that checking the heuristic condition is far more efficient than using the Java File class's `isDirectory()` method. Additionally, the large reduction is only noticed in the Classical Recursion and Directory Stream algorithms as the heuristic could not be applied to the File Visitor and Get Request algorithms due to their implementation (see Appendix A3). The non-heuristic results (See Figure 4.1) show that the File Visitor algorithm largely out-performs the Classical Recursion and Directory Stream algorithms when running in a Java Application but all perform similarly as slow when running in a Java Applet.

These results were of great importance, as it was identified that the GET Request method provided a significant speedup over the Classical Recursive method that was being used at the time. A number of features in the Metadata Management Tool were making use of a recursive directory iterator, namely: Open Files, Open Directories, Add Files, Add Directories and Associate Files. These features were of utmost importance and were significantly slow, taking approximately one minute to calculate the file count in a directory containing 1371 files. This is illustrated in Figure 4.1. It was of utmost importance that these features could calculate the file count quickly to enable feedback via their respective progress bars. The Associate Files feature would need to be capable of associating a site directory. The Musawwarat es`Sufra site, one of the sites the development team had permission to access for testing purposes, contained 53 424 files. Using the Classical Recursion method to calculate the directory count would take approximately 36 minutes as opposed to that of the GET Request method that would take approximately two and a half seconds. This is illustrated in Figure 4.3, the approximate execution times are calculated using the mean execution times in Figure 4.1. The relevant features were modified to make use of the GET Request before usability testing took place.

4.3. Usability Testing

The usability of a system is a critical component that needs to be considered when designing and evaluating a system. Nielson [25] presents that the benefit of a usable system is that the productivity of its users is greatly increased. A usability test was conducted to ensure that the system provides an appealing and usable interface.

A usability test was conducted in order to assess the quality of the system. The quality of the system is assessed by a user's ability to complete tasks without becoming dissatisfied with the system [5]. This was done to assess how suitable the system is for users to use. The System Usability Scale (SUS) [7] was used to measure user's perceived usability of the system.

4.3.1. User Selection

A sample of 15 participants were used in the study. All participants attended the University of Cape Town and were currently studying at a post graduate level. Additionally, students were chosen from different faculties, namely: Science, Commerce, Engineering and Humanities. This was to ensure diversity amongst participants. Ethical clearance was obtained from both the Science Faculty Research Ethics Committee and Department of Student Affairs in order to perform the usability test on students (See Appendix B3 and B2). The participants chosen for the usability experiment were not the typical end-users that would be expected to interact with the system on a daily basis. This was done to avoid any confirmation bias that could occur due to the participants being stakeholders in the project. However, a sample of researchers without a stake in the project would have been an ideal sample for evaluating the usability of the Search Interface component of the system as they are a feasible representation of end-users. Due to the cost and time involved, a sample of students was used, which is a fair representation of typical end-users of the Search Interface.

The experiment's participants varied in age, ranging from 21 to 24. The level of computer and Web browser experience of participants ranged from basic to advanced. Additionally, the level of understanding ranged from none to advanced in the following terms: Metadata and Search Interface.

4.3.2. Procedure

All participants were required to sign a consent form (See Appendix B4) in order to participate in the study. Participants were then explicitly told that their performance in the tasks was not of importance. This was done to reassure participants that it was the system that was being tested and not them. Participants were then given a Zamani Data Archive Usability Test (See Appendix B5) to complete. The usability test consists of five sections, the first section is used to get background information about participants. The other four sections are split into pairs of tasks and questions related to the Metadata Management Tool and Search Interface of the Zamani Data Archive. After the set of tasks were complete, participants answered questions relating to the tasks and completed a System Usability Scale (SUS) questionnaire. Questions related to task completion were used to gauge which tasks participants had trouble with.

The room that the study took place in was set up prior to the actual experiment so that participants were not given any clues relating to the experiment design. Participants were taken to an isolated location in the University of Cape Town honours computer laboratory. All of the participants took part in the experiment during the afternoon and made use of the same laptop for the purpose of the experiment.

4.3.3. Questionnaire

Questionnaire was the primary method used to evaluate the usability of the system. The System Usability Scale (SUS) [7] was chosen as the questionnaire of choice to evaluate usability as it is widely used as well as an accepted method to measure system usability. SUS is a ten item Likert scale. The System Usability Scale was designed to allow researchers to quickly and easily assess the usability of a system. SUS is flexible by nature as the survey is technology-agnostic, thus it is applicable to a wide range of interface technologies from novel hardware platforms to traditional computer interfaces and websites. Additionally, the survey is relatively quick and easy to use for both the study participant and researcher [3].

4.3.4. Results

The results obtained from the System Usability Scale (SUS) are presented graphically in Figure 4.4 and Figure 4.5. The mean SUS scores are presented in Figure 4.3 and reliability analysis of the SUS scores in Figure 4.6.

