UNIVERSITY OF CAPE TOWN

DEPARTMENT OF COMPUTER SCIENCE

CS/IT Honours Final Paper 2020

Title: Implementing the Harvester Component of a Low-Cost Cultural Heritage Web Portal

Author: Ashil Ramjee

Project Abbreviation: HERIPORT

Supervisor(s): Hussein Suleman

Category	*Min*	*Max*	Chosen
Requirement Analysis and Design	*0*	*20*	15
Theoretical Analysis	*0*	*25*	0
Experiment Design and Execution	*0*	*20*	0
System Development and Implementation	*0*	*20*	15
Results, Findings and Conclusions	*10*	*20*	15
Aim Formulation and Background Work	*10*	*15*	15
Quality of Paper Writing and Presentation	*10*		10
Quality of Deliverables	*10*		10
Overall General Project Evaluation (*this section allowed only with motivation letter from supervisor*)	*0*	*10*	0
Total marks		**80**	

Implementing the Harvester Component of a Low-Cost Cultural Heritage Web Portal

Ashil Ramjee
University of Cape Town
Cape Town, South Africa
rmjash002@myuct.ac.za

ABSTRACT

Institutions holding cultural heritage artefacts are constantly digitizing their data. However, these collections are only accessible independently in South Africa. Therefore, this project outlines a way in which data can be centralized through the use of a Web Portal. The Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) provides an interoperable means of achieving such a system. This paper focuses on the design and implementation of the Harvester component of this system. Individual functionalities of the component include OAI harvesting, conversion and indexing of metadata. The component was developed using python and Apache Solr was used for indexing for a repository. Outcomes of this paper include a Harvester that performs, integrates, and scales appropriately.

KEYWORDS

cultural heritage, portal, metadata, harvesting, OAI-PMH, records, archive, verb

1. INTRODUCTION

Through digitization, cultural heritage preservation employs a technological way of retaining cultural artefact information online in the intangible form. Beneficial to historians, school learners and the general public, cultural heritage provides the information needed to influence decisions regarding social, environmental, and political views [1]. With the use of Internet technologies such as Web 2.0, databases and the Open Archives Initiative (OAI) protocols, people with interest in cultural heritage are able to store, search, browse and analyze large numbers of cultural artefacts efficiently and reliably [2][3]. Countries are actively seeking and producing ways for users to access their digital collections. Currently, private institutions in South Africa have maintained independent digital archives while providing services such as search and browse. A few to mention include the Bleek and Llyod collection, the Digital Innovation South Africa archive, the Five Hundred Year Archive, and the Metsemegologolo archive [4][5]. However, there is no centralized repository containing a combination of all these collections in South Africa. Therefore, this project facilitated the development of a centralized cultural heritage Web Portal for South Africa. The system provides typical digital repository essential services while retrieving data by using metadata harvesting. The system was split up into three components, namely; the Data Provider Interfaces that expose the data for harvesting; the Harvester which retrieves the data, indexes it and stores it in a repository, and lastly; the Web Portal, the frontend of the system that allows users to perform services such as search and browse.

A high-level view of the three components are shown in Figure 1 below. This multi-layered architecture was inspired by the NETD architecture discussed in the background section with a few differences. With its interoperable emphasis on low coupling and high cohesion between components, the architecture allows for minimal dependencies and increases reusability, flexibility, maintainability, and scalability, therefore, containing aspects of a low-cost system [5][11].

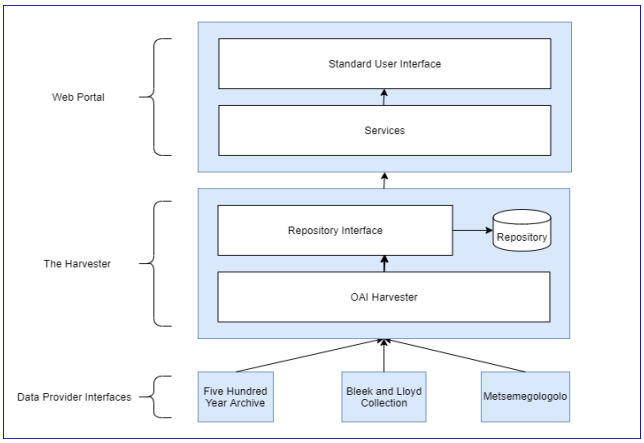

Figure 1. UML Package Diagram Depicting Overall Architecture

This paper primarily focuses on the harvester component of the system whereby the metadata used is retrieved from the data provider interfaces by using the Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) [13].

This paper discusses the aim and primary objective with regards to the project. Furthermore, it outlines the significance of the project by showing which stakeholders it benefits. Next, the background section discusses material from the literature review with regards to the project, however, focusing more on the harvester component. The next section documents the design, implementation, and evaluation of the component. Next, the results from the evaluation are discussed and, lastly, conclusions are drawn on the overall feasibility of the component.

1.1. Project Aims and Objectives

A prime example of a centralized Web Portal is Europeana, a system that retrieves data from multiple European cultural heritage collections [6]. However, such a system cannot be

applied to South Africa's environment due to the high costs of installation, maintenance, and great complexity. South Africa's current socio-economic and economic standing increases the costs of creating and maintaining these systems. The first limitation is that funding is required to implement a working system, however, stakeholders are reluctant to invest in digitizing and cultural preservation [2]. Secondly, due to South Africa's low digital literacy rate and poor education, maintenance of these systems can prove to be cumbersome [2]. Lastly, in terms of accessibility, the system is limited to a somewhat stable Internet connection, however, due to the high costs of bandwidth, users may be bandwidth constrained [2].

The project aimed to develop a standalone harvester component of the centralized cultural heritage portal that entails features that can be considered low-cost while using best practices. This was done by extensively using metadata aggregation and OAI-PMH. In terms of low-cost, the aim was to create a component that conformed to the following features: one that was simple, scalable, and easy to install and maintain. Therefore, the Harvester was made to be applicable to many systems and repositories. The primary objective of the project was to demonstrate the viability of the component as a proof of concept while using a software engineering approach.

1.2. Project Significance

As mentioned above, there is no centralized cultural heritage Web Portal in South Africa that provides cross-archive discovery. In addition, there has been no low-cost Web Portal produced anywhere. Therefore, this project delivers a working prototype that could be used as a foundation of a functional cross-archive system for South Africa. Furthermore, it provides interested research communities with the necessary building blocks that are needed to develop a low-cost Harvester component that could be applied to other environments that are similar to South Africa. In particular, the Harvester component can be applied to a variety of other working systems.

