DIGITARQ - CREATING AND MANAGING A DIGITAL ARCHIVE

JOSÉ CARLOS RAMALHO[1]; MIGUEL FERREIRA[2];

[1] Informatics Department, University of Minho,
Campus Gualtar 4710 Braga – Portugal
jcr@di.uminho.pt
[2] Information Systems Department, University of Minho,
Campus Azurém 4800 Guimarães – Portugal
mferreira@dsi.uminho.pt

In this paper we present the steps followed in a project called DigitArq that aimed at building a centralised repository for archival finding aids. At the O'Porto's District Archive, finding aids existed in several different formats and media. Migration was used to convert all the finding aids into a single normalised format based on an international standard – EAD/XML. After migration, archival management software was developed to maintain the collected information and assist archivists in the creation of new finding aids. Archival finding aids are described by hierarchical structures which can easily be described with XML but present interesting issues while using Relational Databases. The relational data model employed is described in detail.

Keywords: Digital Archives; Migration; EAD; XML; EAD; Relational Database; Middleware

1. INTRODUCTION

Those who have stood patiently at the reference service of a Historical Archive understand the great difficulties in finding specific items of information. Archival work aims at solving this problem by generating multiple indexes, lists, inventories, catalogues and transference guides; although these finding aids help users and archivists attain the artefacts they seek, they comprise a heterogeneous universe extremely hard to manage. The lack of coherence between most finding aids makes updating information a nightmare. To put an end to this scenario, it was carried out in the O'Porto's District Archive a project called DigitArq, whose major goal was to serve as a first attempt at the edification of a Digital Archive.

This project comprised several stages. The first one consisted in the conversion of a series of documents and databases containing different types of finding aids to a common digital format based on an international standard.

The second stage of the project aimed at constructing a centralised repository to store all the collected material and developing archival management software to maintain all information.

A third stage consisted in the development of a search engine that allowed Internet users to find and browse the collections.

ARCHIVAL DESCRIPTION

The purpose of archival description is to identify and explain the context and content of archival material in order to promote its accessibility. This is achieved by creating accurate and appropriate representations and by organizing them in accordance to predetermined models [1].

Description-related processes may begin at, or before, records creation and continue throughout the life of the records. These processes make it possible to institute the intellectual

controls necessary for reliable, authentic, meaningful and accessible descriptive records to be carried forward through time [1].

The information produced in an Archive is organised hierarchically, providing a top-down, general to specific, description of a collection of items. A book, for example, is not described alone. A description of the organisation that produced the book (also referred to as *fonds*) is included, as well as the description of all the physical and logical sections that define that organisation. Similar documents are usually grouped into Series.

We can think of an archival finding aid, as a tree of isolated descriptions were the root element represents the organisation, person or family that produced the document, and the leaf nodes hold the description of documents that cannot be divided any further. Between the root and leaf nodes, information that represents the physical structure of

the organisation and the logical groupings of documents is provided.

3 ENCODED ARCHIVAL DESCRIPTION (EAD)

One of the primary goals of this project set upon the centralisation of the scattered finding aids in a manageable information system and the normalisation of these finding aids by means of description standards.

Another great requisite was the ability to exchange information with other national and international Archives. To fulfil that need, the Encoded Archival Description (EAD) [2] standard was chosen. EAD is a XML standard for encoding archival finding aids. It is based on the General International Standard Archival Description [1], a standard recommended by the International Council on Archives [3] for describing finding aids.

EAD is becoming a standard *de facto* in the archival scene, mostly because of its ability to provide self-explanatory descriptions of entire collections.

4 PROJECT DESCRIPTION

4.1 MIGRATION

A large quantity of information inside the O'Porto's District Archive existed solely on paper. It consisted mainly on typewritten documents with no digital equivalent that were produced at the Archive in its early years.

The information that is commonly generated in an archive is what we call metadata; that is, data about information records or artefacts. There is a big community working with standards to describe metadata: Dublin Core, MPEG-7, EAD , LOM, etc [2, 4-6]. In the context of this project EAD was the obvious choice since a large community of Archives has already adopted it

for describing their collections and because it is based on XML which greatly simplifies the interchange of information.

The paper-based documents found at the Archive were digitalised and their content was extracted by means of optical character recognition (OCR). Further on, specialised archivists were recruited to correct all the errors introduced by the OCR software and to annotate (mark-up) the resulting text. The introduced tags enriched the text with semantics making it easy for computer manipulation. Upon that, several conversion scripts where developed and applied to the annotated text resulting in hundreds of EAD/XML files.