Metadata Management Tool	Search Interface
73.12	83.57

Figure 4.3: Mean SUS scores of the Metadata Management Tool and Search Interface

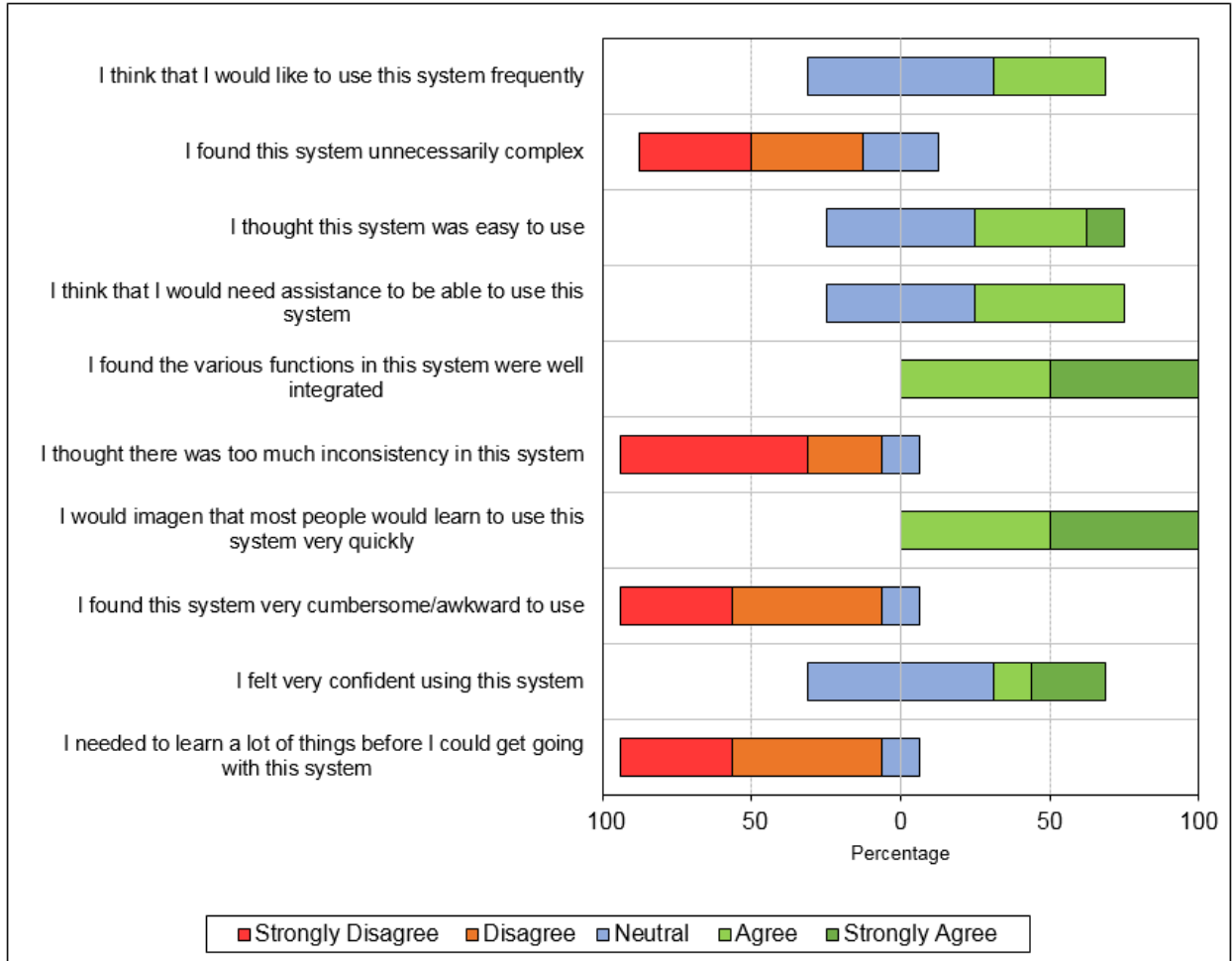


Figure 4.4: Metadata Management Tool: Usability ratings

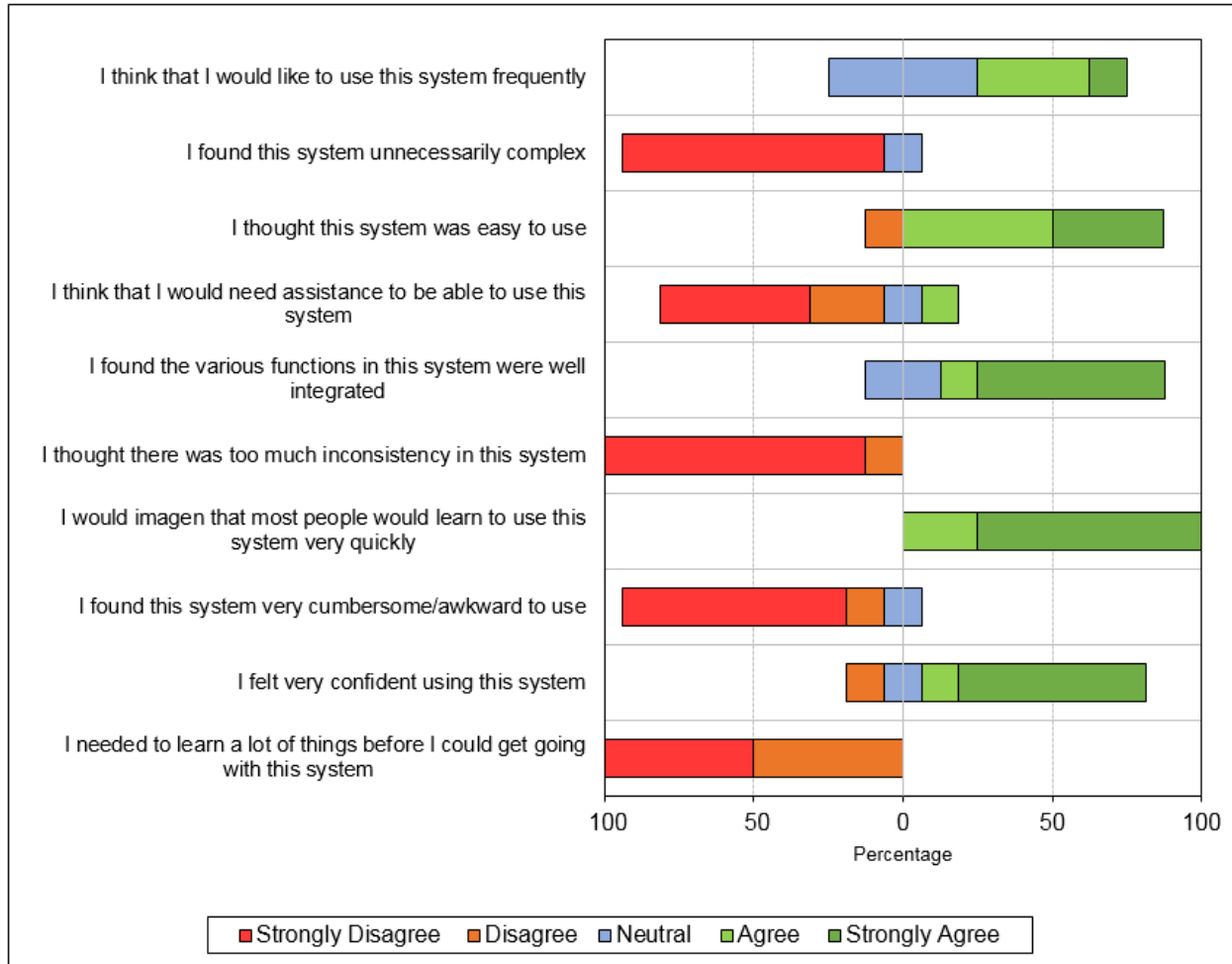


Figure 4.5: Search Interface: Usability ratings

	Metadata Management Tool	Search Interface
Cronbach Alpha	0.85	0.89
Std. Alpha	0.84	0.87
Average R	5.5	6.6

Figure 4.6: Reliability Analysis of the SUS scores of both components

4.3.5. Reliability Analysis

Reliability for internal correlation between the ten items in the System Usability Scale was calculated using Cronbach's Alpha. Reverse coding was used to ensure that the internal correlation was not skewed by reversing the score for all of the negatively phrased statements.

There is a high level of internal consistency in both of the measured components. This can be seen in Figure 4.6. Both the Metadata Management Tool and Search Interface have a Chronbach's alpha of $0.7 \leq \alpha < 0.9$.

4.3.6. Discussion of results

A number of results were produced from the User Experience Evaluation. Users were able to complete the two tasks in an average of 28 minutes, with an interquartile range of 3.5 minutes. Thus, on average, users were able to complete the task within the expected time of 35 minutes. All of the users were able to complete more than 75% of the tasks, with 73% of the users successfully completing all of the tasks. Results were obtained with the use of the survey and task completion time was acquired through observations.

The survey independently quantified data for the Metadata Management Tool and Search interface tasks. This was done to ensure that the components could be evaluated separately. This aimed to test task completion as well as gather user's opinions on the overall usability of each component.

Metadata Management Tool

Users were able to complete all of the Metadata Management Tool tasks, with the exception of 27% of the users who were unable to complete the task that involved batch modifying a set of metadata. This was attributed to batch modifying being one of the more complicated features of the tool.

A breakdown of the various SUS tasks and ratings for the Metadata Management Tool can be seen in Figure 4.4. A subjective label for this component's usability can be obtained with the use of the adjective rating scale [4] and the component's mean SUS score of 73.12 (See Figure 4.3). Thus the Metadata Management Tool's usability is 'Good'.

Search Interface

Users were able to complete all of the Search Interface tasks, with the exception of 20% of the users who were unable to complete the task that involved using the auto-complete feature whilst text-based searching. This was attributed to users not noticing the auto-complete functionality due to reading only half of the presented task.

A breakdown of the various SUS tasks and ratings for the Search Interface can be seen in Figure 4.5. A subjective label for this component's usability can be obtained with the use of the adjective rating scale [4] and the component's mean SUS score of 83.57 (See Figure 4.3). Thus the Search Interface's usability is 'Excellent'.

4.4. User Acceptance Testing

User Acceptance Testing was performed with the Zamani Project team in the Geomatics Department at the University of Cape Town. Three of the team members were able to participate, namely: Roshan Bhurtha, Ralph Schroeder and Stephen Wessels.

The purpose of this evaluation was to demonstrate that all of the project requirements had been met and that the system could potentially be implemented in a live environment. Additionally, it served as an opportunity to guide and train the team to use the new system.

4.4.1. Procedure

In order to demonstrate that the project requirements had been met, the Zamani team members present at the evaluation were given a User Acceptance Test (See Appendix B6). The User Acceptance Test contains: System scope, requirement based test case criteria and test results.