2. BACKGROUND

This section builds on the research done in the literature review while concisely focusing on the harvester. The section starts by giving an explanation about metadata and explains different metadata schemas. Next, it explains the Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH), a framework which is very important for the harvester. Lastly, information retrieval methods, architectures and toolkits are discussed.

2.1. Metadata

Digital artefacts include books, videos, paintings and more. Metadata is the data about data and in terms of cultural heritage, it could have heterogeneous meanings and different structures, therefore, remote archives maintaining this data also vary in terms of architectures [7]. Metadata are structured using metadata schemas with varying complexities. These can be based on the Extensible Markup Language (XML) which is a subset of the Standard Generalized Markup Language (SGML) [8][9][10]. Like HTML, XML can be used to describe metadata with varying complexity with the use of tags. OAI-PMH, which is key for the harvester, sends its responses structured in XML [13]. By using XML, the harvester can retrieve records that have different metadata schemas, therefore, keeping the integrity of said data. In terms of storage, metadata can be stored in XML flat files therefore databases are not needed [5]. This is the case with the Bleek and Lloyd Collection. CDWA Lite, an XML based schema that is specifically tailored to cultural heritage metadata shows how XML could be used to markup a new schema with specific needs [12].

Two examples of metadata schemas include Dublin Core (DC) and the Resource Description Framework (RDF) [7]. DC, a metadata schema used to initially describe Web resources and now a common standard schema that complies with OAI-PMH is used by many archives [7]. DC offers a simplistic, semantic interoperable schema that can be used across various disciplines [9]. DC is split into two structures, namely unqualified and qualified DC, offering flexibility and modularity. Unqualified DC comprises of 15 core elements that have been carefully moderated over the years in order to cater for multiple disciplines. On the other hand, qualified DC allows for extra elements to cater for more specificity, however, it will increase complexity. With a similar initial goal as DC, RDF is another XML based schema that offers semantic interoperability and is used for one of the biggest cultural heritage portals, namely, Europeana [6][10]. RDF has a namespace feature that allows different standards to be used within it, therefore, ensuring the integrity of the data is intact. Interestingly, these capabilities can be incorporated into other schemas, including DC [6]. The next sub-section explains OAI-PMH, which requires unqualified DC to be used when harvesting.

2.2. OAI-PMH

The Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH) is a standardized technical interoperable framework that allows one to harvest or retrieve records from a remote repository, especially for metadata aggregation [13]. Typically, the protocol entails two types of participants, one being a service provider acting as a client and a data provider that acts as a server [14]. Data providers expose their remote repositories in an OAI-compliant service, which in turn responds to requests. These requests are sent by service providers, which provide typical repository services. To put it simply, service providers harvest data that the data providers expose. The requests that service providers send are in the form of HTTP requests and the responses sent by the data providers are in the form of XML. In terms of the protocol retrieving all metadata or records, OAI-PMH defines a record as one with the following XML structure; a *header* that holds a unique identifier and date stamp regarding the creation or modification of the record; *metadata* which holds the actual metadata using the unqualified DC metadata schema or another specified schema; and *about* which is an optional tag that can hold more specific information regarding the record [13]. Notably, there are two identifiers in a record. One in the *header* and one in the DC *metadata*. However, they are different as the one in the *header* is unique to the record and the one in the *metadata* points to the source in the form of a URL.

The table below shows a list of the six requests or verbs that a service provider can request.

Table 1. OAI verbs [13]

Request or Verb	Description
GetRecord	*Used to get a single record by specifying the identifier and metadata format*
Identify	*Used to get information about the repository*
ListIdentifiers	*Used to get the identifiers of records*
ListMetadataFormats	*Used to get all metadata formats that are available in the repository*
ListRecords	*Used to get records from the repository depending on the parameters*
ListSets	*Used to retrieve the structure of sets*

For example, the following URL request can be used to retrieve all records in an OAI repository:

http://testRepo.cs.uct.ac.za/oai?verb=ListRecords&metadataPrefix=oai_dc

The verb is specified and, depending on the verb, different parameters can be specified. Conspicuously, the example retrieves all records, which is rather inefficient [14]. However, selective harvesting can be used to retrieve records that were modified or added in a specific date range with the use of date stamps or sets which means that unmodified records do not need to be harvested again [7][13][14]. However, there are complications when using the date stamps, consequently increasing the chances of missed updates or unnecessary downloads [7]. Incremental harvesting which is different from selective harvesting can be used to get data in batches to ensure flow control [14]. This facilitated by using resumption tokens on list requests. Essentially, for incremental harvests to work, repositories return data in incomplete lists with addition to a resumption token [14]. The harvester then picks up the resumption token, forms and sends another request with the resumption token appended. The repository uses this resumption token to send the next batch or incomplete list. This process continues until all batches are sent resulting in a complete list for the harvester. From the resumption token, the repository determines when the last list is sent. This sequence of requests is called a list request sequence [14].

2.3. Retrieving and Indexing Metadata

Digital archives can retrieve their metadata from multiple sources or data providers. Two ways include the use of traditional union catalogues and OAI metadata harvesting [14][16]. In terms of simplicity, the use of a centralized union catalogue that data providers manually contribute to would be the easiest way of retrieving data [16]. However, this is done in a non-automated way and the onus is on data providers to provide metadata that is accurate. Accuracy and data integrity concerns arise when metadata needs to be converted [11][16]. On the other hand, OAI-PMH provides a means for retrieving data in an automated, incremental, and interoperable manner [13][16]. Harvested metadata may need to be converted into a single schema and this is done by using mapping and cross walks [16].

Once retrieved, the metadata needs to be indexed to facilitate search and browse services effectively and efficiently [18]. MIQS, a service used for indexing and querying uses an in-memory index to achieve efficiency in searching [23]. However, MIQS focuses on self-describing file formats [23]. Two effective low-cost strategies of indexing are outlined in Annika, et al's paper on the low-cost semantic enhancement of metadata and indexing [18]. These are lexical indexing and semantic indexing. Lexical indexing creates index entries of term locations after examining all text, therefore, it is a full-text indexing strategy [18]. Contrastingly, semantic indexing matches concepts and then indexes them to full-text indexes if they match. This is done by using mining techniques from other sources. Interestingly, Annika, et al's paper also draws upon the idea of enhancing lexical searching through semantics by using concept labels [18].

