Enhanced Processing of METS/MODS Library Metadata in CouchDB

Araek Tashkandi
Faculty of Computing and Information Technology
King Abdulaziz University, Jeddah, Saudi Arabia.
Email: asatashkandi@kau.edu.sa

Lena Wiese
Institute of Computer Science
Georg-August-Universität Göttingen
Email: lena.wiese@cs.uni-goettingen.de

***Abstract*—Efficient metadata management is decisive for a useful resource discovery for example in digital libraries. In this paper, we present a transformation of library metadata in METS/MODS format into JSON, so that it can be managed by the CouchDB document database. In this way, metadata searching and analysis became much faster and new features could be added to the legacy system.**

I. Introduction

Getting more evolved in internet technologies, online access has become a norm in many fields. To cope with this digital age, most of the libraries offer their physical resources to be accessed virtually in digital libraries. Their goal was to provide full access to their resources and collections. To reap the benefits of technological development of the digital library and to make effective use of digital collections, creation of good quality metadata is required. Otherwise, without having good metadata, matching digital resources cannot be easily located by users and digital collections cannot be administered effectively. Search engines for literature search systems frequently rely on metadata over content especially when the content cannot easily be scanned and understood. However, library staff usually face many challenges managing these metadata collections of digital resources whose size tends to reach the big data border. This paper considers a literature metadata collection of digitized works maintained by Georg August University of Goettingen, Lower Saxony State and University Library in Germany. To move beyond individual repositories of digital resources, Goettingen university library uses OAI metadata harvesting to collect and harvest METS (Metadata Encoding and Transmission Standard) & MODS (Metadata Object Description Schema) collections that describe the bibliography metadata collection of digital works from different providers. In this way it allows the collections of digital resources to be searched together in a federated fashion. Thus, after the metadata collections are harvested, Goettingen library provides them to aggregators such as ZVDD (Zentrales Verzeichnis Digitalisierter Drucke; central repository of digitized prints). The ZVDD provides library users with a search system on top of a central index on these data [1].

II. Problem Statement

The ZVDD is a directory that acts as a public access point for searching among collections of digitized objects, currently comprising of approximately 1.2 million items (comprising around 90GB) from the 15th century and newer, all digitized in Germany. The role of the Goettingen Library is that it collects METS documents about the digitized objects from various data providers - other libraries - so they can be integrated and then directly published to ZVDD. At the beginning, all the METS documents are downloaded by OAI-PMH, a protocol for metadata harvesting. Unfortunately, not all the data providers are able to provide their data without validity errors or other content and quality issues. Therefore, all the gathered data have to be carefully processed, validated and when no serious obstacles are found, finally forwarded to the output repository. If successful, cleansed METS documents are then ready to be provided to the ZVDD aggregator, once again via OAI-PMH.

Metadata harvesting produces huge data files; large collections of such metadata pose a performance problem to existing literature search systems and searching for a required resource in the traditional XML format (METS & MODS) takes too much time. This is in particular due to the fact that the search in the legacy system is performed on the whole metadata file; each metadata file has three main parts: descriptive metadata, administrative metadata, and structural metadata and hence time is wasted by going through a lot of unnecessary information. The objective of this project is to propose improvements of managing and querying over a collection of digitized objects maintained by the Goettingen University Library and to make resource discovery by metadata of ZVDD as efficient as possible. However at the same time it must be ensured that the legacy systems relying on traditional data formats remain productive.

III. Approach

The main idea is to extract from the long METS and MODS metadata records only the metadata parts which are required in the resource discovery process and transform it into Java Script Object Notation (JSON; see [6]) – and then to improve managing and querying over this collection by uploading the output JSON files into the NOSQL database CouchDB (see [10], [11]). The main part of the metadata that is needed for resource discovery, is the information in the descriptive metadata part that has the bibliography descriptions. This is the information used when searching for a resource by details such as the author name and the publication year.

There are three main types of metadata (see [2], [3]):

1) Descriptive metadata: describes intellectual contents (i.e. title, author etc.) to identify the resource.

2) Structural metadata: includes the information that indicates the internal structure of a resource which is comprised of multiple files. For instance, in case of a book which is composed of many chapters and each chapter is composed of ordered pages, the structural metadata will indicate how these pages are ordered to form chapters.
3) Administrative metadata: provides information to curate and manage a resource. It includes several subsets such as: Rights management metadata for intellectual property rights for instance who can access the resource, and Preservation metadata that includes information which is required for archiving and preserving a resource. Another subset of the administrative metadata is the technical metadata which contains the necessary technical information for example, file formats and when and how it was created.

