From Users to Systems: Identifying and Overcoming Barriers to Efficiently Access Archival Data

Nicola Ferro
Department of Information Engineering
University of Padua
Padua, Italy
nicola.ferro@unipd.it

Gianmaria Silvello
Department of Information Engineering
University of Padua
Padua, Italy
gianmaria.silvello@unipd.it

ABSTRACT

Digital archives are one of the pillars of our cultural heritage and they are increasingly opening up to end-users by focusing on accessibility of their resources. Moreover, digital archives are complex and distributed systems where interoperability plays a central role and efficient access and exchange of resources is a challenge.

In this paper, we investigate user and interoperability requirements in the archival realm and we discuss how next generation archival systems should operate a paradigm shift bringing a new model of access to archival resources which allows to better address these needs.

To this end, we employ the data structures and query primitives based on the NEsted SeTs for Object hieRarchies (NESTOR) model to efficiently access archival data overcoming the identified barriers and limitations.

Keywords

set-based data models, archival data, XPath, XML

1. INTRODUCTION

Archives, along with libraries and museums, are one of the main cultural institutions encompassed by Digital Libraries (DL). Archives represent the trace of the activities of a physical or legal person in the course of their business which is preserved because of their continued value over time. They are composed of unique documents interlinked with each other as well as with their production and preservation environments. The main characteristic of archives lies in the *hierarchical structure* used to retain the *context* and the full informational power of archival data.

The hierarchical structure shaping archives is a foundational feature of traditional paper-based archival description – the so-called *finding aid*. This is reflected in its digital counterpart, the Encoded Archival Description (EAD) [14] eXtensible Markup Language (XML) format, which is the key brick for managing, finding and accessing archival data.

Over the last decade, thanks to the centrality of the Web for information access and the rapid evolution of DL services, we have witnessed a major shift towards a "*radical user orientation*" [12] of archives, where usability and findability of resources are becoming number one priorities [20] given the "dramatic increase" [3] in the number of people accessing them. A recent user study [11] analyzing the user interaction patterns with finding aids highlighted that "[they] *focus on rules for description rather than on facilitating access to and use of the materials they list and describe*" and that many archive's users have serious issues using finding aids [1]. Common and frequent user interaction patterns with finding aids are navigational and thus they require to browse the archival hierarchy to make sense of the archival data; for instance, two common interaction patterns are [11]: *top-down* where users "*start at the highest level, gain background and context, and work down to the most specific level of detail*" and *bottom-up* where users "*start at the most detailed level seeking specific information, and then move back to the higher levels*".

From this new point-of-view, digital finding aids (i.e. EAD) constrain user orientation of archives because several key operations are not possible nor efficient, given that it is problematic to: (i) let the user access a specific item on-the-fly, whereas we have to define fixed access points to the archival hierarchy [8]; (ii) let the user reconstruct the context of an item without requiring to browse the whole archival hierarchy [2]; and, (iii) present the user with only selected items from an archive, whereas we have to give them the archive as a whole [7, 18].

From the technological perspective, the presented limitations also affect the interoperability of archives in distributed environments, thus preventing the exchange of resources by means of standard DL technologies such as the Open Archives Initiative Protocol for Metadata Harvesting (OAI-PMH)[1] [8, 15]. Indeed, a single EAD file describes a whole archive and thus it is not possible to share or exchange in a distributed environment only a subset of records; for archives, it is common to be required to exchange only the high-level descriptions (e.g., fonds and sub-fonds) or to exchange only the records open to public disclosure. This problem affects the possibility to exchange finding aids with variable granularity by means of OAI-PMH forcing archival institutions to share whole archives or nothing. EAD provides archivists with many degrees of freedom in tagging practice exacerbating the differences in how XML elements are used and nested one inside the other [10]. This makes it difficult to know in advance how an institution will use the hierarchical elements and then to define general rules and paths to access EAD elements; for instance, there is no guarantee that an XML Path Language (XPath) expression returning all the series or the units in a given EAD file will

In Proceedings of 1st International Workshop on Accessing Cultural Heritage at Scale (ACHS'16), June 22, 2016, Newark, NJ, USA.

[1] http://www.openarchives.org/pmh/

work with a different file in another collection or even in the same one.

In this paper, we stem from the above observations about the user and interoperability needs in the archival realm to discuss how next generation archival systems should operate a paradigm shift bringing a new model of access to archival resources which allows to better address these needs. In particular, the contribution of the paper is to turn the above requirements into specific access use cases to archival resources, discussing how and why current approaches represent a barrier to their complete fulfillment, and showing how our proposed solution, called NEsted SeTs for Object hieRarchies (NESTOR) [8, 9], represents a step forward.