2.4. Architectures

The simplest, most cost-effective approach to creating a Web portal with all its components would be to implement a union catalogue [16]. INFLIBNET, a union catalogue, was created to centralize academic libraries in India [17]. Its three-tier architecture offers a system with both backend and frontend services.

By using the Z39.50 protocol, federated search engines can be implemented to multilingually search through heterogeneous collections [15]. This happens when queries are sent to registered sites. Once sent, results are returned and merged. One downfall of this approach is that each registered site has its own search syntax, consequently increasing the complexity when the system is scaled up [15]. However, issues with this protocol were eliminated as it was refined. Following Z39.50, Z39.50 International: Next Generation (ZING) was introduced which expands the functionality of Z39.50 [15].

Contrastingly, the National Electronic Thesis and Dissertation Portal in South Africa (NETD) makes use of OAI-PMH in its architecture, which comprises of a Web Portal, a Harvester, and a Repository [11]. OAI-PMH provides an interoperable connection between the three layers of components and ultimately allows for reusability and scalability [11]. The Networked Digital Library of Theses and Dissertations (NDLTD) follows a similar approach to NETD, Both, NETD's and NDLTD's repositories are OAI-conformant, therefore, acting as a data and service providers [8].

Europeana, a cultural heritage Web portal that uses OAI-PMH, caters for rich data on large scale with high complexity, consequently having high development and implementation costs [6]. Its complex architecture uses RDF, APIs and the OAI Object Reuse and Exchange framework (OAI-ORE) to display complex digital objects [6].

Contrastingly, the Bleek and Llyod collection is an archive with low-cost design principles that stores its data in flat XML files [4][5]. Its portable architecture uses XML, XSLT, XHTML. It uses XSLT keys to index its data and an Ajax-based system for searching [5]. With the same repository architecture, The Five Hundred Year Archive (FHYA) is based on some low-cost principles that are derived from the Bleek and Llyod collection, however, it uses a JavaScript-based faceted search engine [4]. Effectively, this architecture allows one to implement a repository without a database [5].

2.5. Toolkits

Toolkits offering repository and harvesting services reduce the burden of creating a centralized archive or digital library from scratch. Furthermore, they provide the means for creating the Harvester component. However, they require knowledge of installation and maintenance, and a stable internet connection [4]. Such toolkits include DSpace, Fedora and Omeka, which have their independent advantages. All three toolkits provide the means to support OAI-PMH [19][20][21].

DSpace, the simplest to install and configure, is an open-source toolkit that can be used to implement an archive [19]. One downfall is that it is specifically tailored to be used by institutional archives. Also, due to its nonconformity to DC, there may be inconsistencies and incomplete records when harvesting [19]. Fedora, an open-source toolkit, was built upon the idea of object models and offers standard Web services such as search, browse and indexing [21]. However, its implementation comes at a cost of complexity. More suitably, Omeka is another open-source toolkit that is tailored towards offering repository services with regards to cultural heritage data by allowing customization of metadata schemas [20]. Although the toolkits are worth mentioning, they were not developed to serve as meta-archives, therefore they lack the functionality needed to implement the Harvester component in this case.

3. REQUIREMENTS AND DESIGN

This section discusses the in-depth design and implementation of the system. It starts with a sub-section explaining the initial requirements gathering phase. Next, the system architecture along with the development environment is explained.

3.1. Initial Requirements Gathering Phase

Initial project requirements first consisted of input from our supervisor in the form of emails and meetings. Requirements were to create a Harvester component for cultural heritage metadata using best software engineering practices that could scale up to 30000 records. Furthermore, it was suggested we use the Dublin Core metadata standard as a start before contacting the data providers. The next part of the requirements gathering phase was to inform our data providers of our project, our progress and what we needed from them in terms of raw data. In addition, we requested suggestions on how their metadata should be mapped to Dublin Core and any other feedback. It was decided that no toolkits were to be used due to its increasing cost of complexity to the system. Furthermore, they provide many features that would not be used by the system, therefore providing redundant code. From these emails and meetings, we also determined it was best to use Python as the programming language, an Apache Web Server running in a Linux environment and an Amazon EC2 virtual machine to host the server. Python provides a simplistic and understandable programming language with useful packages for the URL and HTTP requests. The Apache Web server was suggested as it is commonly used and is one of the simplest. The Amazon EC2 virtual machine was chosen as it offered a free tiered account. From our data provider emails, it was determined that some metadata fields would need to be linked. For example, the *event* tags were suggested to be linked from the Five Hundred Year Archive (FHYA).

3.2. System architecture

From the requirements gathered, the following use case was formed in Figure 2 below. This formed the underlying basis of the Harvester design. A basic summary of the use case is that a user should be able to run a whole harvest process with one command. In addition, users should be able to run the specific functionalities within the harvest process.

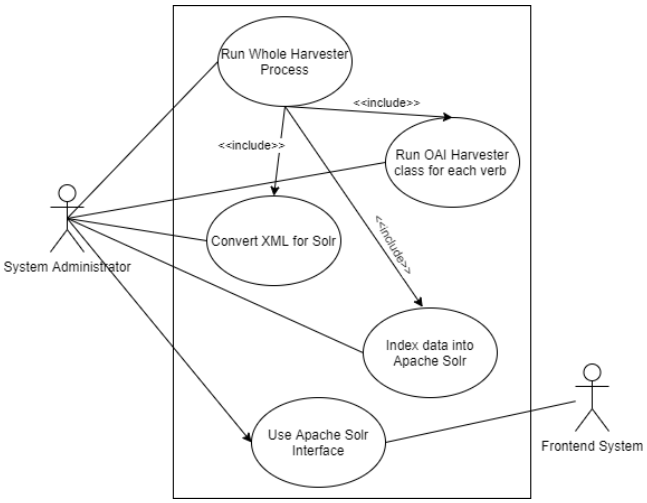

Figure 2. UML Use Case Diagram

The Harvester component consists of the OAI harvester and repository. From Figure 1 in section 1, the Repository Interface refers to the interface provided by Solr and the Repository refers to XML flat files. The OAI harvester harvests records into the repository from the provided data provider interfaces. With the use of an interface, the frontend Web Portal can access the repository and provide value-added services to different users.