A. METS

METS is the Metadata Encoding and Transmission Standard expressed in XML [4]. In 2001 the METS XML schema was created sponsored by the Digital Library Federation (DLF) and supported by the Library of Congress as its maintenance agency; it is governed by the METS Editorial Board. METS defines the metadata which is necessary for the digital objects management within a repository and also for the objects exchange between repositories or between repositories and their users.

The METS root element contains seven major sections that the METS document structure consists of:

1) METS Header element `<metsHdr>`: Contains metadata that describe the METS document such as information about the creator of the document.
2) Descriptive Metadata Section `<dmdSec>`: Contains descriptive metadata about the METS object, or multiple descriptive metadata sections for each component of a METS object. METS provides means to link this descriptive metadata section to the other related metadata sections in the METS document of a specific object. Therefore, descriptive metadata can be expressed by using one of the many content standards (i.e., MARC, MODS, Dublin Core, etc.) or by producing a local XML schema. Consequently, This descriptive metadata can be external to the METS document by `<mdRef>`, internally embedded by `<mdWrap>`, or both.
3) Administrative Metadata Section `<amdSec>`: Is partitioned into four sub-sections/elements: `<techMD>` technical metadata about the component parts of the digital object, `<rightsMD>` intellectual property rights metadata which lists rights notices and use restrictions, `<sourceMD>` analog/digital source metadata which is about the source format or media of components that is used to generate the METS digital object, and `<digiprovMD>` digital provenance metadata which records the history of preservation-related actions that are done on the various files which contain a digital object. All of these elements use the same attributes. We can express administrative metadata by content standards as the: NISO Technical Metadata for Still Images, or MIX XML Schema. Secondly, each of Administrative Metadatas subsections can either be wrapped the metadata internally (by `<mdWrap>`) or be external to the METS document (by referencing an external location by `<mdRef>`) or both.
4) File Section `<fileSec>`: Includes a list of all files that contain the content of the electronic version of the digital object being described in the METS document. Moreover, it provides the location of each content file. An element of this section is a `<fileGrp>` element which can be used to group the files and then to provide subdivided related files by such criteria as file type, or object version. Within each `<fileGrp>` there is a `<file>` element for each file that comprises the encoded document.
5) Structural Map `<structMap>`: Outlines a hierarchical structure for the digital object. For example, in case of a book it outlines its hierarchical structure according to its chapters. Moreover, it links each structure's element to its related content files and metadata, which means it organizes the content of the digital object that is represented by the `<file>` elements in the `<fileSec>` into a coherent hierarchical structure. For example, in case of a book, if the `<fileSec>` of this digitized book includes different groups of images then the structMap will link each chapter of this book to its related content i.e. images/file from the fileSec. StructMap provided organization can be logical (such as a book divided into chapters), or physical (a book divided into sequences of pages), or a mixture of both.
6) Structural Link `<structLink>`: Is one of the elements of METS used to archive hypermedia such as Websites. StructLink records the existence of hyperlinks between nodes/components in the hierarchy outlined in the Structural Map. Thus, it mainly links each node in the logical structMap to its related nodes in the physical structMap. Consequently, in case of a digitised book, structLink links each chapter or section of this book in the logical structMap to its related pages in the physical structMap. To express a hyperlink that exists between two nodes in the structural map, structLink uses a `<smLink>` element.
7) Behavior Section `<behaviorSec>`: Is used to associate executable behaviors such as computer programming code with the digital content of the digitized object for many purposes such as for displaying the digital object or for transforming its content files.

This paper focuses on the descriptive metadata section `<dmdSec>`, and on the structural map section `<structMap>`. These are the two sections that are required to be converted from METS and MODS to JSON in order to make the resource discovery more efficient.

B. MODS

In 2002, Metadata Object Description Schema (MODS) [5] was developed by The Library of Congress' Network Development and MARC Standards Office. In ZVDD metadata, MODS is used as an extension schema to METS. ZVDD's METS metadata file uses MODS in the descriptive metadata in `<dmdSec>`. Examples of MODS's elements that are used to describe a resource is the titleInfo element which has subelements title (includes the title of the resource), name (includes the author's name) and typeOfResource (includes the text type). Therefore, this paper also considers how to convert MODS to JSON since the resource discovery process needs the descriptive metadata section in `<dmdSec>`.

C. *JSON*

JSON (JavaScript Object Notation [6]) is a lightweight text-based data-interchange format and very easy for humans to read and write and for machines to parse and generate.

JSON is built on two structures:

1) JSON objects: a collection of name/value pairs that are enclosed within curly braces ({})
2) JSON arrays: an ordered list of values that are enclosed within square brackets ([])

Data can be represented as one of the simple value types: strings, numbers, booleans and null.