Indeed, NESTOR [8] defines an alternative way to represent hierarchical data by expressing the relationships between objects through the inclusion property between sets, in contrast to the binary relation between nodes exploited by the tree which is the typical model used to represent archival data. NESTOR has been instantiated by three data structures on which query primitives, proven to be highly efficient in a wide spectrum of cases, have been realized [9]. NESTOR represents a paradigm shift with respect to state-of-the-art solution to access hierarchical data because it answers query primitives – e.g., descendants and children to deal with the top-down interaction pattern and ancestors and parent to deal with the bottom-up one – by exploiting basic set operations which do not require to browse and navigate the hierarchy.

Moreover, in order to fully understand the difference between NESTOR and state-of-the-art navigational (i.e., based on XPath) approaches, we conducted a case study evaluation based on ten real-world heterogeneous EAD files representing different key challenges for the identified access use cases, where we discuss the main drawbacks of a navigation-based access approach and how they are addressed by the NESTOR set-based one. We also show how the intrinsic differences between NESTOR and traditional navigational approaches are also consistently reflected in the query execution times, which are a quantitative proxy for appreciating the paradigm shift represented by NESTOR and its impact.

The rest of the paper is organized as follows: Section 2 provides relevant background information; Section 3 discusses the examined use cases; Section 4 presents the experimental outcomes. Finally, Section 5 draws some conclusions.

2. BACKGROUND

2.1 Digital Archives

Archives are composed by "*unique records of corporate bodies and the papers of individuals and families*" [14]. The *original order* – i.e. the principle of provenance – of the documents within an archive is preserved because the context and the physical order in which the documents are held are as valuable as their content [6].

According to the *International Standard for Archival Description (General)* (ISAD(G)), archival description (i.e. the finding aids) proceeds from general to specific as a consequence of the provenance principle and has to show, for every unit of description, its relationships and links with other units and to the general fonds, taking the form of a tree as shown in Figure 1 on the left. The digital encoding of ISAD(G) is the Encoded Archival Description (EAD) [14],

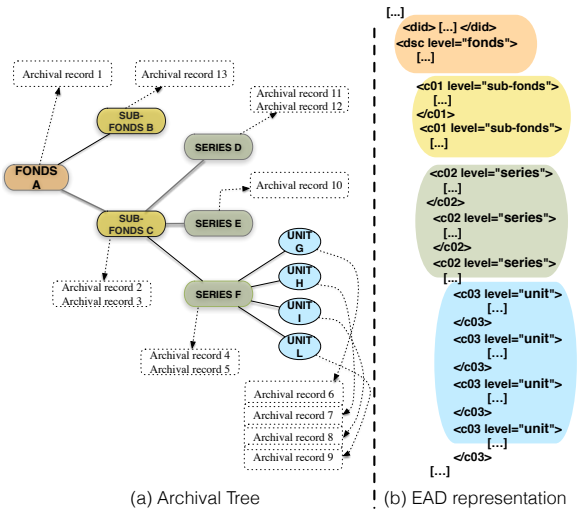


Figure 1: A sample archive and its EAD representation.

shown in Figure 1 on the right, which is an XML description of a whole archive, reflects the archival structure, holds relations between entities and retains context.

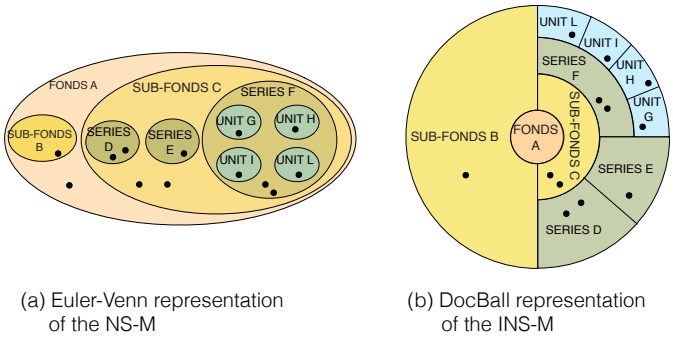
EAD follows the traditional archival paradigm where experts know exactly what they are looking for and, for example, they browse EAD to know the location of physical records [12]. By contrast, in the new user-oriented paradigm enabled by digital archives "users no longer have to be dependent on the physical presence of archivists to identify, review, and retrieve materials" [23], but they need effective means for performing information seeking activities. As a matter of fact, EAD turns out to be problematic in: (i) supporting user-oriented information access; (ii) supporting flexible control access policies; (iii) enabling interoperability between digital archives working in distributed environments.

2.2 XPath: A Navigational Approach

XPath[2] is widely adopted for searching and selecting portions of EAD files. XPath is a language for addressing parts of an XML document; it provides basic facilities for manipulation of several data types and adopts a path notation for navigating through the hierarchical structure of an XML document. "Location path" is a common kind of XPath expression, which selects a set of nodes relative to a given node and as output returns the node-set containing the nodes selected by the location path. Each part of an XPath expression can be composed of three parts: (i) an axis, which specifies the tree relationship between the nodes; (ii) a node test, which specifies the node type and expanded-name of the selected nodes; and (iii) zero or more predicates that can further refine the selected set of nodes.