From OAI-PMH, *ListRecords* was the only verb used throughout our overall architecture. However, the other five verbs were also implemented as it would prove to be useful to the research community. Therefore, six python classes were created for the OAI harvester, with each capable of harvesting data related to the verb. Required parameters for the ListRecords verb include the *metadataPrefix* [14]. Optional parameters include *from*, *until* and *set*. One exclusive parameter that was used is the *ResumptionToken*, which facilitates flow control [14]. Essentially, the Resumption Token is used for incremental batch harvesting. Each of the six python classes uses a *httpRequest.py* class to send the request and get the response.

Records are harvested in the form of flat XML files and parsed using an XML DOM parser package which converts each into an Element Tree. A command-line interface was used to supply each verb with required and optional parameters. Apache Solr, an open-source scalable search platform that was built on Apache Lucene was used to implement the index [22]. It provides a simple repository interface to allow for indexing and searching and was also used in the frontend of the system. It takes files in the form of XML that is specific to Solr and indexes them into the platform. The *dc2solr* python class with an XSLT template was used to convert the harvested XML into XML specified by Solr. This

specific XML uses *doc* tags for each record and *field* tags for each DC attribute within each record. All records are then enclosed within an *add* tag.

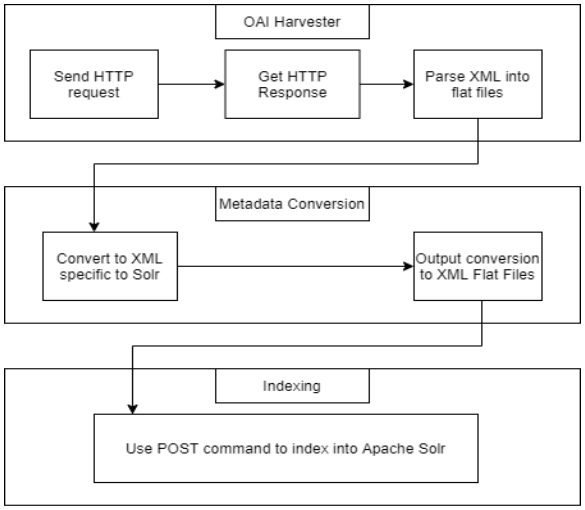

Figure 3. High-Level Overall Process

The full sequential process (Figure 3) of how the metadata ends up in the repository is as follows: metadata records are harvested using *ListRecords.py* in the form of flat XML files; these files are then converted to a specific XML using *dc2solr.py* and are then indexed into Apache Solr using the POST terminal command, ready to be used by the Web Portal. To initialize this process, two bash scripts are used, namely, *harvestAll.sh* and *harvestToDate.sh*. The first script executes the process mentioned above. The second executes the same process; however, it uses a data text file in order to implement selective harvesting. Both these processes are scheduled using a crontab in a Linux environment. A more low-level view of the process is shown in Figure 4 below. The user interacts with the harvestAll.sh bash file which then facilitates the OAI harvesting, conversion, and indexing.

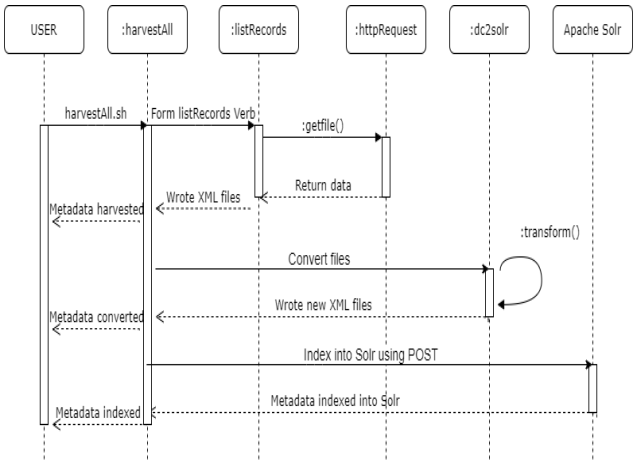

Figure 4. UML Sequence Diagram of the Whole Process

Running and installation of the Harvester component requires a user with knowledge of Linux systems. However, the two bash scripts can easily be altered in order to run in Windows or MacOS environments. The ease of implementation addresses the concerns of the lack of digital literacy in South Africa.

3.3. Development Environment

The code was developed using Python 2 due to some of the packages being used. Efforts were made to conform to the PEP 8 style guide for Python code to ensure best practices. Apache Solr version 7.7.3 was used to implement the index. XML files were stored as local flat files without the need for a database. All code was developed and tested in a local Linux environment with a machine with a 9th Gen Intel Core i7 with 6 CPUs @2.6GHz, 16gb RAM and a 512gb SSD. To host the Web portal using Apache Web Server and, run and test the Harvester regularly, a server provided our Computer Science department was used. Server specifications were as follows: 10 CPU Intel(R) Xeon(R) CPU @2.20Ghz with a virtual machine allocated for us to use on the server. The virtual machine was allocated the following specifications: 4 CPUs, 32gb RAM and 2TB HDD. Performance of the virtual machine varied from 95% to 100%. Testing in section 5 refers to Machine 1 (M1) and Machine 2 (M2). In this case, M1 is the machine with the Intel Core i7 and M2 is the machine with the Intel Xeon Processor. Git was used as version control to ensure that all changes made to the code were discoverable.

4. IMPLEMENTATION

The overall approach taken for the project was a combination of characteristics taken from the Iterative and Agile software methodologies. Agile's advantages over the Waterfall approach allowed us to overcome the quickly changing requirements and to make the necessary adjustments needed in a timely manner. The project consisted of four iterations, each with a different timing schedule. Each iteration followed the Agile framework of re-assessing requirements, designing, implementing, and integrating with the whole system. Weekly meetings ensured that everyone was on track with project deliverables. From the Extreme Programming framework, our test-driven development and constant refactoring of code ensured efficiency, understandability, interoperability, and successful integration. The next four sub-sections outline these four iterations.

4.1. Initial Setup and Development of Skeleton

After acquiring our initial requirements, iteration one commenced. The Apache Web Server was set up on the EC2 virtual machine and development commenced. The first and most important part of the harvester, namely *httpRequest.py*, and *listRecords.py*, were coded. This formed the basis for the other five verbs that were implemented in iteration two. Harvested files were stored as XML flat files. With the use of a combination of black and white box testing, unit testing was conducted for both the classes. Each required and optional parameter was tested using the following repository, which is OAI-conformant: *www.hindawi.com/oai-pmh/oai.aspx*. One challenge we faced was that the Amazon EC2 virtual machine executed its services very slowly due to the account tier type. In addition, it incurred additional charges even though it was a free account. Upon this discovery, the account was terminated, and we shifted to using a more sustainable server provided by the Computer Science department.