IV. TRANSFORMATION

We built the JSON structure based on the rules of the metadata standard i.e. METS and MODS. For example, when an element in METS is repeatable, it is represented in JSON as an array of objects. Moreover, in the ZVDD use case there are many data providers who use different software to generate the metadata, so we had to consider different representations and semantics of the metadata.

We adapted the conversion patterns between XML and JSON of [7] as well as the "friendly" (well readable and processable) approach of conversion of [8] that leads to a flat JSON structure. This means that XML element names become JSON object names; XML children elements become JSON objects fields (that is, a collection of name/value pairs) or arrays (if the child elements are repeatable); and XML text nodes become JSON simple values. Notably, the conversion need not be round-trippable, because the original XML files remain to support the legacy systems.

For the purpose of the transformation of the considered METS & MODS metadata fields to JSON we use an XSLT (XSL Transformations) script, which is then run in batch by the Saxon parser [9]. XML transformation with XSLT gives the ability to add/remove elements and attributes to or from the output file, and the ability to rearrange and sort elements. For example, XSLT helps us to select the elements or sections out of the seven major sections that METS document consists of to convert to JSON and remove all other elements from the output file; we just select the two elements `<dmdSec>` and `<structMap>` (where `type=logical`) to be in the output file in JSON format. As another example, we remove or hide some elements from the XML document which are not be included in the output JSON file (such as `language` element which has subelement `languageTerm` which again has attributes `type` and `authority`), because they are not needed for the search. Moreover, XSLT helps to make the elements in JSON searchable by CouchDB by type conversions. For example, `originInfo` element has subelements about dates such as `dateIssued` and `dateCreated`; sometimes the date value is not a number but a string so we developed a function `<xsl:function name="json:dates_2_number">` to convert the date to a number. In case the date value is a string, this function extracts the four digits of the year out of the date string and puts the value in an element in JSON, that makes the date element searchable in CouchDB when we search by the year of publication.

Most importantly, this transformation saves storage space: the entire metadata collection of 1.244.507 files in JSON format occupies only 6.3 GB, whereas in METS format their size is 90 GB. Rather than having the METS XML metadata in the file system which is hard to manage, by converting them to JSON we have the ability to store them in CouchDB which makes their management much easier.

V. COUCHDB

Apache CouchDB is a database management system that manages multiple collections of JSON documents that are easily accessed via a RESTful HTTP API and queried by views which are MapReduce JavaScript functions. B-tree is the data structure that is used in CouchDB for database file storage, document indexes and view indexes. In the B-tree data structure, the data of documents and views is sorted and queried by a key. SQL-experienced users are usually not familiar with all of these features. Nevertheless we believe that CouchDB and its data structure B-tree are a good fit for our project goal to improve managing and querying a big collection of digitized objects.

A. *Document IDs*

Each document in CouchDB has a unique ID per database which is the key for sorting the documents. This unique ID has the same role as the primary key in the relational database system. In general, the user is free to choose any string to be the ID of a JSON document. Yet, the choice of ID has a significant impact on the layout of the B-tree [10], [11]. For example, using a sequential ID algorithm while uploading a large batch of documents will avoid the need of rewriting many intermediate B-tree nodes. A random ID algorithm may require rewriting intermediate nodes on a regular basis, resulting in significantly decreased throughput and wasted disk space due to the append-only B-tree design. In the case of ZVDD metadata files, the value of MODS element `<recordIdentifier>` is unique. Therefore, for ZVDD metadata, we decide that the JSON document ID should be assigned the value of `<recordIdentifier>` which is the same as the XML metadata file name which is also used to be the JSON file name.

B. *Bulk loading*

We use a simple yet efficient approach to upload a bulk of thousands of JSON documents to CouchDB which is by using this bash script as in [12]:

```
#!/bin/bash
```

```
FILES=/filepath/*
for filepath in $FILES
do
jsonfilename=$(echo $filepath |
 sed -e 's/\/folderpath\///g')
docname=$(echo $jsonfilename |
 sed -e 's/.json//g')
url="http://IP:port/dbname/${docname}"
curl -X PUT "$url" -d @"$filepath"
done
```

To upload an entire folder to CouchDB, the bash script loops over the file in the folder. First, the for loop gets the JSON file name where `sed` replaces the file path to the file name. Secondly, it gets the `docname` which is the file name without the extension .json. Finally, it uploads the JSON file to CouchDB by the `curl` command.