As it emerges from the previous discussion, archival systems typically rely on third-party and standard libraries for XPath processing. Since the NESTOR data structures and query primitives are implemented in Java and work in-memory, we are interested in comparing to state-of-the-art

[2]http://www.w3.org/TR/xpath/



(a) Euler-Venn representation of the NS-M

(b) DocBall representation of the INS-M

Figure 2: The archive of Figure 1 modeled with the NS-M and the INS-M.

in-memory Java-based solutions. Xalan[3], Jaxen[4] and JXpath[5] are the three most used state-of-the-art Java libraries for XPath processing.

2.3 NESTOR: A Set-Based Approach

The NESTOR model is defined by two set-based data models: The Nested Set Model (NS-M) and the Inverse Set Data Model (INS-M) [8]; they are formally defined in the context of set theory as a collection of subsets. The most intuitive way to understand how these models work is to relate them to the archival tree. In Figure 2a we can see how the archive shown in Figure 1 is mapped into an organization of nested sets based on the NS-M.

From Figure 2a we can see that the NS-M adopts a bottom-up approach: (i) each set corresponds to an archival division; (ii) the innermost sets are the leaves of the hierarchy, e.g. the units; (iii) you create supersets as you climb up the hierarchy, e.g. the series, sub-fonds and fonds. The archival records are represented as elements belonging to the sets. With the NS-M an archive is modeled as a collection of subsets where there is a set – i.e. "fonds" – which contains all the subsets – i.e. "subfonds", "series", "units" – of the archive and where two subsets at the same level – e.g. two "series" – cannot have common elements, thus their intersection is empty.

As shown in Figure 2b, the INS-M adopts a top-down approach: (i) each set corresponds to an archival division; (ii) the innermost set is the root of the hierarchy, i.e. the fonds; (iii) you create supersets as you climb down the hierarchy, e.g. sub-fonds, series and then units. As for the NS-M, also in this case the archival records are represented as elements belonging to the sets. With the INS-M an archive is modeled as a collection of sets where there exists an archival division shared by all other divisions; in our example, the "fonds" is the archival division common to all the other divisions in the archive.

This vision overcomes EAD limitations because in NESTOR each archival record is an element belonging to a set which can be selected and managed independently from the other records; thus, we can return to the users a list of records belonging to different archival divisions at any level allowing them to access and consult the records hiding the complexity of the whole archival structure.

[3]http://xml.apache.org/xalan-j/
[4]http://jaxen.codehaus.org/
[5]http://commons.apache.org/proper/commons-jxpath/

NESTOR can be instantiated by three data structures [9]: Direct Data Structure (DDS), Inverse Data Structure (IDS) and Hybrid Data Structure (HDS). Each one of these structures is composed by three dictionaries, one containing the materialization of the sets, one containing the direct subsets of each set and the last one containing all the supersets of each set. DDS is a structure built around the constraints defined by the NS-M, IDS is a structure built around the constraints of INS-M and HDS can be seen as a mixture between DDS and IDS [9].

When we deal with a collection of sets defined by NESTOR, we can distinguish between *set-wise* and *element-wise* primitives. The former ones enable us to query the structure of an archive, whereas the latter ones query the content of the archive (i.e., the archival records). For instance, by means of the set-wise primitives we can ask for all the series of a specific sub-fonds, whereas with the element-wise primitives we can ask for all the archival records belonging to the series of that sub-fonds.

NESTOR primitives (i.e., Descendants, Ancestors, Children and Parent) are efficient alternative implementations of XPath primitives as shown in [9] where we conducted an extensive evaluation on five EAD collections, Wikipedia and two synthetic XML datasets and we compared NESTOR with state-of-the-art XPath engines. In [9] we evaluated NESTOR on average performances by testing the primitives on thousands of files and then presenting mean execution times; in this paper we investigate how NESTOR primitives behave with specific digital archives and how efficiently they answer to common and frequent archival operations.

3. USE CASES

We present three user-oriented use cases derived from common interaction patterns individuated in the archival domain and four interoperability use cases based on the exchange of archival data in distributed environments.

3.1 User-oriented Use Cases

Use Case 1: identifying and selecting relevant material

This use-case is related to the "*searching for known material*" information seeking activity investigated by Duff and Johnson in [5]. This activity may be performed by researchers at the beginning of a project to establish a context and detect relevant information and it may be re-iterated several times to "*reevaluate information that has suddenly gained new significance*" [5]. Such activities can be associated to the top-down pattern of interaction identified by Freund and Toms in [11] where the users "*start at the highest level* [of an archival description]*, gain background and context, and work down to the most specific level of detail*".

In Figure 3 we can see a graphical representation of this use case. We consider an archival system that answers a user query that starting from a given context node requires to return a list of archival records. From this list the user then selects the description of, say, sub-fonds C; in this case two frequent queries to be answered are: to return the subdivisions (series D, series E, series F, unit G, unit H, unit I and unit L) which are part of this sub-fonds – i.e a structural query – and to return all the records (the actual records or their descriptions contained by the three series and four units which are children of sub-fonds C) associated to this sub-fonds – i.e a content query.