4.2. Full Implementation of OAI Harvester

Iteration Two started by revisiting the initial requirements and making any necessary adjustments. In addition, it consisted of the full implementation of the other five verbs required in an OAI harvester component. These verbs were *listSets.py*, *identify.py*, *listMetadataFormats.py*, *listIdentifiers.py* and *getRecord.py*. Similarly, to the previous iteration, all six verbs were unit tested using black and white box testing. However, these were tested using *www.hindawi.com/oai-pmh/oai.aspx* and our first data provider interface, namely the Bleek and Lloyd Collection. During the unit testing, all required and optional parameters were tested using the command line interface specific to each verb. Upon the successful unit testing with the first data provider interface, integration between our Harvester component and the data provider interface was confirmed.

4.3. Conversion, Indexing and the Repository

Similarly, to the previous iteration, requirements were reassessed. It was determined that multiple indices were to be used to index the data. Other than all fields being indexed, the OAI-DC title and description field were to be linked. Iteration three consisted of the conversion, indexing of metadata and implementation of the repository. From the previous iteration, records were harvested and stored into XML flat files. Apache Solr was configured and used to implement the index while providing the necessary frontend services and connecting interface needed for the Web Portal component. However, Apache Solr would only take in XML files in a form specific to Solr. Therefore, the *dc2solr.py* and *dc2solr.xsl* files were created to facilitate this conversion. Next, by using a Solr POST terminal command with specified parameters, files were indexed into Solr. Referring to the Apache Solr documentation, it was determined that a single index would have the same functionality as multiple indexes by specifying a type value, therefore a single index was used for each record. It should be noted that there can be multiple tags or fields of the same key with different values in data provider XML responses. For example, there could be one or more *dc:creator* tags. Apache Solr deals with this by combining the tags into one tag and making the values a list. Apache Solr requires a unique identifier for each record. Initially, this identifier was set as the *dc:identifier* from the record. However, upon unit testing, it was discovered that some repositories have multiple dc:identifier tags pointing to different URLs which caused problems when indexing into Apache Solr. Consequently, the identifier included in the header of a record was used instead as the unique identifier. All DC tags including *dc:identifier* were still included when indexing into Solr. All files were to be stored locally as flat files without the need for a database.

Unit testing for this process was facilitated by converting data after harvesting from our three data provider interfaces. Indexing of metadata into Apache Solr ensured successful integration of the Harvester component and Web Portal. Next, the two bash scripts to fulfil the whole harvesting process were created, providing a simple and scheduled means of running the harvester. In this iteration, we determined that we had a fully integrated system.

4.4. Evaluation and Testing

Iteration four consisted of the overall evaluation of the harvester component. Firstly, unit testing was conducted again for each python class and terminal commands to ensure a correctly working component in compliance with OAI-PMH.

Next, full integration testing of the Harvester component was conducted. This was done by using the white box testing techniques to ensure the correct functionality and correct data-driven input and outputs. These tests were conducted on a pass-fail basis by accessing each use case in the process. The basic idea was to ensure that all metadata from the three data providers were harvested, converted, and indexed into the repository.

Following the integration testing, performance and scalability testing were conducted. Furthermore, incremental selective harvesting was also tested. For performance and scalability testing, individual components were tested using two different machines in a Linux environment and with three different repositories, each with different data sizes. Selective harvesting was used to approximate the number of records harvested. The three different repositories tested were our three Data Providers Interfaces, the Hindawi collection containing publicly accessible research journals, and the OpenUCT repository containing educationally oriented metadata. These tests were conducted by timing the whole harvest process. Outputs from the three repositories and two machines were compared and evaluated. A general Internet speed test containing download and uploads speeds were conducted on the two machines as this plays a factor in harvest speeds. In addition, a performance comparison of the three repositories was conducted. This was done by testing the local code and the end-to-end code. To test the scalability of the harvester, data sizes varying up to 30000 records were tested. To test the incremental selective harvesting, OpenUCT's repository was used since it is manageable in terms of size. Records were harvested using specific dates to ensure no duplicates and no records were left out. In order to do this, all records were first harvested to verify the number of records. Next, records were harvested in increments by specifying dates up until the current date. A successful harvest was determined by summing the number of incremental records and comparing it with the number of records in the repository.

In terms of usability testing, end-users would only end up seeing the metadata of the Web Portal due to the Harvester component being a backend service. However, ways in which users expect their search results related to the indexing, which entails the following question: "*Were your search results as expected?*". Therefore, this question was answered when a fellow group member conducted their user evaluation on the Web Portal component. The test conducted was a user task evaluation. This question was targeted towards five participants made up of people from our data providers whom each represented their respective collections.

5. RESULTS AND DISCUSSION

This section discusses the outcomes of the testing undertaken on the Harvester component. These include unit, integration, performance, unit, and scalability testing.

Examples of inputs and outputs from unit testing the process in Figure 3 are illustrated below. Firstly, a ListRecords request is formed and sent. This is formed using the terminal command shown in Figure 5 below.

```
ashil@ashil:~/HERIPORT/harvester$ python2 listRecords.py
 -l http://rafiki1.cs.uct.ac.za/~alex/cgi-bin/MetsemegologoloOAIInterface.py -o dcXML/metse.xml -m oai_dc
```

Figure 5. Command for Simple ListRecords Response

Figure 6 below shows a portion of the flat XML created after the command above is executed. The output consists of a record with a *header*, *metadata* in the form of unqualified DC and the *about* tag.

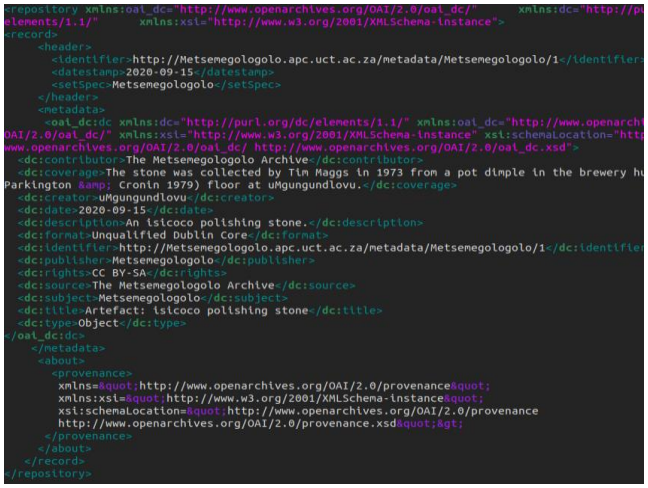

Figure 6. Flat XML File Result from the Response

Next, the conversion takes place. Figure 7 below shows the XML flat file after conversion. Note how the DC attributes are now *field* tags. The identifier from the header is included as the unique identifier.