The list of requirement based test case criteria (See Section 4.4.2) was compiled using the in-scope functionality for the Metadata Management Tool and Search Interface respectively. Both components were demonstrated to the members present using the requirement-based test case criteria. This ensured that all of the functionality that was in-scope was demonstrated to the team and that the team could familiarize themselves with the system. After both components had been demonstrated, the team was asked to assess if the requirements for both components of the project had been met by completing the Test Results section. The completed Test Results section is illustrated in Section 4.4.3. This involved the team assigning a “pass” or “fail” to the Metadata Management Tool and Search Interface components.

4.4.2. Requirement-based Test case criteria

ID	Criteria
1	Metadata Management Tool (Meta-fy)
1.1	Open Meta-fy in the browser from the Web-based Backend
1.2	Automate fields using the Automate Tab
1.3	View the Console Tab after executing actions
1.4	Select a metadata item to view its preview image and use the Toolbar to manipulate the image as desired
1.5	Sort metadata by: Identity, File, Status, Tag, and Contains.
1.6	Demonstrate Persistence by saving and loading a .metafy file
1.7	Open files & open directories (clears previously open files)
1.8	Add files & add directories (adds to previously open/added files)
1.9	Import coordinate triplets from a text file
1.10	Import camera calibration settings from a text file
1.11	Remove already added camera calibration settings
1.12	Calculate metadata titles for all metadata present in the Metadata Table
1.13	Associate new files from an Aluka transformed .metafy file
1.14	Validate metadata after associating new files
1.15	Modify a batch of metadata
1.16	Transform an Aluka generated .xml file into a .metafy file
1.17	Export a set of metadata
2	Search Interface
2.1	Search by Type
2.2	Search by Collection by selecting a marker on the map
2.3	Facet search results
2.4	Text-based search
2.5	Text-based search making using of auto-complete functionality
2.6	View a large image preview in the ‘Show Metadata’ view
2.7	View Metadata for an item

4.4.3. Results

IDs	Test Cases	Pass/Fail	Tested By	Date Tested
1.1 - 1.17	Metadata Management Tool (Meta-fy)	Pass	Zamani team	20/10/2014
2.1 - 2.7	Search Interface	Pass	Zamani team	20/10/2014

Figure 4.7: User Acceptance Test results

4.4.4. Discussion of results

The system was evaluated and determined to be a success (See Figure 4.7). All of the functionality that was specified as in-scope and was a requirement of the Zamani team was successfully implemented and approved by the team. The team responded positively to the system and are interested in the system being implemented in a production environment in the future.

Chapter 5

Conclusion and Future Work

5.1. Conclusion

The system proposed was the Zamani Data Archive, an archive created for the Zamani Project for its large collection of spatial data. The system consisted of an Archive, Public Portal, Metadata Management Tool and Search Interface.

The Metadata Management Tool component of the Zamani Data Archive was developed to assist the Zamani Project team with the difficulty involved in creating metadata for a site containing large volumes of data. Additionally, the tool was required to effectively manage the interlinked nature of site datasets.

A Search Interface was developed that facilitated the discovery process of the Zamani dataset. The Search Interface provided navigation through the dataset with the use of: text-based search, site collection search via a Google Map overlay, type-based search and faceted search. Additionally, collections and their parent collections can be backtracked to from the metadata view of a file.

This development project was concluded with the successful design and implementation of the Zamani Data Archive. The Metadata Management Tool and Search Interface components were designed and implemented in three iterations, this is explained in depth in Chapter 3. Lastly, the system was evaluated in Chapter 4 in terms of its performance, usability and its acceptance by the Zamani Project team. The following results were obtained from the evaluation:

1. The Metadata Management Tool was successfully optimized when performance issues were experienced.
2. The results obtained from the System Usability Scale using a sample size of 15 users, revealed based on an adjective scale that the Metadata Management Tools usability was ‘Good’ and the Search Interfaces ‘Excellent’.
3. The User Acceptance Test results revealed that the Zamani Project team considered the system to be a success.

The Zamani Project team responded positively to the system and are interested in the system being implemented in a production environment. The team indicated their enthusiasm to continue working on the Zamani Data archive in the future. Additionally, the team brought up a number of future improvements that are mentioned in the future work section.

5.2. Future Work

This section outlines avenues for future work in the Metadata Management Tool and Search Interface. It incorporates suggestions from the evaluation and other possible features that were not implemented due to time constraints.

5.2.1. Metadata Management Tool

Autosave

The automatic saving of a previously saved metadata set. Autosaves would occur on a predefined interval and before any of the complex editing features. This would remove the hassle of having to continually save a metadata set that is being edited. Additionally, it would lessen the possibility of losing information.

Log File

The feedback that is provided via the console should also be written to a log file. This would act as an event log that could be used to understand system activity and diagnose errors.

Directory Explorer

An additional sort method that can be used to edit a subset of a metadata set. The subsets would consist of the different directories within the metadata set. When a directory is selected from the directory explorer the metadata table would be re-populated with the directories metadata items.

5.2.2. Search Interface

Solr Spatial Search

The addition of an additional navigation method, namely Solr Spatial Search. This would enable the Zamani Project's dataset to be filtered by spatial concepts such as: bounding box, shape and distance.

Zamani Website Integration

Integrating the Public Portal of the Zamani Data Archive and the Zamani Project website. This would ensure that the Zamani Data Archive could be accessed from the Zamani Website and vice versa.

References

- [1] "Aluka". [Online]. Available: <http://www.aluka.org/>. [Accessed: 16-October-2014].
- [2] "Apache - Tika". [Online]. Available: <http://tika.apache.org/>. [Accessed: 24-October-2014].
- [3] Bangor, A., Kortum, P.T., & Miller, J.T. An empirical evaluation of the system usability scale. *Intl. Journal of Human–Computer Interaction*, 24(6), 2008, 574-594.
- [4] Bangor, A., Kortum, P. T., & Miller, J.T. Determining what individual SUS scores mean: Adding an adjective rating scale. *Journal of usability studies*, 4(3), 2009, 114-123.
- [5] Bevan, N. Measuring usability as quality of use. *Software Quality Journal*, 4(2), 1995, 115-130.
- [6] "Bootstrap". [Online]. <http://getbootstrap.com/>. [Accessed: 24-October-2014].
- [7] Brooke, J. SUS-A quick and dirty usability scale. *Usability evaluation in industry*, 1996, 189-194.
- [8] Bucanek, James. *Learn Objective-C for Java Developers*, 2009, 353-402.
- [9] "Duraspace – Fedora – Resource Index Search". [Online]. Available: <https://wiki.duraspace.org/display/FEDORA37/Resource+Index+Search>. [Accessed: 24-October-2014].
- [10] "Duraspace – Fedora – Resource Index". [Online]. Available: <https://wiki.duraspace.org/display/FEDORA37/Resource+Index>. [Accessed: 24-October-2014].
- [11] "Duraspace – Fedora – FOXML". [Online]. Available: <https://wiki.duraspace.org/pages/viewpage.action?pageId=34664480>. [Accessed: 24-October-2014].
- [12] "Duraspace – Fedora – Generic Search Service". [Online]. Available: <https://wiki.duraspace.org/display/FCSVCS/Generic+Search+Service+2.7>. [Accessed: 24-October-2014].
- [13] "Duraspace – Fedora – Digital Object Model". [Online]. Available: <https://wiki.duraspace.org/display/FEDORA37/Fedora+Digital+Object+Model>. [Accessed: 24-October-2014].
- [14] "Duraspace – Fedora – Digital Object Relationships". [Online]. Available: <https://wiki.duraspace.org/display/FEDORA37/Digital+Object+Relationships>. [Accessed: 24-October-2014].
- [15] "Duraspace – Fedora – Getting Started with Fedora". [Online]. Available: <https://wiki.duraspace.org/display/FEDORA37/Getting+Started+with+Fedora>. [Accessed: 24-October-2014].
- [16] Fitts, P. M. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 1954, 47(6), 381.
- [17] Flanagan, D. *JavaScript: The definitive guide: Activate your web pages*. O'Reilly Media, Inc. 2011.
- [18] Garrett, J. J. *Ajax: A new approach to web applications*, 2005.
- [19] "Google Map API ". [Online]. Available: <https://developers.google.com/maps/documentation/javascript/reference>. [Accessed: 24-October-2014].
- [20] "jQuery ". [Online]. Available: <http://jquery.com/>. [Accessed: 24-October-2014].
- [21] "jQuery – Auto-complete". [Online]. <http://jqueryui.com/autocomplete/>. [Accessed: 24-October-2014].