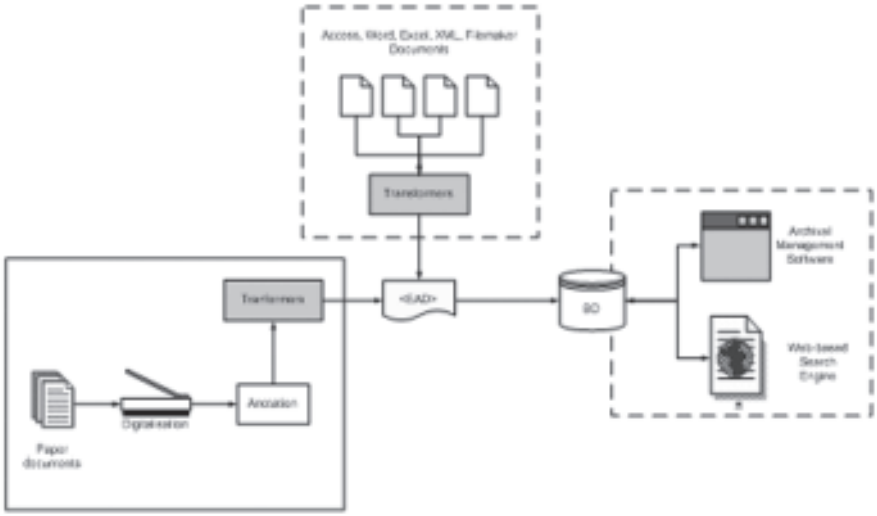

FIGURE 1 - DIFFERENT STAGES OF THE PROJECT.

Besides paper-based documents, many digital files and databases existed at the Archive. To name some examples, digital material was found in Word, Excel and XML files. Relational databases made in Access and Filemaker were found as well.

The biggest challenge though, was the conversion of two CDS/ISIS hierarchical databases. The fields' structure in those databases was not fully compatible with the EAD standard, so some field multiplexing had to be performed; this means that information that was stored in a single field had to be separated and assigned to several, more specific, ones. The field separation was based on natural language patterns existing in the description text.

The conversion strategy for the remaining databases was fairly similar to the one applied to the paper-based material. Recognition and annotation phases were suppressed thought, since the documents already existed in some sort of digital form and the majority of distinct information was already separated by some kind of text marker (e.g. tab or comma separator).

4.2 ARCHIVAL MANAGEMENT SOFTWARE

As a result of migration, several hundred XML files based of the EAD schema were produced. Software capable of storing, managing and retrieving this information was urgently needed.

After consulting with the group of archivists responsible for describing archival material, a list of software requirements was compiled:

Relative referencing

In order to prevent human mistakes, the units' reference should be handled in a relative manner. It is common in archival management software to identify items with an absolute node path reference that both identifies the item and positions it in the tree of description. It was very common for archivists to provide incorrect references to items. Because the reference was used to position the item in the tree, by changing a single letter could move it to an erroneous branch of the tree or even to a completely different collection.

Hierarchy preservation

Hierarchy could be corrupted if constrains in the software were not included. There is an order by which levels of description can be organised, e.g., a subseries must be child of a series and never the other way around. That would violate the rules of description.

Detection of description errors

While archivists are describing fonds, errors and omissions are quite frequent. The software should be able to generate a list of errors and warnings upon user request. The software was able to point out missing dates, swapped initial/final dates, empty mandatory fields, etc.

Inference from lower levels

In archival finding aids, special information can be automatically calculated from lower levels of description. For example, each level (or node) of description comprises two dates corresponding to the earliest and latest date of its child records. For a particular node, this information can be calculated by traversing all its child sub-trees.

Drag and Drop

Drag and drop should be used to move records within the description tree. This eliminates the chances of reference inconsistencies, since references are automatically updated by the system with no human intervention.

Cooperative work

The software should be able to allow different users to work with a collection at the same time. A centralised repository was, therefore, required.

4.3 WEB-BASED SEARCH ENGINE

After migration, a huge amount of finding aids was collected and stored in a centralised repository. The next logical step was bringing all that information to the Web. A web-based search engine was developed in order to allow local and remote users to find and browse the Archive's collections.

A large percentage of users that access the Archive's web-site live abroad, especially in countries such as Brazil, France, Switzerland and United Kingdom. Allowing users to browse the Archive's collection remotely, dramatically improves the quality of the service provided.

5 THE DATA MODEL

After migration, a centralised repository was needed to maintain the resulting EAD/XML finding aids. EAD it self gave us the first clues on how to build a data model capable of holding all that information.