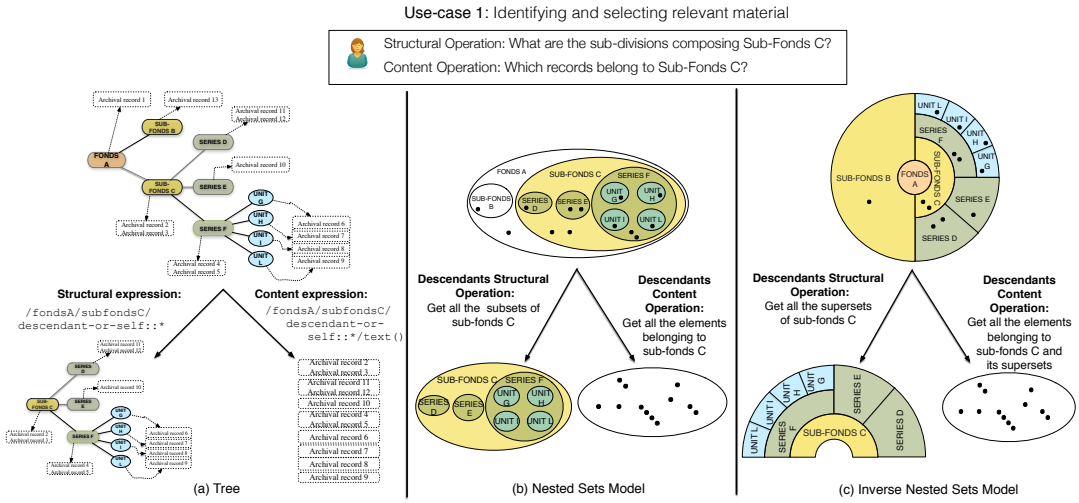


Figure 3: Use-case 1: Identifying and selecting relevant material.

With a navigational approach based on XPath, the structural query corresponds to the following XPath expression: `/fondsA/subfondsC/descendant-or-self::*`; and the content query corresponds to: `/fondsA/subfondsC/descendant-or-self::*/text()`. Both these expressions require to navigate the archival tree to the sub-fonds C division and then to visit all of its descendants.

In Figure 3 we see that the NS-M answers the structural query by returning all the subsets of sub-fonds C (i.e. all its descendants), whereas the INS-M answers it by returning all the supersets of the sub-fonds (i.e. all its ancestors). The content query is answered by NS-M by returning all the elements belonging to sub-fonds C, whereas INS-M has to return the union of all the elements belonging to sub-fonds C and its supersets. We can see that the NS-M and the INS-M answer the queries by exploiting two different primitives, the first is based on the *subsets* of a set, whereas the second is based on its *supersets*. In NS-M the descendants of an archival node, say sub-fonds C, are the subsets of the set representing sub-fonds C; whereas, in INS-M the descendants are the supersets of the given set.

Use Case 2: building contextual knowledge

"*Building context is the sine qua non of historical research*" [5] and one of the main functions of archives. As we described above, the context of an archival record is required to disclose its full informational power and thus, reconstructing the knowledge of a record or of an archival division is one of the most common and important operation an archival system has to provide. This operation can be associated with the bottom-up pattern of interaction identified also by [11] where the users "*start at the most detailed level seeking specific information, and then move back to the higher levels to make sense of the information and place it in context if necessary*".

Figure 4 presents the operations required to "build contextual knowledge" of an archival description. To better guide the user when exploring the archive the more accurate the contextual information returned are, the better; indeed, if we return the whole archive to the user then s/he might be disoriented by the large amount of heterogeneous information [22]. To address this aspect we need to return to the user all and only the archival divisions from the selected unit up to the root.

If we consider the case presented in Figure 4 where we need to reconstruct the context of "Unit L", we can see that a structural query needs to return all the archival divisions up to the root – i.e., the ancestors of unit L which are series F, sub-fonds C and fonds A – and the content query returns all the records or descriptions contained by these divisions.

With an XPath-based approach, the structural query (e.g., `/fondsA/subfondsC/seriesF/unitL/ancestor-or-self::*`) requires to navigate the archival tree from the leaf "unit L" up to the root; the output of this query is a sub-tree with the same root of the original tree, but containing only those nodes on the path between "Fonds A" and the leaf "unit L". The content query (`/fondsA/subfondsC/seriesF/unitL/ancestor-or-self::*/text()`) does the same operation but selects only the data nodes that are then returned to the user.

As shown in Figure 4, the NS-M answers the query about the context by exploiting a set-wise primitive which returns all the supersets of the selected division, whereas the INS-M does so by returning all its subsets. This operation also has an element-wise counterpart answering the content query and in this case, NS-M returns all the elements belonging to the union of the supersets of the selected unit, whereas the INS-M simply returns the elements belonging to the set of the unit.