- [22] "jQuery – Simple Pagination Plugin". [Online]. <http://plugins.jquery.com/simple-pagination/>. [Accessed: 24-October-2014].
- [23] Kindler, E., Krivy, I. Object-Oriented Simulation of systems with sophisticated control. *International Journal of General Systems*, 2011, 313–343.
- [24] "Nginx". [Online]. Available: http://nginx.org/en/#tested_os_and_platforms. [Accessed: 24-October-2014].
- [25] Nielsen, J. Usability 101: Introduction to usability. *Jakob Nielsen's Alertbox*, August, 25, 2003.
- [26] Norman, D. A. Design principles for human-computer interfaces. *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 1983, 1-10.
- [27] Press, N. I. S. O. Understanding metadata. *National Information Standards* 20, 2004.
- [28] Rüter, H., Chazan, M., Schroeder, R., Neeser, R., Held, C., Walker, S. J., Matmon, A. & Horwitz, L. K. Laser scanning for conservation and research of African cultural heritage sites: the case study of Wonderwerk Cave, South Africa. *Journal of Archaeological Science*, 36(9), 2009, 1847-1856.
- [29] Rüter, H., Held, C., Bhurtha, R., Schroeder, R & Wessels, S. From point cloud to textured model, the zamani laser scanning pipeline in heritage documentation. *South African Journal of Geomatics*, 1(1), 2012, 44-59.
- [30] "Solr ". [Online]. Available: <http://lucene.apache.org/solr/>. [Accessed: 24-October-2014].
- [31] Thimbleby, H. User interface design. *ACM*, 1990.
- [32] "Zamani Project". [Online]. Available: <http://www.zamaniproject.org>. [Accessed: 16-October-2014].
- [33] "Zamani Project - Data Types". [Online]. Available: <http://www.zamaniproject.org/index.php/data.html>. [Accessed: 16-October-2014].
- [34] "Zamani Project – About". [Online]. Available: <http://www.zamaniproject.org/index.php/project.html>. [Accessed: 16-October-2014].

Appendix A

A1: Embedding a Java applet: Avoid user file system security constraints

Manifest File Creation

1. Use Eclipse to create a manifest file
2. Put it in the project folder
3. Ensure the contents look like:

```
Manifest-Version: 1.0
Created-By: Fat Jar Eclipse Plug-In
Permissions: all-permissions
Application-Name: meta-applet
```

JAR File Creation

1. Download and install 'Fat Jar' Eclipse Plugin
2. Once installed, right click on the eclipse project and select Build Fat Jar to create the JAR file
 - a. When creating the JAR use the Manifest file which was created
 - i. Check the 'Select Manifest File' Option
 - ii. Browse for the Manifest file
 - iii. Uncheck 'merge individual-sections of all MANIFEST.MF files'
 - iv. Press Finish

Self-signed JAR Creation

1. Open command prompt as administrator.
2. Navigate to the Java JDK bin (e.g C:\Program Files (x86)\Java\jdk1.7.0_55\bin)
3. Ensure that you have a Java SDK keytool and jarsigner in your path.
4. Create a key in a new keystore as follows:

```
Keytool -genkey -keystore newKeystore -alias newAlias
```

You will be prompted for information for the creation of the new key. This information includes: i) password; ii) first and last name; iii) name of organizational unit; iv) name of organization; v) city or locality; vi) state or province; vii) two-letter country code,

Once you have successfully entered all of the above information, you will be prompted to confirm if the information is correct.

Enter yes, if the information is correct.

You will be prompted to enter a key password for the new alias.

Once complete, a newKeystore file will be created on the disk.

5. Create a self-signed certificate as follows:

```
Keytool -selfcert -alias newAlias -keystore newKeystore
```

You will be prompted for the keystore password. Certificate generation might take some time, up to a few minutes.

6. Check to see that the self-signed certificate was created. List the contents of the keystore using:

```
Keytool -list -keystore newKeystore
```

The outputted list should look similar to:

```
Keystore type: jks
Keystore provider: SUN

Your keystore contains 1 entry:
newAlias, Oct 21, 2014, PrivateKeyEntry,
Certificate fingerprint (SHA1): C6:74:59:00:00:06:C1:E5:04:C6:C9:A9
```

7. Sign the JAR file using the created certificate as follows:

```
Jarsigner -keystore newKeystore unsignedJar.jar newAlias
```

Note: If you have multiple JARS, these steps will have to be repeated for all of the JAR files. Alternatively a plugin for Eclipse, namely Fat Jar can be used to create one JAR file of a project that contains JAR file dependencies and makes use of one central manifest file.

Embedding a self-signed JAR containing a Java applet in an HTML Document

Note: The <object> tag is used to define an embedded object, this approach makes the assumption that the HTML Document and self-signed JAR will be located in the same directory

1. Add the following object tag and its contents to an HTML Document.

```
<object classid="java:zamani.metafy.core.Main.class"
        type="application/x-java-applet;version=1.7"
        archive="Meta-fy.jar"
        height="97%" width="100%" codebase="." >
  <param name="code" value="zamani.metafy.core.Main.class" />
  <param name="persistState" value="false" />
  <param name="cache_option" value="no"/>
  <param name="codebase_lookup" value="false"/>
  <param name="java_version" value="1.7*">
  <param name="java_arguments" value="-Xmx256m">
  <param name="classloader_cache" value="false">
  <param name="permissions" value="all-permissions" />
</object>
```

2. Modify the classid to “java:” and the name of the class that contains the Java applets init() method (E.g “java:zamani.metafy.core.Main.class”)
3. Modify the types version if you are using a different version of java
4. Modify the archive to the name of the self-signed JAR file
5. Modify the code param value to the name of the class that contains the applets init() method.
6. Modify the java_version param value to the java version that was used.

Configure Java

1. Search for ‘Configure Java’
2. Navigate to the Security tab
3. Select Edit Site List...
4. Add the URL of the HTML imbedded Java applet.
5. Press Ok to add the site to your Exception Site List (It will be allowed to run after the appropriate security prompts).

A2: Basic FOXML file – demonstrates Dublin Core data stream

```
▼<foxml:digitalObject xmlns:foxml="info:fedora/fedora-system:def/foxml#"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" PID="uctnew:0000004" VERSION="1.1"
xsi:schemaLocation="info:fedora/fedora-system:def/foxml# http://www.fedora.info/definitions/1/0/foxml1-
1.xsd">
  ▼<foxml:objectProperties>
    <foxml:property NAME="info:fedora/fedora-system:def/model#state" VALUE="Active"/>
    <foxml:property NAME="info:fedora/fedora-system:def/model#label" VALUE=""/>
  </foxml:objectProperties>
  ▼<foxml:datastream CONTROL_GROUP="X" ID="DC" STATE="A">
    ▼<foxml:datastreamVersion FORMAT_URI="http://www.openarchives.org/OAI/2.0/oai_dc/" ID="DC.1"
    LABEL="Dublin Core Record for this object" MIMETYPE="text/xml">
      ▼<foxml:xmlContent>
        ▼<oai_dc:dc xmlns:dc="http://purl.org/dc/elements/1.1/"
        xmlns:oai_dc="http://www.openarchives.org/OAI/2.0/oai_dc/">
          <dc:description/>
          <dc:identifier>uctnew:0000004</dc:identifier>
          <dc:title/>
          <dc:coverage>Sudan</dc:coverage>
          <dc:coverage>Musawwarat es-Sufra</dc:coverage>
          <dc:coverage>Great Enclosure</dc:coverage>
          <dc:coverage/>
          <dc:coverage/>
          <dc:coverage/>
          <dc:creator>Ruther, Heinz</dc:creator>
          <dc:creator>Held, Christoph</dc:creator>
          <dc:creator>Schroeder, Ralph</dc:creator>
          <dc:creator>Bhurtha, Roshan</dc:creator>
          <dc:creator>Wessels, Stephen</dc:creator>
          <dc:date>2009-10-07</dc:date>
          <dc:format>model/mesh</dc:format>
          <dc:source>University of Cape Town, Geomatics Department</dc:source>
          <dc:type>3D Models</dc:type>
        </oai_dc:dc>
      </foxml:xmlContent>
    </foxml:datastreamVersion>
  </foxml:datastream>
</foxml:digitalObject>
```