Hundreds of thousands of records resulted from the migration phase and this number was

expected to grow over a million in the foreseeable future. Due to the amount of data and the need for a management system we had to start thinking about databases and EAD raises an interesting problem since its nature is hierarchical.

A very important requirement of the system was that EAD/XML files could be easily imported and exported from the repository for sharing purposes.

To implement the repository we opt for a Relational Database. Our choice was based on two basic constrains:

Budget

Due to budget limitations, the purchase of a native-XML database was impossible.

Time

The project had a time constrain of nine months and a team limited to two programmers (migration, management software, repository and search engine) and two archivists (for text annotation and error correction). Given the team previous experience with Relational Databases, it seamed reasonable to find a solution in that domain.

In order to ingest our XML files into a Relational Database, a middleware solution based on an abstract class called LazyNode was used. LazyNode class defined a comprehensive set of methods that allow the programmer to fully manipulate the hierarchical structure of finding aids:

TABLE 1- LAZYNODE ABSTRACT CLASS METHOD DEFINITION.

Method signature	*Description*
Sub Upload	Updates database with the information stored in memory.
Sub Download	Brings information from the database to the application level. This accounts for the refreshing of information as well.
Function Children() As LazyNodeCollection	Returns a collection of child nodes.
Sub RemoveChild(ByVal child As LazyNode)	Removes a child node.
Sub AppendChild(ByVal child As LazyNode)	Appends a new child node.
Function HasChildren() As Boolean	This method returns True if the selected node contains child nodes, and False otherwise.
Function CreateNode() As LazyNode	Creates an instance of a node. This instance should be appended to a node.
Function Clone() As LazyNode	Clones the selected node.
Function Parent() As LazyNode	Returns the parent node or *null* if the selected node is a root node.

Apart from the described methods, *sets* and *gets* were implemented for each field/property that comprised the finding aids' nodes (e.g., Unit Title, Unit Reference, etc.).

The abstract class was called LazyNode because the information was loaded from the database to the application level only when required. By doing it so, the application was not overload with unnecessary information and network traffic was severely reduced.

Two implementations of the LazyNode abstraction were developed: EADLazyNode and SQLLazyNode. The first one is responsible for manipulating EAD/XML files, and last one handles

communication with Relational Databases.

Each of these classes implements the nine methods described in the LazyNode specification.

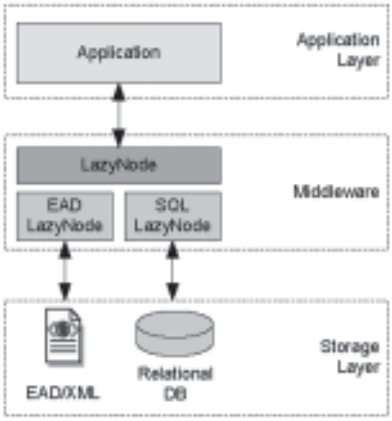

FIGURE 2 - LAZYNODE ABSTRACTION LAYER.

6.1 EADLAZYNODE

EADLazyNode class inherits abstract class LazyNode and is responsible for handling EAD/XML files.

In order to implement the mandatory nine methods over XML files the DOM [7] was used. Reading values was easily accomplished by performing XPath queries. Writing was equally straightforward although special attention had to be given to missing elements. In this situation, the path had to be constructed before the element value could be assigned.

6.2 SQLLAZYNODE

SQLLazyNode class was responsible for handling communication between the application and Relational Databases.

The relational schema capable of holding the EAD description tree was obtained by applying the following set of rules [8, 9]:

1. FOR EACH NODE FIELD, A COLUMN WAS CREATED.

2. ALL OPTIONAL FIELDS WERE SET TO ALLOW *NULL* VALUES.

3. ADDITIONAL COLUMNS WERE INCLUDED TO MANAGE THE HIERARCHY RELATIONSHIPS: ID, PARENTID AND HASCHILDREN.

4. THE DESCRIPTION TREE WAS MAINTAINED BY A CIRCULAR RELATIONSHIP BETWEEN FIELDS PARENTID AND ID. IF A PARTICULAR RECORD IS ROOT NODE THEN PARENTID IS SET TO *NULL*.

It should be noticed that all levels of description (e.g. fonds, subseries, document, etc.) do not comprise the exact same set of information (although, the majority of fields is common to all levels). Specific level fields are considered optional to other levels therefore a *null* value is assigned on those cases.