Figure 7. XML Flat File After Conversion

Figure 8 below, shows the data in the Apache Solr interface after being indexed by Solr. Once indexed, a version number is also included to ensure accurate version control of the record.

Figure 8. Data in Apache Solr

Table 2. Unit and Integration Testing

Test	Outcome (Pass/Fail)
OAI Harvesting	Pass
All records with only 'metadataPrefix' parameter specified	*Pass*
Using all subsets of 'from', 'until' and 'set' parameters	*Pass*
Test 5 other verbs	*Pass*
Conversion	Pass
Convert flat XML files for Solr	*Pass*
Indexing	Pass
Index into Solr Repository	*Pass*
Process Bash Scripts	Pass
HarvertAll.sh	*Pass*
HarvestToDate.sh	*Pass*

Table 2 above, illustrates the full functionality of the Harvester component in line with the process shown in Figure 3 in section 3.2. Furthermore, it shows the use case tests undertaken to determine the successful integration of the system, therefore ensuring interoperability between components. The OAI harvester managed to harvest all records from our three Data Provider Interfaces, ultimately ending up in the Apache Solr repository. One limitation that we faced was that a Virtual Private Network (VPN) was to be used to retrieve records from our data provider interfaces due to the nature of our server environment, therefore this may have affected the performance.

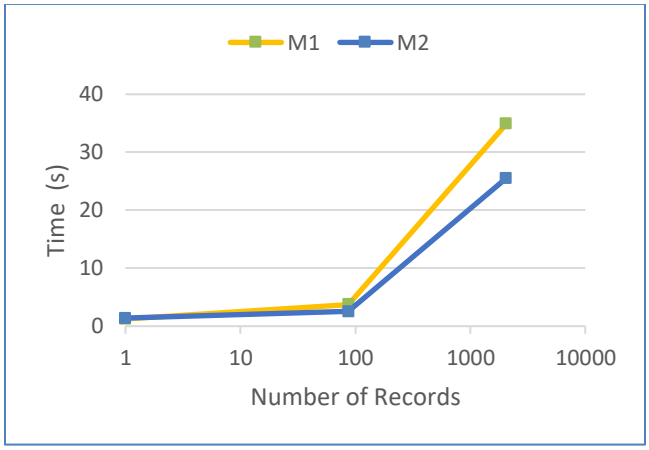

Figure 9. M1 vs M2 (3 Data Provider Interfaces)

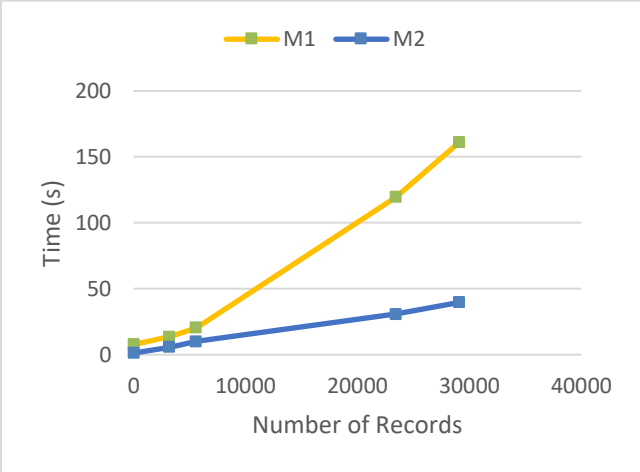

Figure 10. M1 vs M2 (OpenUCT)

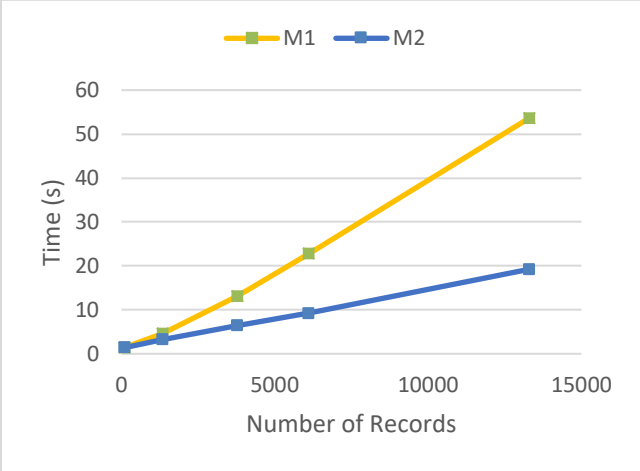

Figure 11. M1 vs M2 (Hindawi)

Table 3. M1 vs M2 Internet Speeds (Mbit/s)

	Download	Upload
M1	7.40	2.98
M2	840.02	4.12

Following the integration testing, performance testing was led by stress-testing the Harvester on different machines and repositories. Timing for the whole harvesting process was done for different dataset sizes. Firstly, the performance of two machines, namely M1 and M2, was compared on each repository. Machine specifications are outlined in section 3.3. The outcomes of these tests are depicted on Figures 9,10 and 11. Both Figures 10 and 11 show a large performance difference in terms of M1 and M2 whereas Figure 9 shows a smaller difference. As expected, M2 with a Xeon processor outperforms M1 with an i7 due to its bigger L3 cache. Figure 9 shows a test whereby records were harvested on the same machine containing the three Data Provider Interfaces, therefore, fewer factors contributed to the time taken. One of these factors is the Internet speeds shown in Table 3. The larger time difference in Figures 10 and 11 show how M2's higher download speed of 804Mbit/s contributes to better performance. Therefore, from these tests, we have determined that harvesting speed increases with a better processor and better internet. Interestingly, it also shows how the harvester can be run on a laptop (M1) and still have relatively good performance. Therefore, in terms of low cost, the Harvester component can have lower funding costs in order to implement and maintain. Furthermore, it shows that Gigabit internet connections are not needed, addressing the concerns of unstable or slower internet connections in South Africa.