Appendix B

B1: Performance Test Algorithms

Nio – Directory Stream

```
static int fileCountNIODirectoryStream = 0;
private static void getFilesCountNIODirectoryStream(Path path) throws IOException {
    DirectoryStream<Path> stream = Files.newDirectoryStream(path);
    for (Path entry : stream) {
        // Non Heuristic Method
        if (entry.getFileName().toString().indexOf('.') != -1){
            fileCountNIODirectoryStream++;
        }
        else{
            getFilesCountNIODirectoryStream(entry);
        }

        // Heuristic Method
        /*
        if (Files.isDirectory(entry)) {
            getFilesCountNIODirectoryStream(entry);
        }
        else{
            fileCountNIODirectoryStream++;
        }*/
    }
}
```

Classical Method – Recursion

```
static int fileCountClassicalMethod = 0;
private static void getFilesCountClassicalMethod(File dir) throws IOException {
    File[] listOfFiles = dir.listFiles();
    for (File file : listOfFiles) {
        // Non Heuristic Method
        if (file.getName().indexOf('.') != -1){
            fileCountClassicalMethod++;
        }
        else{
            getFilesCountClassicalMethod(file);
        }
        // Heuristic Method
        /*
        if (file.isDirectory()) {
            getFilesCountClassicalMethod(file);
        } else {
            fileCountClassicalMethod++;
        }
        */
    }
}
```

NIO - File Visitor

```
private static int getFilesCountNIOFileVisitor(File dir) throws IOException {  
    FileVisitorCounter visitor = new FileVisitorCounter();  
    Files.walkFileTree(dir.toPath(), visitor);  
    return visitor.count;  
}
```

File Visitor Class

```
class FileVisitorCounter implements FileVisitor<Path>{  
    public int count;  
    FileVisitResult CONTINUE = FileVisitResult.CONTINUE;  
  
    public FileVisitorCounter(){  
        count = 0;  
    }  
  
    @Override  
    public FileVisitResult postVisitDirectory(Path arg0, IOException arg1) {  
        return CONTINUE;  
    }  
  
    @Override  
    public FileVisitResult preVisitDirectory(Path arg0, BasicFileAttributes arg1) {  
        return CONTINUE;  
    }  
  
    @Override  
    public FileVisitResult visitFile(Path arg0, BasicFileAttributes arg1) {  
        count++;  
        return CONTINUE;  
    }  
  
    @Override  
    public FileVisitResult visitFileFailed(Path arg0, IOException arg1) {  
        return CONTINUE;  
    }  
}
```

HTTP GET request

```
public static int getFilesCountHTTPGET(String url) throws Exception {

    URL obj = new URL(url);
    HttpURLConnection con = (HttpURLConnection) obj.openConnection();

    // optional default is GET
    con.setRequestMethod("GET");

    //add request header
    con.setRequestProperty("User-Agent", "Mozilla/5.0");

    int responseCode = con.getResponseCode();

    if (responseCode==200){
        BufferedReader in = new BufferedReader( new InputStreamReader(con.getInputStream()));
        String inputLine;
        StringBuffer response = new StringBuffer();

        while ((inputLine = in.readLine()) != null) {
            response.append(inputLine);
        }
        in.close();

        if (response.toString().equals("")){
            return -1;
        }

        return Integer.parseInt(response.toString());
    }
    else{
        return -1;
    }
}
```

Server Side script used by GET request

```
// Call with: script.php?dir=PathToCount
<?PHP

// Ensure results aren't cached
header("Pragma: no-cache");
header("Expires: 0");

if (isset($_GET['dir'])){
    // Execute find command via shell
    // find path -type f -ls used to get all filepaths in a given directory recursively
    // wc -l used to count lines
    echo $pp = shell_exec("find ".$_GET['dir']." -type f -ls | wc -l");
}
?>
```

B2: Science Faculty Ethics Clearance

Faculty of Science
University of Cape Town
RONDEBOSCH 7701
South Africa

E-mail: richard.hill@uct.ac.za
Telephone: + 27 21 650 2786
Fax: + 27 21 650 3456



22 October 2014

Mr Michael Ferguson & Mr Jay Benson
Department of Computer Science

ZAMANI DATA ARCHIVE

Dear Mr Ferguson & Mr Benson

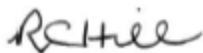
I am pleased to inform you that the Faculty of Science Research Ethics Committee has approved the above-named application for research ethics clearance, subject to the conditions listed below. You are required to:

- implement the measures described in your application to ensure that the process of your research is ethically sound; and
- uphold ethical principles throughout all stages of the research, responding appropriately to unanticipated issues: please contact me if you need advice on ethical issues that arise.

Your approval code is: FSREC 080– 2014

I wish you success in your research.


Yours sincerely



Dr Richard Hill
Chair: Faculty of Science Research Ethics Committee

Cc: A/Prof Hussein Suleman, Supervisor

B3: Department of Student Affairs Ethics Clearance

	RESEARCH ACCESS TO STUDENTS	DSA 100
---	--	----------------

NOTES

- This form must be FULLY completed by the applicant/s who want to access UCT students for the purpose of research or surveys.
- Return the fully completed (a) DSA 100 application form by email, in the same word format, together with your: (b) research proposal inclusive of your survey, (c) copy of your ethics approval letter / proof (d) informed consent letter to: Moonira.Khan@uct.ac.za. Your application will be attended to by the Executive Director, Department of Student Affairs (DSA), UCT.
- The turnaround time for a reply is approximately 10 working days.
- NB: It is the responsibility of the researcher/s to apply for and to obtain ethics approval and to comply with amendments that may be requested; as well as to obtain approval to access UCT staff and/or UCT students, from the following, respectively:
 - Ethics: Chairperson, Faculty Research Ethics Committee' (FREC) for ethics approval, (b) Staff access: Executive Director: HR for approval to access UCT staff, and (c) Student access: Executive Director: Student Affairs for approval to access UCT students.
- Note: UCT Senate Research Protocols requires compliance to the above, even if prior approval has been obtained from any other institution/agency. UCT's research protocol requirements applies to all persons, institutions and agencies from UCT and external to UCT who want to conduct research for academic, marketing or service related reasons at UCT.