In case the JSON file is valid and is uploaded to CouchDB, the server replies as follows:

```
{"ok":true,"id":"PPN3303600759",
 "rev":"dca1f87ea3b9ef986b48c8debbbbdfc6"}
```

While uploading JSON files to CouchDB, some JSON files were not uploaded to CouchDB and the server returned the following error message

```
{"error": "bad_ request",
 "reason" :"invalid_json"}
```

although these files were valid according to the Oxygen program (the program that was used to transform METS files to JSON). The problem was that JSON numbers must not have leading zeros, a case which however occurred in some of the METS files in the `order` element. Therefore, we had to change the XSLT code by removing the leading zeros from this element

```
<xsl:value-of select=
 "replace(./@order, '^0+', '')" />
```

and then re-transform the METS files to JSON.

C. *MapReduce*

CouchDB provides a Spidermonkey-based JavaScript view engine that supports creating ad hoc views [13]; views are defined by MapReduce functions in JavaScript which perform aggregations or joins on database documents. In CouchDB, queries are described as views and expressed using the MapReduce processing paradigm. This means each query is defined in term of two functions: first, a Map function is called on the stored documents in order to emit key-value pairs (by `emit(key, value)`); then (in this case optional) a Reduce function is called on the result (i.e. key and value pairs) and aggregates the values with the identical key component to a single value.

Moreover, there are two kinds of views in CouchDB: permanent and temporary views. [14] The permanent view's result is preserved in the database, which saves the time of recomputing the unsaved temporary view each time it gets queried. When a database is updated (e.g. new documents are added), there is no need to evaluate the views from scratch but only recompute the MapReduce function for the new or the updated documents since the last time the view was queried. This feature obviously turns out to be very important with respect to handling of large data volumes and provides efficient use of physical resources. The second feature is, the very efficient look-up operations when accessing data according to keys, by which the values inside these B-trees are organized and ordered. This efficient key look-up can be performed for the JSON documents as well as for the views which are all stored in the B-tree structure. In case of the view, the look-up key can be any data field in the document that is emitted by the `emit` function, whereas in the document it is the unique ID of the document. Once the views are prepared, they become available for access via HTTP REST API.

The querying of our considered library metadata is mainly one of two kinds. The first query or view type is based on the fields that appear in the public search interface of ZVDD [1]: here the user is able to search for all MODS elements in the `dmdSec` such as `mods:titleInfo`, `mods:name`, `mods:originInfo` and also for the type attributes in the `mets:div` in the `structMap`. For example, to view the documents of digital works by their year of publication (i.e. start date of any `mods:dateIssued`, `mods:dateCreated`, and `mods:dateOther`), a permanent view which executes once on each document is created and its map function emits the key as the year of publication and the value as the ID of the document. This map function is recomputed only for the new or updated documents (i.e. incremental update).

The second kind of views are about the internal analytical queries for the library staff. They are meant for analyzing the data to check for data errors. We created a particular view that is capable of verifying the presence of all the compulsory metadata fields during the integration process. The library staff can easily distinguish the invalid metadata file in case of a missing element that is required. For example, the library staff requires a view that represents the existence of these compulsory elements: document type (which is inside logical structMap), title (that is inside titleInfo element), and from relatedItem element they want to check existence of the record identifier.

Furthermore, the library staff requires views that give some statistics information about their metadata records. For example, a view that gives information on all the different struct types of their metadata (such as monograph, multivolume work with the numbers of the records with a corresponding type), and a view that gives a summary of languages of their collection and how many literature resources are written in each language. This view is a map function that gives key and value pairs where keys are the `<language>` element, then the reduce function uses the function `sum` to aggregate the emitted values with the identical key component into one value. For example, in our database, there are 309341 documents written in German language.

VI. Evaluation

According to the experiments we conducted on standard hardware on the considered collection of all the 1.2 million

METS documents (exactly 1.244.507 JSON files in CouchDB), our newly proposed approach really brings advantages. The view for the analytical purpose (verifying the presence of all the compulsory metadata fields during the integration process like a document type, title, record identifier etc.) as well as the query view that is based on a field of the search interface of ZVDD (e.g. viewing the documents by their titles) were both materialized into permanent views each in just 10 to 30 minutes on average for the entire collection. Querying these permanent views then takes only some milliseconds.

For example, the library staff need a view for analysis purposes that presents document type, title, record identifier elements. The time to run the view creation over the entire collection and to construct its B-tree took 26:45.809 minutes. Moreover, one example of a query on this view takes 117 ms:

```
/_view/analysisview?key=["ContainedWork",
"Patrick Kennedy's journal up the Illinois
 river"]
```

Furthermore, the views that give statistics information about the metadata are easy to implement by MapReduce functions. For example, we implemented the MapReduce function that gives the list of all the different languages that the digital works are written in as well as how many documents are written in each language. View creation takes 13:31.368 minutes to run over the entire 1.2 million records.