Use Case 3: seeking unknown archival material

This use-case is related to the "*becoming oriented to a new archive or collection*" information seeking activities investigated in [5]. It analyses a common scenario where users have not a clear idea about what they are looking for and may proceed systematically from an archival division to the other. This use case is also related to the two previous ones because, among other operations, it may require to analyze the descendants of a given archival division or record as well as to climb up the hierarchy. Indeed, we can see this use case as a combination of the top-down and the bottom-up patterns and can be associated to the "systematic interrogation" interaction [11], where the users "*develop hypotheses*

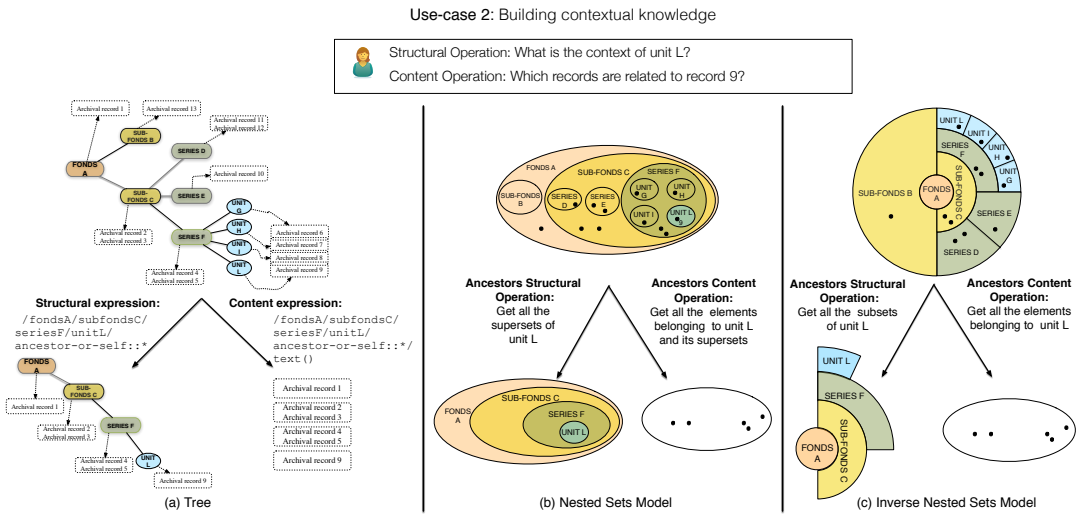


Figure 4: Use-case 2: Building Contextual Knowledge.

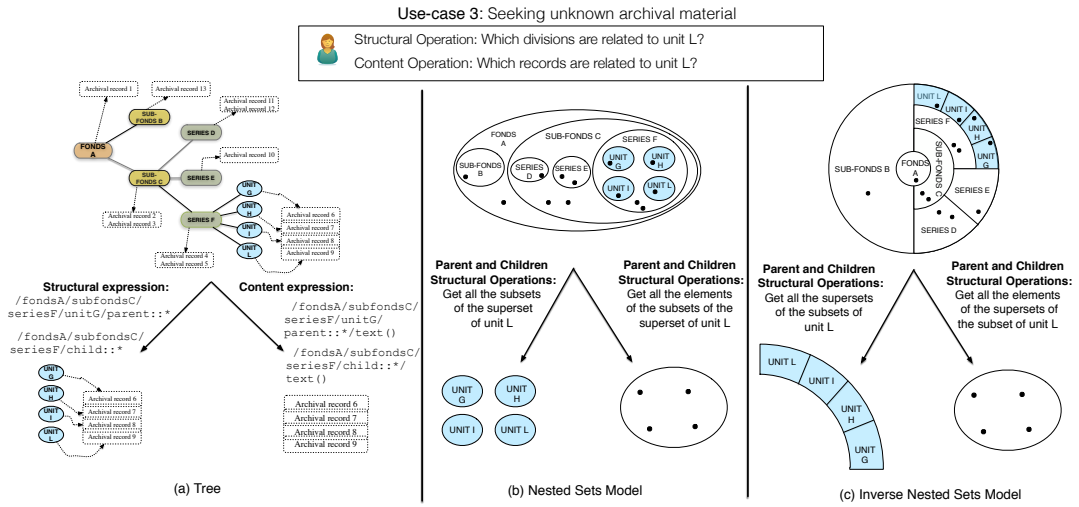


Figure 5: Use-case 3: Seeking unknown archival material.

as to where in the finding aids structure the information is most likely to be and check each one in turn".

In Figure 5 we show this use case where the user selects an archival division or a record and then asks for all the archival divisions (structural or set-wise) or all the records (content or element-wise) at the same level of the selected element (e.g. the siblings of this element). For instance, if the user selects one of record descriptions represented by "Unit L" in the figure, this operation allows her/him to retrieve all the other descriptions connected to it (e.g. all the sibling units of "Unit L" or the elements belonging to them).