SECTION A: RESEARCH APPLICANT/S DETAILS

Position	Staff / Student No	Title and Name	Contact Details (Email / Cell / land line)
A.1 Student Number	FRGMIC005 BNSJAY001	Mr Michael Ferguson Mr Jay Benson	Email: frgmic005@myuct.ac.za / (072) 720 4871 Email: bnsjay001@myuct.ac.za / (073) 186 7141
A.2 Academic / PASS Staff No.			
A.3 Visitor/ Researcher ID No.			
A.4 University at which a student or employee	University of Cape Town	Address if <u>not</u> UCT:	
A.5 Faculty/ Department/School	Department of Computer Science		
A.6 APPLICANTS DETAILS If different from above	Title and Name	Tel.	Email

SECTION B: RESEARCHER/S SUPERVISOR/S DETAILS

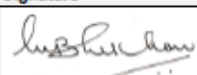
Position	Title and Name	Tel.	Email
B.1 Supervisor	Hussein Suleman	+27 21 650 5106	hussein@cs.uct.ac.za
B.2 Co-Supervisor/s	Maria Keets	+27 21 650 2664	mkeet@cs.uct.ac.za

SECTION C: APPLICANT'S RESEARCH STUDY FIELD AND APPROVAL STATUS

C.1 Degree (if a student)	BSc Computer Science
C.2 Research Project Title	Zamani Data Archive
C.3 Research Proposal	Attached: Yes <input checked="" type="checkbox"/> No <input type="checkbox"/>
C.4 Target population	Students in different years from varying faculties
C.5 Lead Researcher details	If different from applicant:
C.6. Will use research assistant/s	Yes <input type="checkbox"/> No <input checked="" type="checkbox"/> if yes- provide a list of names, contact details and ID no.
C.7 Research Methodology and Informed consent:	Research methodology: Questionnaire and survey Informed consent: will be obtained prior to participation in the study. Confidentiality will be assured by not capturing personal information that can be traced back to a participant.
C.8 Ethics clearance status from UCT's Faculty Ethics Research Committee (FREC)	Approved by the FREC Yes <input checked="" type="checkbox"/> With amendments: Yes / No <input type="checkbox"/> (a) Attach copy of your ethics approval. Attached: Yes (b) State date and reference no. of ethics approval: Date: 24 October 2014 Ref. No.: FSREC 080-2014

SECTION D: APPLICANT/S APPROVAL STATUS FOR ACCESS TO STUDENTS FOR RESEARCH PURPOSE

(To be completed by the ED, DSA or Nominee)

D.1 APPROVAL STATUS	Approved / With Terms / Not	* Conditional approval with terms	Applicant's Ref. No.:
	Approved <input checked="" type="checkbox"/>	(a) Access to students for this research study must only be undertaken <u>after</u> written ethics approval has been obtained. (b) In event any ethics conditions are attached, these must be complied with <u>before</u> access to students.	FRGMIC005/ Mr Michael Ferguson BNSJAY001/ Mr Jay Benson
D.2 APPROVED BY:	Designation	Name	Signature
	Executive Director Department of Student Affairs	Dr Moonira Khan	
			Date
			24 October 2014

B4: Consent Form

Consent to participate in a research study

Dear UCT student,

Invitation to participate in research investigating the usability of the Zamani Archival System.

Introductory Statement

The Zamani Archive is a system built for the Zamani Project (<http://www.zamani-project.org/>) to manage its large collection of geo-spatial data captured at various cultural heritage sites in Africa and the Middle-East. The systems sole purpose is the creation of an archive for the Zamani Project, in order to enable structure, preservation and distribution of the data. The system includes a metadata management tool, namely Meta-fy to ensure that the creation of metadata is easy and automated where possible. The system features a public portal that enables users to search through the archives metadata allowing them to view, request and download data from the archive. Additionally, the public portal includes a permission system whereby users need to register and accept the terms and conditions associated with the data before requesting and downloading data from the archive.

Study Purpose

You are being asked to participate in a research study being conducted by two computer science honours students at the University of Cape Town. The purpose of this study will be to analyse the Zamani Archival System.

Study Procedures

If you decide to participate in this study, you will be asked to complete a number of tasks as specified in the section to follow. After each task you will be asked to complete a short questionnaire. Please note that all information obtained from you will be kept strictly confidential.

Tasks

- Meta-fy related tasks
- Search interface related tasks
- Permission related tasks
- Archival related tasks

Environment & Duration

The study will take place in the honours laboratory in the Computer Science building at the University of Cape Town. The duration of the study is estimated to take 45 minutes but is user-dependent and will vary.

Voluntary Participation

Participation in this study is completely voluntary. You are free to refuse to answer any question. Your decision regarding participation in this study will not have any consequences for you. If you decide to participate, you are free to change your mind and discontinue participation at any time without any consequences.

Questions

Any study-related questions, problems or emergencies should be directed to the following researchers:

Michael Ferguson	frgmic005@myuct.ac.za
Jay Benson	bnsjay001@myuct.ac.za
Prof Hussein Suleman	hussein@cs.uct.ac.za

I have read the above information and am satisfied with my understanding of the study. I hereby voluntarily consent to participation in the research study as described.

Signature of participant

Name of participant

Date

Michael Ferguson and Jay Benson

Researchers

B5: System Usability Test

Zamani Data Archive Usability Test

Questionnaire

There is a total of five sections in this study, three of which consist of questions and the other two tasks.

1. Background Information

This section is essential in obtaining background information about you.

- a. What is your age? _____
- b. Rate your experience in the following:
 - Using a computer (none, very little, basic, advanced, expert)
 - Using a Web browser (none, very little, basic, advanced, expert)
- c. Rate your understanding of the following terms:
 - Digital archive (none, very little, basic, advanced, expert)
 - Permission system (none, very little, basic, advanced, expert)
 - Metadata (none, very little, basic, advanced, expert)
 - Search interface (none, very little, basic, advanced, expert)
- d. What faculty are you enrolled in? _____

2. Search Interface Tasks

The Zamani Archive search interface is open in a tab in the browser.

- 2.1 Search by type for all Photogrammetric Images in the archive
- 2.2 Return to the Archive home page
- 2.3 Search for the Musawwarat es-Sufra Collection using the map.
 - 2.3.1 Refine your search by type to all Photogrammetric Images and Photographs
 - 2.3.2 Refine your search further by format to all image/jpeg formatted images
 - 2.3.3 View the 3rd page of results
 - 2.3.4 Navigate to the record view of the first result on the 3rd page
 - 2.3.5 View a larger version of the image
 - 2.3.6 Exit the view
 - 2.3.7 Go to the parent collection of the currently selected item
 - 2.3.8 View the Musawwarat es-Sufra collection
- 2.4 Return to the Archive home page
- 2.5 Use the text based search to search for “Musawwarat es-Sufra”, make use of the auto-complete functionality

3. Search Interface Questions

- 3.1 Did you successfully complete all of the ‘Search Interface Tasks’?

Please choose one of the following (estimate where needed):

- A) Yes
- B) More than half the tasks successfully completed
- C) More than 75 percent of the tasks successfully completed
- D) Less than half the tasks successfully completed

- 3.2 Were you able to search by type? (YES , NO)
- 3.3 Were you able to refine your search? (YES , NO)
- 3.4 Were you able to view the 3rd page of results? (YES , NO)
- 3.5 Were you able to navigate to the record view of an item? (YES , NO)
- 3.6 Were you able to search by collection using the map? (YES , NO)
- 3.7 Were you able to search using text based search? (YES , NO)
- 3.8 Did you manage to make use of the auto-complete feature? (YES , NO)

Instructions: For each of the following statements, mark one box that best describes your reactions to the system.

		Strongly Disagree				Strongly Agree
1.	I think that I would like to use this system frequently.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2.	I found this system unnecessarily complex.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
3.	I thought this system was easy to use.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
4.	I think that I would need assistance to be able to use this system.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
5.	I found the various functions in this system were well integrated.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
6.	I thought there was too much inconsistency in this system.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
7.	I would imagine that most people would learn to use this system very quickly.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
8.	I found this system very cumbersome/awkward to use.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
9.	I felt very confident using this system.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
10.	I needed to learn a lot of things before I could get going with this system.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

4. Metadata Management Tasks

The Metadata Management Tool (Meta-fy) is open in a tab in the browser.