Hence, by this kind of view the metadata analysis becomes an easy process with CouchDB. Without CouchDB, this was a difficult process where library staff had to execute the analysis directly over the METS collection stored in file system. They had to transform the data to raw data which took days for transforming some metadata records. Moreover, the major problem with the tedious transformation method is that the records get flattened and they lose some information so if a compulsory element in the metadata record is missing, it was hard for the staff to figure out the cause of the error: is it an error from the data provider or is it lost during data transforming into raw data? Last but not least, the result of this transformation process was not as easy to be read as the view of CouchDB.

Moreover, this kind of view can help the library staff in analyzing their data, since it gives the list of the keys that are wrongly used. For example, `<language>` element of MODS which specifies the language of the literature resource is described in the record. The value of this element is expressed by language codes from ISO 639-2/b, such as `ger` for German language. The permanent view in CouchDB gives a list of all the language values that are used in all the documents in the metadata. Therefore, if there is any language value code used incorrectly, then the library staff can spot the error easily. They can use the view that lists the documents by the language key to find the documents with the erroneous language code (one such sample query only took 96 ms). Consequently, the user benefits from the shortness of the query execution time.

Lastly, the compact feature of CouchDB for the database and for the views saves storage space without negative effects. For example, the analysis view files before compaction require 5,7 GB storage, whereas after compacting the view, only 313 MB are occupied.

VII. Conclusion

Metadata is the "core of any information retrieval system" [3] – in this paper, the metadata is really the core for the ZVDD portal. Metadata has profound implications for any digital library, as described in [3], [15], [2]: it gives ability to deliver objects in a meaningful way, it is the key to ensure long-term ability to access and preserve the digital objects and it plays a critical role in ensuring authoritative, scalable, and interoperable cultural heritage information.

The main improvement of our approach lies in a fact that we are able to perform complicated queries and internal library analysis, which were previously nearly impossible to conduct, or at most manually with difficulties. Faster execution times were achieved primarily by choosing a modern database system, together with pruning of the original METS documents, from which we preserved only the descriptive and several other metadata fields that are actually required to evaluate all the relevant queries. Finally, JSON files are shorter and easy to read, understand and modify on the fly by ordinary users. Thus, big metadata files (1.2 million records) of ZVDD and Goettingen university library significantly benefited from this efficiency gain; they were managed with query latency of only some milliseconds and support for incremental updates.

Acknowledgment

The authors would like to thank Alexander Jahnke, Stefanie Rühle and Timo Schleier for their support.

References

[1] Zentrales Verzeichnis Digitalisierter Drucke, `http://www.zvdd.de/dms/esuche/`

[2] National Information Standards Organization, *Understanding Metadata*, NISO Press, 2004.

[3] Richard Gartner, *Metadata for Digital Libraries: State of the Art and Future Directions*, JISC: Bristol, UK, 2008.

[4] The Library of Congress, *Metadata Encoding and Transmission Standard: Primer and Reference Manual*, 2010.

[5] The Library of Congress, *Outline of Elements and Attributes in MODS Version 3.5*, 2013.

[6] Ecma International, *The JSON Data Interchange Format*, 2013.

[7] Stefan Goessner, *Converting Between XML and JSON*, O'Reilly XML.com, O'Reilly Media, Inc., 2006.

[8] John Boyer, Sandy Gao, Susan Malaika, Michael Maximilien, Rich Salz, and Jerome Simeon, *Experiences with JSON and XML Transformations*, Proc. of W3C Workshop on Data and Services Integration, Bedford, MA, USA, IBM, 20-21 Oct. 2011.

[9] SAXON The XSLT and XQuery Processor, `http://saxon.sourceforge.net/`

[10] J. Chris Anderson, Jan Lehnardt, and Noah Slater. *CouchDB: The Definitive Guide* 1st ed., O'Reilly Media, 2010.

[11] Apache CouchDB Documentation, The Apache Software Foundation, `http://docs.couchdb.org/`

[12] Michael Lenahan, *Document-Oriented Persistence with CouchDB*, Project Report, Birkbeck, University of London, School of Computer Science and Information Systems, 2010.

[13] Joe Lennon, *Beginning CouchDB*, Apress, Berkeley, CA, 2009.

[14] CouchDB Wiki, *Introduction to CouchDB Views*, `https://wiki.apache.org/couchdb/Introduction_to_CouchDB_views`

[15] Tony Gill, Anne J. Gilliland, Maureen Whalen, and Mary S. Woodley, *Introduction to Metadata*, Getty Publications, 2008.