We can see that to answer this interrogation, both from the structural and the content viewpoints, the navigational approach requires two XPath expressions where the first one returns the parent node of the given node and the second, starting from this last one node, returns all of its children; note that to do this, navigational approaches need to visit each child node and thus the higher the number of children, the higher the complexity of this operation.

The NS-M answers the query with a set-wise primitive by returning all the direct subsets (i.e. the children) of the superset (i.e. the parent) to which the selected unit belongs; as usual, the INS-M reverses this logic and answers by returning all the direct supersets of the subset to which the selected unit belongs. The element-wise primitive takes the sets outputted by the set-wise one and then returns all the elements belonging to them.

3.2 Interoperability-oriented Use Cases

As described above and reported in [15], digital finding aids based encoded by the EAD standard represent a barrier towards the very interoperability this standard aims to enable. Indeed, as we see below, with EAD there are several OAI-PMH functions which cannot be used by archival systems. On the other hand, NESTOR set-based operations can be straightforwardly employed by archival systems to use all OAI-PMH functionalities with digital finding aids [8].

Use Case 4: Get Records

This use case is based on the a common OAI-PMH request where a service provider requests all the records belonging to an archive. This use case can be addressed also by navigational approaches just by exchanging the whole EAD file via OAI-PMH.

NESTOR addresses this case by relying on the descendant content operation shown in Figure 3 with a slight variation; indeed in the figure we ask for all the descendants of subfonds C, whereas in this case we are asking the NS-M to return the set representing "Fonds A" which contains all the records in the archive, and the INS-M to return the union of all records belonging to the set "Fonds A" and its supersets.

Use Case 5: Get Sub-hierarchy

This use case is a specification of the previous one where the service provider requests only those records belonging to the sub-hierarchy rooted in a given archival division. Navigational approaches do not apply to this case, whereas NESTOR can address it by means of the descendant content operation as shown in Figure 3.

Use Case 6: Get Context

In this case the service provider requests all the records belonging to a specific division, say "Unit L", and to all the related divisions up to the root as shown in Figure 4.

As in the previous case, navigational approaches do not apply to this case, whereas NESTOR addresses it by employing the ancestor content primitive which for the NS-M returns the union of all the records belonging to "Unit L" and its supersets and for the INS-M returns all the elements belonging to the "Unit L".

Use Case 7: List Sets

This use case is related to the "listSets" OAI-PMH verb "used to retrieve the set structure of a repository" and allows the service provider to know the structure of a local repository in advance.

This request cannot be answered by an XPath expression because it is not possible to extract only structural information filtering out all data nodes; moreover, the OAI-PMH set-based organization of metadata does not apply to EAD. On the other hand, answering the "listSets" verb is natural for NESTOR because it retains the structure by exploiting inclusion relationships between sets. Therefore, it answers this request by employing the descendant structure operation as shown in Figure 3.

4. VALIDATION

We proposed three different instantiations of NESTOR according to three alternative data structures, namely DDS, IDS and HDS. In order to compare the query operations defined on these data structures with currently adopted solutions for operating on digital archives we selected two EAD collections that provide us with real-world archival data: the National Archives of the Netherlands[6] and the Library of Congress finding aids.

We selected ten EAD files taken from these collections representing a wide variety of archives with different characteristics representing key challenges for archival systems. The statistics about these files are reported in Table 1.

[6]http://www.nationaalarchief.nl/

Table 1: Statistics of ten selected EAD files.

	Size (MB)	# nodes	depth	max fan-out	average fan-out
EAD-01	0.368	7,316	10	823	4.33
EAD-02	1.853	21,355	10	1,610	1.62
EAD-03	3.131	42,123	13	2,453	1.49
EAD-04	3.866	75,094	9	10,271	1.73
EAD-05	4.043	51,946	12	1,320	1.80
EAD-06	5.310	73,372	12	3,663	1.87
EAD-07	6.017	57,362	14	565	1.91
EAD-08	9.242	103,703	18	340	1.62
EAD-09	9.746	160,031	14	8,930	2.01
EAD-10	15.512	188,862	17	696	1.62

DDS, IDS and HDS are compared to widely-adopted ready to use solutions based on the XPath for operating of the structure and the content of EAD files: Xalan, Jaxen and JXPath, which represent the state-of-the-art solutions for dealing with EAD files[7].

The main characteristic of EAD files representing a challenge for XPath libraries is the number of nodes in each file; the selected files are of increasing sizes to show that navigational-based solution performances depend by the number of nodes and the overall dimension of the EAD files, whereas this does not apply for the set-based operations implemented by NESTOR. Indeed, in Figure 6 we can see that all the XPath libraries answer in linear time with respect to the size of the EAD file because they need to navigate big hierarchies by visiting a great number of nodes. On the other hand, we can see that IDS answers the descendant structural operation in constant time for all the EAD files and it is five orders of magnitude faster than XPath-based solutions. DDS and HDS show some dependence on the size of the EAD file; indeed, they need to perform some set operations (more nodes mean more operations) that require some time, even though for the descendant content operation, they are several orders of magnitude more efficient than navigating the archival hierarchy. Overall, IDS is the best solution for addressing use case 1 and 7, whereas DDS is the best for use cases 1, 4 and 5.