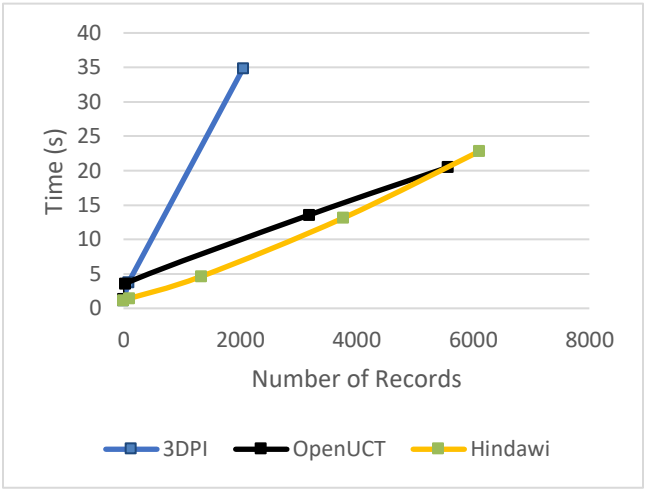

Figure 12. Repositories Comparison (End to End)

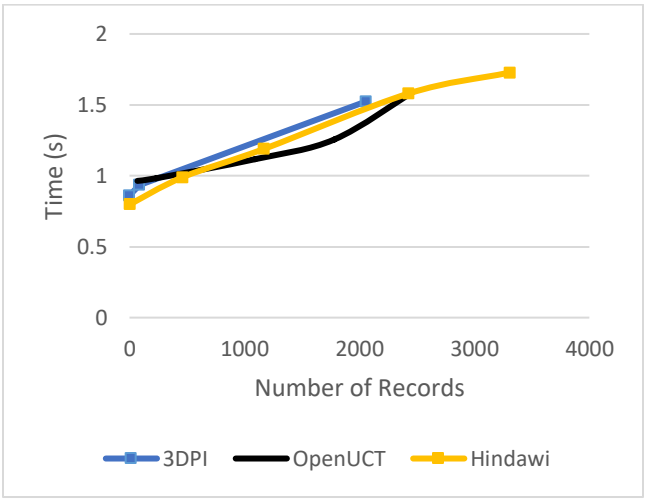

Figure 13. Repositories Comparison (Local)

Next, the performance of the three repositories was compared. All tests were run on M1 to minimize inconsistencies. Figure 12 illustrates the tests conducted from the time the HTTP request is sent until the files are indexed into Solr. Therefore, this tests the speed of both the code and the data providers. Results from these tests show that both the OpenUCT and Hindawi repositories have similar harvesting performance speeds. On the other hand, our three Data Provider Interfaces illustrate a performance hit

compared to the other two repositories. Interestingly, this mostly depends on how the repository generates its XML response. Factors that affect this generation include processing and conversion of metadata. It should be noted that the Hindawi average times were affected by a first outlier, therefore, times for receiving the first response from Hindawi were left out. This could be explained by the repository converting its metadata to comply with the OAI protocol upon the first response sent by any machine. However, further responses seemed to have used this already pre-processed data, which it probably holds temporarily. Contrastingly, our three data providers do not store pre-processed data. Conversion to unqualified DC is done upon creating the response. Therefore, the disparities between the three repositories and their performance are primarily derived from the way they form their XML responses. Figure 13 shows the test conducted after the HTTP response until files are indexed therefore testing the code only. As expected, the three repositories perform and scale similarly, therefore confirming that the performance differences in Figure 12 are explained by the external formation of XML responses.

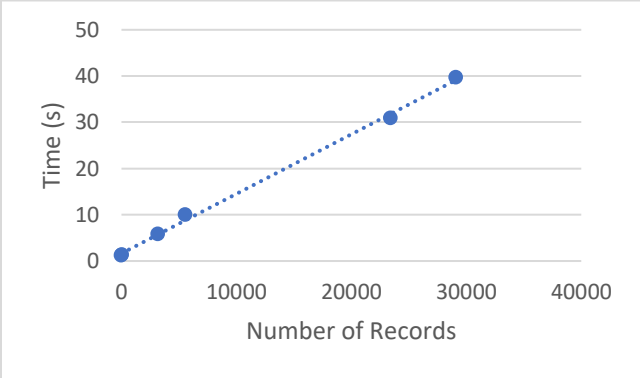

Figure 14. Scalability Illustration

As per requirements, the Harvester component needed to scale up to records of a number greater than 30000. However, our Data Providers Interfaces only provided a total of 2145 records. Therefore, the OpenUCT repository was used to test the scalability of the component as it had 29067 records. As shown in the scatter plot in Figure 14, the harvester positively scales linearly as the number of records increases. From the graph, the positive linear trend indicates that scalability is catered for by the Harvester component. However, as seen before, the onus is not only on the Harvester component to ensure a linear scaling as data provider pre-processing may factor into the scaling.

Table 4. Incremental Selective Harvesting for OpenUCT

Increment Batch	Number of Records
1st Batch	4702
2nd Batch	858
3rd Batch	91
4th Batch	600
5th Batch	22836
Total	**29067**

Table 4 illustrates the purpose of selective harvesting. Selective harvesting was conducted on the OpenUCT repository that had 29067 records. Batches were harvested with varying time periods, which were set using the *from* and *until* parameters for ListRecords. The total number of harvested records from the batches equals the number of records in the OpenUCT repository. This confirms that there were no duplicates or missing records when selectively harvesting. Benefits of incremental selective harvesting tackle concerns of scalability and efficiency factors due to no repeated harvesting of unmodified records. Therefore, records that are only newly created or modified are harvested from this repository.

From the user evaluation conducted by a fellow group member, all 5 participants answered the question at hand with a "*Yes*", indicating that the participants were satisfied with their search results, ultimately, suggesting that the records were indexed in the correct manner.

6. LIMITATIONS AND CHALLENGES

In terms of the development of the Harvester component, one limitation we had was the size of the dataset acquired from our data providers. We were supplied with a subset of the actual metadata displayed on their sites, except for the Bleek and Lloyd collection. Ideally, it would have been beneficial to acquire the whole dataset in order to have more accurate performance and scalability testing results.

In terms of the project itself, challenges included undertaking such a project during a pandemic due to Covid-19. Understandably, communication between stakeholders was slower than originally anticipated. Furthermore, most of the project was conducted in a rather challenging work environment due to the split of group members in different countries.

7. CONCLUSIONS

We hoped to develop a functional low-cost cultural heritage Web portal system that could be used as a basis for other similar systems with emphasis an on the low-cost aspect. Although making comprises, the system was easy to install and run with little knowledge of Linux environments.