The HasChildren (boolean type) column was included to optimise traversals. By keeping this value updated, the database is not overloaded with heavy queries.

The inherited methods defined by LazyNode abstract class, can easily be implemented with standard SQL queries:

TABLE 2 - SQL IMPLEMENTATION OF THE INHERITED METHODS.

Method	*SQL implementation*
Sub Upload	UPDATE Components SET {fields to be updated}={value} WHERE ID={node id}
Sub Download	SELECT * FROM Components WHERE id={node id}
Function Children() As LazyNodeCollection	SELECT id FROM Components WHERE ParentId={node id}
Sub RemoveChild(ByVal child As LazyNode)	For Each Node In Child.Children RemoveChild(Node) Next DELETE FROM Components WHERE id = {Child.Id}
Sub AppendChild(ByVal child As LazyNode)	UPDATE Components SET ParentID={node id} WHERE ID={child id}
Function HasChildren() As Boolean	SELECT HasChildren FROM Components WHERE ID = {node id}
Function CreateNode() As LazyNode	INSERT INTO Components ([field names]) VALUES ([default values])
Function Clone() As LazyNode	NewNode.ImportFields(Me) NewNode.Upload() For Each Child In Children() NewNode.AppendChild(Child.Clone) Next
Function Parent() As LazyNode	SELECT ParentID FROM Components WHERE Id={node id}

7 ADVANTAGES AND DISADVANTAGES OF THE DATA MODEL

The presented model is easy to understand, simple to implement and provides a reasonable set of advantages to the programmer. First of all, it is transparent in what concerns accessing different storage types/formats. From the application point of view, the description trees (or finding aids) can the stored in any format provided that the nine abstract methods are correctly implemented.

Transparency allows the development of *catamorphisms* over different storage types. This is a strongly welcome property since most operations performed automatically by the description management software consist of entire fonds traversals, e.g. automatic revision of fonds descriptions, inference of values from leaf to root nodes, construction of description summaries and reports, unit reference normalisation (to fix residual migration errors).

Another important advantage of this model is capacity to easily convert between different storage formats. The conversion is accomplished by traversing the source description tree, recreating it in an empty destination tree of a different format.

The data model also accounts for platform independence. The technologies involved in the implementation of the model consist solely in standard SQL and DOM - two technologies available in most computing platforms today.

Although the model is simple to use and appropriate to the requirements of our project, it carries some disadvantages. The information is hierarchical not relational so in order to perform

some basic tasks, like finding a specific node, a full tree traversal must be performed by the application. On a relational database this would be accomplished with a simple SQL query. The abstraction class only provides a tree-based perception of the stored information; therefore it is impossible to have direct access to a node. In order to reach a leaf node, all nodes from root to leaf must be downloaded into the application.

Full traversal operations (e.g. *fonds* revision) can take up to 1,4 minutes for the average-sized tree (around 1000 nodes). Although performance was not astounding, consulting with users showed us that it was not a major problem since this kind of operations were only performed once or twice a day.

CONCLUSIONS AND FUTURE WORK

In this paper we present briefly all stages that comprised a project called DigitArq that was carried out at the O'Porto's District Archive. Special attention was given to a middleware solution capable of storing hierarchical information in relational databases. The proposed solution grants access transparency to different types of storage. The conversion between data models can be accomplished by a simple traversal of the source tree.

Catamorphisms can easily be implemented over any storage format since a common unique interface is exposed to the application layer. Implementations for EAD/XML files and Relational Databases were developed.

Because the model is general-purpose it doesn't take advantage of any special features provided by relational databases. In this context, future work could be developed to optimise the implementation over this type of storage. The use of *stored procedures* could be a good starting point in this direction.

Faster loading and network traffic reduction could be accomplished by adding cache and pre-loading mechanisms to the middleware. The implementation of these methods would require additional control data to certify that cached information is coherent at all time.

9 REFERENCES

1. ICA, *ISAD(G): General International Standard Archival Description, Second edition*, in *ICA Standards*. 1999, International Council on Archives.
2. LC, *EAD - Encoded Archival Description*, Library of Concress.
3. ICA, *International Council on Archives*. 2004.
4. ISO, *Learning Object Metadata*.
5. MPEG7, *MPEG-7 Description Definition Language*.
6. DCMI, *Dublin Core Metadata Initiative*.
7. W3C, *World Wide Web Consortium*. 2004.
8. Lu, S., et al., *A new inlining algorithm for mapping XML DTDs to relational schemas*, Wayne State University - Department of Computer Science.
9. Bourret, R., *Mapping DTDs to Databases*.