It is interesting to note that for addressing use cases 1, 4 and 5, XPath-based libraries are slower for the EAD-04 file which is the one with the highest number of children (i.e., 10,271) followed by EAD-09 which also has a high number of children (i.e., 8,930). These two files are challenging for all the use cases requiring the descendants or the children of a node such as use cases 1, 3 and 5. Navigational-based solutions are particularly challenged by this case as we can see in Figure 6 for the content operation and in Figure 8. On the other hand, we can see that the IDS and the HDS are not affected by the high max fan-out of these files given that they can answer without visiting the high number of child nodes, but just by returning a set or by performing basic set operations. DDS requires more set operations than the other two set-based solutions; even though in most cases it is consistently more efficient than navigation-based solutions, it is still less performing than IDS and HDS which are extremely efficient for these cases. The overall performances reported in Figure 8 with a particular focus on EAD-04 and EAD-09 show that set-based solutions are particularly well-suited to address the operation employed by use case 3.

[7]We ensure a fair comparison because all the tested solutions are implemented in Java, work in central memory and are tested on the same machine.

Lastly, use case 2 requires to climb up the archival hierarchy from a given entry point. We considered EAD files with variable depth (from 9 to 17) and we validated the ancestor operations using the deepest node in each hierarchy as entry point which represents the worst case scenario for any archival system. From a performance viewpoint, in Figure 7 we can appreciate the difference between the NESTOR set-based approaches and the XPath navigational approaches. Indeed, NESTOR-based solutions behave consistently for all the tested EAD files and do not depend by the depth and size of EAD files. On the other hand, the XPath libraries behave differently from file to file showing a dependence on the number of nodes, fan-out and depth of the files; for instance, JXPath behaves less efficiently when EAD files have a high max fan-out (EAD-04 and EAD-09), whereas Xalan performances worsen as the number of nodes increases.

5. CONCLUSIONS

In this paper we identified and described the barriers preventing an efficient access to archival data. We described the main drawbacks of EAD and we showed how it impairs a smooth and efficient access to archival descriptions as well as that it does not satisfy several interoperability requirements.

We analyzed the role of the NESTOR model in the context of digital archives and described its main advantages with respect to state-of-the-art navigational-based solutions. We have seen that NESTOR set-based approach represents a paradigm shift in the access of XML files which is well-suited to enable interaction and interoperability functionalities in the archival context.

We identified and described seven use cases highlighting the key challenges archival systems have to address in order to deal with common user interaction patterns and to satisfy interoperability requirements. In this frame of reference, we compared and discussed strengths and limitations of navigational-based solutions with respect to NESTOR set-based ones.

We have seen that NESTOR is a model of access to archival resources that allows us to better address the identified needs both from the user and the interoperability viewpoints. From a quantitative standpoint, the experimental validation confirms that NESTOR-based solutions consistently outperform state-of-the-art solutions; moreover, we have seen that NESTOR-based solutions are less dependent – or not dependent at all – on the hierarchical structure of archives than navigational-based ones.

References

[1] J. C. Chapman. Observing Users: An Empirical Analysis of User Interaction with Online Finding Aids. *J. of Arch. Org.*, 8(1):4–30, 2010.

[2] J. G. Daines and C. L. Nimer. Re-Imagining Archival Display: Creating User-Friendly Finding Aids. *J. of Arch. Org.*, 9(1):4–31, 2011.

[3] M. G. Daniels and E. Yakel. Seek and You May Find: Successful Search in Online Finding Aid Systems. *American Archivist*, 73:535–468, 2010.

[4] E. Discovery, S. Shaw, and P. Reynolds. Creating the Next Generation of Archival Finding Aids. *D-Lib Mag.*, 13(5/6), 2007.

[5] M. W. Duff and C. A. Johnson. Accidentally Found on Purpose: Information-Seeking Behavior of Historians in Archives. *The Library Quarterly*, 72(4):472–496, 2002.

[6] L. Duranti. *Diplomatics: New Uses for an Old Science*. Society of Amer. Arch. and Association of Canadian Arch., 1998.

[7] M. Y. Eidson. Describing Anything That Walks: The Problem Behind the Problem of EAD. *Journal of Archival Organization*, 1(4):5–28, 2002.

[8] N. Ferro and G. Silvello. NESTOR: A Formal Model for Digital Archives. *Inf. Proc. Manage.*, 49(6):1206–1240, 2013.

[9] N. Ferro and G. Silvello. Descendants, Ancestors, Children and Parent: A Set-Based Approach to Efficiently Address XPath Primitives. *Inf. Proc. Manage.*, 52(3):399-429, 2016.

[10] L. Francisco-Revilla, C. B. Trace, H. Li, and S. A. Buchanan. Encoded Archival Description: Data Quality and Analysis. *Proc. American Society for Inf. Science and Tech.*, 51(1):1–10, 2014.