Key findings during the testing included the ability to cater to performance and scalability aspects. Another interesting finding was that the OAI harvestngi times also depended on any pre-processing done by OAI-conformant repositories which were confirmed by Figures 12 and 13.

The development of the Harvester component was concluded as a success in terms of integration, performance, and scalability. Furthermore, in terms of the whole system, all components integrated successfully providing us with a sensibly working Cultural Heritage Web Portal. We have met our project aim as our specific success factors have been met. Furthermore, we provide to the research community a project of reasonable significance which can be used to dive into aspects of low-cost metadata aggregators.

8. FUTURE WORK

Future work that could be done to improve on the Harvester component and the overall system would be to implement an architecture similar to the Bleek and Lloyd collection. This would remove the need for databases and ultimately Apache Solr. Consequently, this would increase the complexity of the frontend of the system. Another future aspect that can be looked into would be to incorporate the use of multiple metadata schemas within the whole system, however, this would also contribute to the complexity of the system.

ACKNOWLEDGEMENTS

I would like to thank all stakeholders involved in the project. These include my reliable group members, Alex Priscu and Toshka Coleman, our supervisor Hussein Suleman, our second reader and our data providers. I would like to thank Hussein Suleman and the Computer Science Department for providing the server for our development and testing purposes.

REFERENCES

[1] Michele Pickover, Dale Peters. 2002. DISA: An African Perspective on Digital Technology. *Innovation, 24.*

[2] Hussein Suleman. 2011. An African Perspective on Digital Preservation. In Multimedia Information Extraction and Digital Heritage Preservation, 295- 306. https://doi.org/10.1142/9789814307260_0016

[3] Hanumat G. Sastry, Lokanatha C. Reddy. User interface design principles for digital libraries. International Journal of Web Applications, 1(2), 86-91. http://dirf.org/ijwa/v1n20109.pdf

[4] Hussein Suleman. 2019. Reflections on Design Principles for a Digital Repository in a Low Resource Environment. In Proceedings of HistoInformatics Workshop 2019, 13 September 2019, Oslo, Norway, CEUR. http://pubs.cs.uct.ac.za/id/eprint/1331/1/ho_2019_lowresource.pdf

[5] Hussein Suleman. 2007. Digital libraries without databases: The bleek and lloyd collection. In International Conference on Theory and Practice of Digital Libraries. 392-403. Springer, Berlin, Heidelberg

[6] Martin Doerr, Stefan Gradmann, Steffen Hennicke, Antoine Issac, Carlo Meghini, Herbert van de Sompel. 2010. The Europeana data model (edm). In World Library and Information Congress: 76th IFLA general conference and assembly. 10-15.

[7] Herbert Van de Sompel, Michael L.Nelson, Carl Lagoze, Simeon Warner. 2004. Resource Harvesting Within the OAI-PMH Framework. D-lib magazine. 10(12). https://digitalcommons.odu.edu/cgi/viewcontent.cgi?article=1002&context=computerscience_fac_pubs

[8] Hussein Suleman, Edward A. Fox. Towards universal accessibility of ETDs: building the NDLTD union archive. In Fifth International Symposium on Electronic Theses and Dissertations (ETD2002), Provo, Utah, USA (Vol. 30).

[9] Stuart Weibel. 1997. The Dublin Core: A Simple Content Description Model for electronic resources. Bulletin of the American Society for Information Science and Technology. 24(1), 9-11. https://asistdl.onlinelibrary.wiley.com/doi/full/10.1002/bult.70

[10] Judith R. Ahronheim. 1998. Descriptive Metadata: Emerging Standards. Journal of academic librarianship, 24(5), 395-403.

[11] Lawrence Webley, Tatenda Chipeperekwa, Hussein Suleman. 2011. Creating a national electronic thesis and dissertation portal in South Africa. http://pubs.cs.uct.ac.za/id/eprint/748/1/etd2011_webley.pdf

[12] Regine Stein. Erin Coburn. 2008. CDWA Lite and museumdat: New developments in metadata standards for cultural heritage information. In Proceedings of the 2008 Annual Conference of CIDOC 15-18. https://pdfs.semanticscholar.org/5673/ced82275df25749a62297cf412b31b59621a.pdf

[13] Carl Lagoze and Herbet Van de Sompel, 2001. The Open Archives Initiative: Building a low-barrier interoperability framework, Proceedings at Joint Conference on Digital Libraries. 54-62.

[14] Carl Lagoze, Herbert Van de Sompel, Michael Nelson, and Simon Warner, 2002. The Open Archives Initiative Protocol for Metadata Harvesting, Open Archives Initiative. http://www.openarchives.org/OAI/openarchivesprotocol.html

[15] James Powell, Edward A.Fox. 1998. Multilingual Federated Searching Across Heterogeneous Collections. In D-Lib Magazine. 4(8). September 1998. http://www.dlib.org/dlib/september98/powell/09powell.html

[16] Mary S. Woodley. 2008. Crosswalks, metadata harvesting, federated searching, metasearching: Using metadata to connect users and information. Getty Research Institute. http://scholarworks.csun.edu/handle/10211.2/2001

[17] Prem Chand, Suresh K. Chauhan. 2008. The union catalogue of academic libraries in India: an initiative by INFLIBNET. Interlending & Document Supply

[18] Annika Hinze, Sally Jo Cunningham, David Bainbridge, J. Stephen Downie. 2016. Low-cost semantic enhancement to digital library metadata and indexing: Simple yet effective strategies. In 2016 IEEE/ACM Joint Conference on Digital Libraries (JCDL). 93-102. IEEE.

[19] Mary Kurtz. 2010. Dublin Core, DSpace, and a brief analysis of three university repositories. Information technology and libraries, 29(1), 40-46. https://ejournals.bc.edu/index.php/ital/article/view/3157

[20] Jason Kucsma, Kevin Reiss, Angela Sidman. 2010. Using Omeka to build digital collections: The METRO case study. D-Lib magazine. 16(3/4), 1-11

[21] Thorton Staples, Ross Wayland, Sandra Payette. 2003. The Fedora Project. DLib Magazine, 9(4), 1082-9873. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.458.4750&rep=rep1&type=pdf

[22] Apache Solr. Available: https://lucene.apache.org/solr/

[23] Wei Zhang, Suren Byna, Houjun Tang, Brody Williams, Yong Chen. 2019. MIQS: metadata indexing and querying service for self-describing file formats. Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis. 1-24.