- 4.1 Start by setting the following Automation Fields
 - 4.1.1 Set the Identifier Prefix to “studynew”
 - 4.1.2 Set the Source to “Study”
 - 4.1.3 Set the Coverage Spatial Structure to nothing
- 4.2 Next modify your Field Dictionary with the following:
 - 4.2.1 Change the selected Fields item to “Type”
 - 4.2.2 Add “Photographs” and “Models” to the Type field
 - 4.2.3 Remove all of the current Creators
 - 4.2.4 Add “Study Creator” to the Creator Field
- 4.3 Use the Open Directories feature to open the 3D Models folder located on the desktop
 - 4.3.1 An item should appear in the metadata management table called ‘studynew:0000001’, select the item
(The item should now be outlined in red with its respective metadata on the right)
 - 4.3.2 Modify the Type to “Models”
- 4.4 Use the Add Files feature to add all of the files in the Images folder located on the desktop
(A number of items should now appear in the metadata management table)
 - 4.4.1 Select the second item *(The items preview image should appear)*
 - 4.4.2 Use the toolbar above the image to resize the image to “best fit”
- 4.5 Select all items in the table except for the first item *(Selected items will be outlined in red)*
 - 4.5.1 Edit the selected items using the Batch Modify feature
 - 4.5.2 Replace the Title field with “Study Image” and the Type field with “Photographs”
 - 4.5.3 Modify the selected items
 - 4.5.4 Check if the selected items have actually been modified
 - 4.5.5 Use the Console to view the console messages to ensure that the batch modify was successful
- 4.6 Save the newly created metadata to the desktop with the name “study_metadata.metafy” using the Save Metadata as feature
- 4.7 Refresh the browser
- 4.8 Load your previously saved metadata (.metafy file) using the Load Metadata feature
- 4.9 Finally, use the Export Metadata feature to export the metadata
 - 4.9.1 Set the "Path to Shared Drive (Client)" and "Path to Output Directory" to the desktop
 - 4.9.2 Check to see if your metadata has been exported, there should be a folder called “Sudan_MusawwaratEsSufra_output” on the desktop

5. Metadata Management Questions

5.1 Did you successfully complete all of the ‘Metadata Management Tasks’?
Please choose one of the following (estimate where needed):

- A) Yes
- B) More than half the tasks successfully completed
- C) More than 75 percent of the tasks successfully completed
- D) Less than half the tasks successfully completed

- 5.2 Were you able to successfully set the automation fields? (YES , NO)
- 5.3 Were you able to successfully configure the automation dictionary? (YES , NO)
- 5.4 Were you able to open a directory? (YES , NO)
- 5.5 Were you able to open files? (YES , NO)
- 5.6 Were you able to modify a metadata items field value? (YES , NO)
- 5.7 Were you able to modify multiple metadata items using Batch Modify? (YES , NO)
- 5.8 Were you able to save the metadata that you created? (YES , NO)
- 5.9 Were you able to load the metadata that you saved? (YES , NO)
- 5.10 Were you able to export the metadata that you created? (YES , NO)

Instructions: For each of the following statements, mark one box that best describes your reactions to the system.

		Strongly Disagree				Strongly Agree
1.	I think that I would like to use this system frequently.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
2.	I found this system unnecessarily complex.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
3.	I thought this system was easy to use.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
4.	I think that I would need assistance to be able to use this system.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
5.	I found the various functions in this system were well integrated.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
6.	I thought there was too much inconsistency in this system.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
7.	I would imagine that most people would learn to use this system very quickly.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
8.	I found this system very cumbersome/awkward to use.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
9.	I felt very confident using this system.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
10.	I needed to learn a lot of things before I could get going with this system.	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Thank you for you for participating in the study

B6: User Acceptance Test

User Acceptance Testing

Project name: Zamani Data Archive
Project Owner: Zamani Project – Geomatics Department UCT
Supervisor: Hussein Suleman
Development Team: Michael Ferguson and Jay Benson
Test Date: 20/10/2014

Scope

In Scope

1. Metadata Management Tool (Meta-fy)
 1. Web-based
 2. Automation of Fields
 3. Console Feedback
 4. Image Preview & Manipulation
 5. Sorting of Metadata
 6. Persistence
 1. Save Metadata
 2. Load Metadata
 7. Open Files or Directories
 8. Add Files or Directories
 9. Import Coordinate Triplets
 10. Import Camera Calibration Settings
 11. Remove Camera Calibration Settings
 12. Calculate Metadata Titles
 13. Associate Files
 14. Validate Metadata
 15. Batch Modify
 16. Transform Aluka Metadata
 17. Export Metadata
2. Search Interface
 1. Search by Type
 2. Search by Collection
 3. Faceted Search
 4. Text-based Search
 5. Text-based Search Auto-complete
 6. Image Viewer
 7. Metadata View

Out of Scope

All of the required features were identified as being fundamental to producing a functional Metadata Management Tool and Search Interface.

Requirement-Based Test Case Criteria

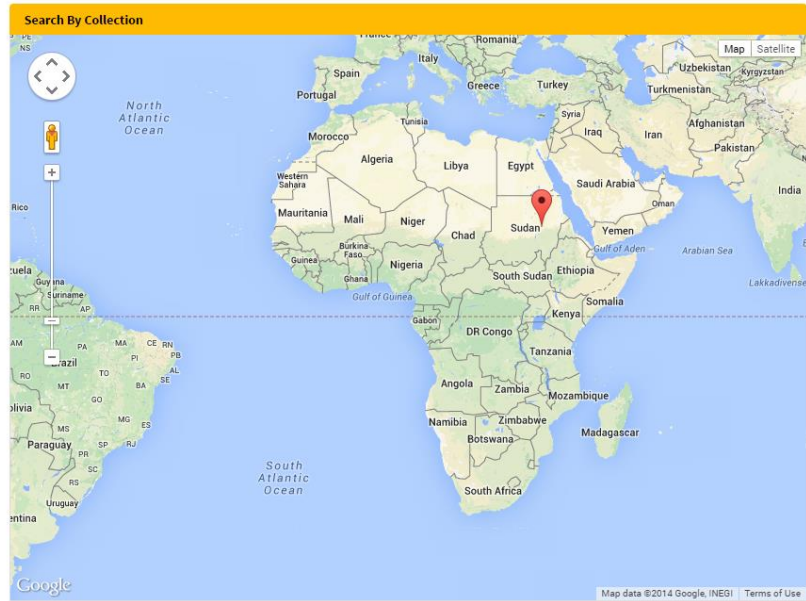
ID	Criteria
1	Metadata Management Tool (Meta-fy)
1.1	Open Meta-fy in the browser from the Web-based Backend
1.2	Automate fields using the Automate Tab
1.3	View the Console Tab after executing actions
1.4	Select a metadata item to view its preview image and use the Toolbar to manipulate the image as desired
1.5	Sort metadata by: Identity, File, Status, Tag, and Contains.
1.6	Demonstrate Persistence by saving and loading a .metafy file
1.7	Open files & open directories (clears previously open files)
1.8	Add files & add directories (adds to previously open/added files)
1.9	Import coordinate triplets from a text file
1.10	Import camera calibration settings from a text file
1.11	Remove already added camera calibration settings
1.12	Calculate metadata titles for all metadata present in the Metadata Table
1.13	Associate new files from an Aluka transformed .metafy file
1.14	Validate metadata after associating new files
1.15	Modify a batch of metadata
1.16	Transform an Aluka generated .xml file into a .metafy file
1.17	Export a set of metadata
2	Search Interface
2.1	Search by Type
2.2	Search by Collection by selecting a marker on the map
2.3	Facet search results
2.4	Text-based search
2.5	Text-based search making using of auto-complete functionality
2.6	View a large image preview in the 'Show Metadata' view
2.7	View Metadata for an item

Test Results

ID	Test Cases	Pass/Fail	Tested By	Date Tested
1	Metadata Management Tool (Meta-fy)			
2	Search Interface			

Appendix C

C1: Home Page View



C2: Show Collection View



Refine search

Type

- Photographs **229**
- Photogrammetric Images **10**
- 3D Models **4**
- Models **1**

Date

Year to Year

Format

- image/jpeg **232**
- image/tiff **6**
- model/mesh **5**
- text/plain **1**

Musawwarat es-Sufra Collection

Displaying 1-10 of 246 results.

	Undefined Author: Undefined Date: 2009-08-31 Type: Photographs Source: Undefined
	Undefined Author: Undefined Date: 2009-08-31 Type: Photographs Source: Undefined
	Undefined Author: Undefined Date: 2009-08-31 Type: Photogrammetric Images Source: Undefined
	Undefined Author: Undefined Date: 2009-09-17 Type: Photographs Source: Undefined
	Undefined Author: Undefined Date: 2009-09-17 Type: Photographs Source: Undefined

C3: Pagination


	Undefined Author: Undefined Date: 2009-09-17 Type: Photographs Source: Undefined
--	--

Prev **1** 2 3 4 5 ... 24 25 Next

C4: Show Parent Collection View



Sudan Collection



[Musawwarat es-Sufra Collection](#)
Collection: Musawwarat es-Sufra
Country: Sudan
Latitude: 16.488064
Longitude: 33.415227

C5: Faceted Search



Refine search

Type

Photographs **229**

Photogrammetric Images **10**

3D Models **4**

Models **1**

Date

Year to Year

Format


image/jpeg **232**

image/tiff **6**


text/plain **1**

Musawwarat es-Sufra Collection

1-6 of 6 results.




[Undefined](#)
Author: Undefined
Date: 2009-08-31
Type: Photogrammetric Images
Source: Undefined




[Undefined](#)
Author: Undefined
Date: 2009-08-31
Type: Photogrammetric Images
Source: Undefined

IMAGE NOT AVAILABLE

[Undefined](#)
Author: Undefined
Date: 2009-08-31
Type: Photographs
Source: Undefined



[Undefined](#)
Author: Undefined
Date: 2009-08-31
Type: Photogrammetric Images
Source: Undefined



[Undefined](#)
Author: Undefined
Date: 2009-08-31
Type: Photogrammetric Images
Source: Undefined

C6: Text-based Search

The screenshot shows the ZAMANI PROJECT search interface. At the top, there is a navigation bar with the ZAMANI PROJECT logo (a yellow outline of Africa), the text 'ZAMANI PROJECT', a giraffe silhouette, and a tree silhouette. Below this is a search bar containing the text 'sudan' and a search icon. The main content area is divided into two columns: 'Refine search' and 'Search Results'.

Refine search

- Type**
 - Photographs: 229
 - Photogrammetric Images: 10
 - 3D Models: 4
 - Models: 1
- Date**
 - Year to Year (Update)
- Format**
 - image/jpeg: 222
 - image/tiff: 6
 - model/mesh: 5
 - text/plain: 1

Search Results

Displaying 1-10 of 245 results.

Sudan Collection


Terms of Use
[Musawwarat es-Sufra Collection](#)

IMAGE NOT AVAILABLE	Undefined Author: Undefined Date: 2009-09-17 Type: Photographs Source: Undefined
IMAGE NOT AVAILABLE	Undefined Author: Undefined Date: 2009-10-07 Type: Models Source: Undefined
IMAGE NOT AVAILABLE	Undefined Author: Undefined Date: 2009-09-17 Type: Photographs Source: Undefined

C7: Text-based Search: Auto-complete


The screenshot shows the ZAMANI PROJECT search interface with an auto-complete dropdown menu. The search bar contains the text 'ph'. Below the search bar, a dropdown menu displays two suggestions: 'photographs' and 'photogrammetric images'.

C8: Show Metadata View



Archive

Undefined



Parent Collection: [Sudan Collection](#)

Collection: [Musawwarat es-Sufra Collection](#)

Author: Ruther, Heinz; Held, Christoph; Schroeder, Ralph; Bhurtha, Roshan; Wessels, Stephen

Date: 2009-08-31

Type: Photographs

Coverage (Spatial): Sudan, Musawwarat es-Sufra, Great Enclosure, Engravings

Source: University of Cape Town, Geomatics Department


Format: image/jpeg

File Size: 794.1 KB

Position:

- **Latitude:** undefined
- **Longitude:** undefined
- **Ellipse:** undefined

C9: Search by Type




Archive

Search By Type

- Photographs 229
- Photogrammetric Images 10
- 3D Models 4
- Models 1

Search By Collection



Map data ©2014 Google, INEGI | Terms of Use

C10: Show Metadata View – View Larger Image

The screenshot shows the ZAMANI PROJECT website interface. At the top, there is a navigation bar with the 'Archive' link and a search bar containing the text 'What can we help you find?'. Below the navigation bar, there is a 'Search By Type' sidebar with the following options:

- Photographs: 229
- Photogrammetric Images: 10
- 3D Models: 4
- Models: 1

A 'Search' button is located at the bottom of the sidebar. The main content area is titled 'Undefined' and features a 'Request File' button. On the left, there is a thumbnail image of a desert landscape with ruins. Below the thumbnail is a link that says 'View Larger Image'. On the right, there is a metadata table:

Parent Collection:	Sudan Collection
Collection:	Musawwarat es-Sufra Collection
Author:	Ruther, Heinz; Held, Christoph; Schroeder, Ralph; Bhurtha, Roshan; Wessels, Stephen
Date:	2009-08-31
Type:	Photographs
Coverage: (Spatial)	Sudan, Musawwarat es-Sufra, Great Enclosure, 100
Source:	University of Cape Town, Geomatics Department
Format:	image/jpeg
File Size:	194.5 KB
Position:	<ul style="list-style-type: none">Latitude: undefinedLongitude: undefinedEllipse: undefined

C11: Show Metadata View – Popup Image View

The screenshot shows the ZAMANI PROJECT website interface, similar to C10, but with a large popup image overlaid on the main content area. The popup image shows a detailed view of the desert landscape with ruins. The metadata table is partially obscured by the popup image, but the following information is visible:

Parent Collection:	Sudan Collection
Collection:	Musawwarat es-Sufra Collection
Author:	Ruther, Heinz; Held, Christoph; Schroeder, Ralph; Bhurtha, Roshan; Wessels, Stephen
Date:	2009-08-31
Type:	Photographs
Coverage: (Spatial)	Sudan, Musawwarat es-Sufra, Great Enclosure, 100
Source:	University of Cape Town, Geomatics Department
Format:	image/jpeg
File Size:	194.5 KB
Position:	<ul style="list-style-type: none">Latitude: undefinedLongitude: undefinedEllipse: undefined

Appendix D

D1: Overview Tab - Metadata View

	Overview	Edit	Console
Description			
Identifier	uctnew:0000007		
Title			
Coverage Spatial Country	Sudan		
Coverage Spatial Name	Musawwarat es-Sufra		
Coverage Spatial Structure	Great Enclosure		
Coverage Spatial Position			
Coverage Spatial Part			
Coverage Spatial Locality			
Creator	Ruther, Heinz		
Creator	Held, Christoph		
Creator	Schroeder, Ralph		
Creator	Bhurtha, Roshan		
Creator	Wessels, Stephen		
Date	2009-08-31		
Format	image/jpeg		
Source	University of Cape Town, Geomatics Department		
Type			
Position Latitude			
Position Longitude			
Position Ellipse			

D2: Batch Modify Popup

Batch Modify Selected Items

Key

CLEAR: Leave field blank IGNORE: Ignore the value in the field

Publish Status: Ignore Change Pub

Title: IGNORE

Date: IGNORE

Source: IGNORE

Type: IGNORE

Coverage Spatial Country: IGNORE

Coverage Spatial Name: IGNORE

Coverage Spatial Structure: IGNORE

Coverage Spatial Position: IGNORE

Coverage Spatial Part: IGNORE

Coverage Spatial Locality: IGNORE

Creators:

UPDATE DELETE

ADD New Creator

Batch Modify