[11] L. Freund and E. G. Toms. Interacting with Archival Finding Aids. *JASIST*, 67(4):994-1008, 2015.

[12] I. Huvila. Participatory archive: towards decentralised curation, radical user orientation, and broader contextualisation of records management. *Archival Science*, 8(1):15–36, 2008.

[13] N. A. Khan. Emerging Trends in OAI-PMH Application. In *Design, Development, and Management of Resources for Digital Library Services*, pages 147–159, 2013.

[14] D. V. Pitti. Encoded Archival Description. An Introduction and Overview. *D-Lib Mag.*, 5(11), 1999.

[15] C. J. Prom. Does EAD Play Well with Other Metadata Standards? Searching and Retrieving EAD Using the OAI Protocols. *J. of Arch. Org.*, 1(3):51–72, 2002.

[16] C. J. Prom. User Interactions with Electronic Finding Aids in a Controlled Setting. *The American Archivist*, 67(2):234–268, 2004.

[17] C. J. Prom and T. G. Habing. Using the Open Archives Initiative Protocols with EAD. In *Proc. 2nd Joint Conference on Digital Libraries*, pages 171–180. ACM Press, 2002.

[18] J. Roth. Serving Up EAD: An Exploratory Study on the Deployment and Utilization of Encoded Archival Description Finding Aids. *The Amer. Arch.*, 64(2):214–237, 2001.

[19] W. Scheir. First Entry: Report on a Qualitative Exploratory Study of Novice User Experience with Online Finding Aids. *J. of Arch. Org.*, 3(4):49–85, 2006.

[20] A. Sexton, C. Turner, G. Yeo, and S. Hockey. Understanding users: a prerequisite for developing new technologies. *Journal of the Society of Archivists*, 25(1):33–49, 2004.

[21] S. L. Shreeves, T. G. Habing, K. Hagedorn, and J. A. Young. Current Developments and Future Trends for the OAI Protocol for Metadata Harvesting. *Library Trends*, 53(4):576–589, Spring 2005.

[22] S. Yako. It's Complicated: Barriers to EAD Implementation. *American Archivist*, 71(2):456–475, 2008.

[23] J. Zhang. Archival Representation in the Digital Age. *J. of Arch. Org.*, 10(1):45–68, 2012.

[24] X. Zhou. Examining Search Functions of EAD Finding Aids Web Sites. *J. of Arch. Org.*, 4(3/4):99–118, 2008.

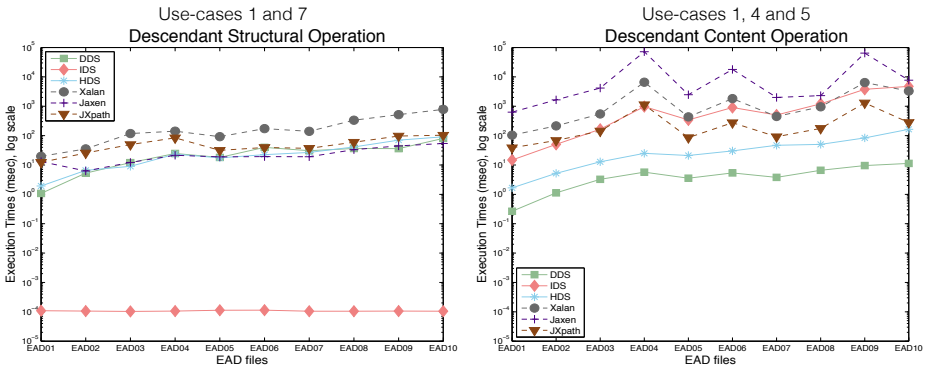


Figure 6: Execution times of the descendant structural and content operations.

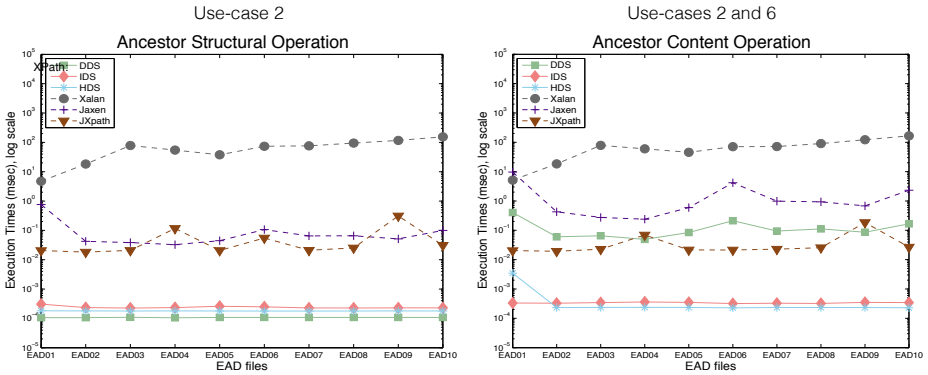


Figure 7: Execution times of the ancestor structural and content operations.

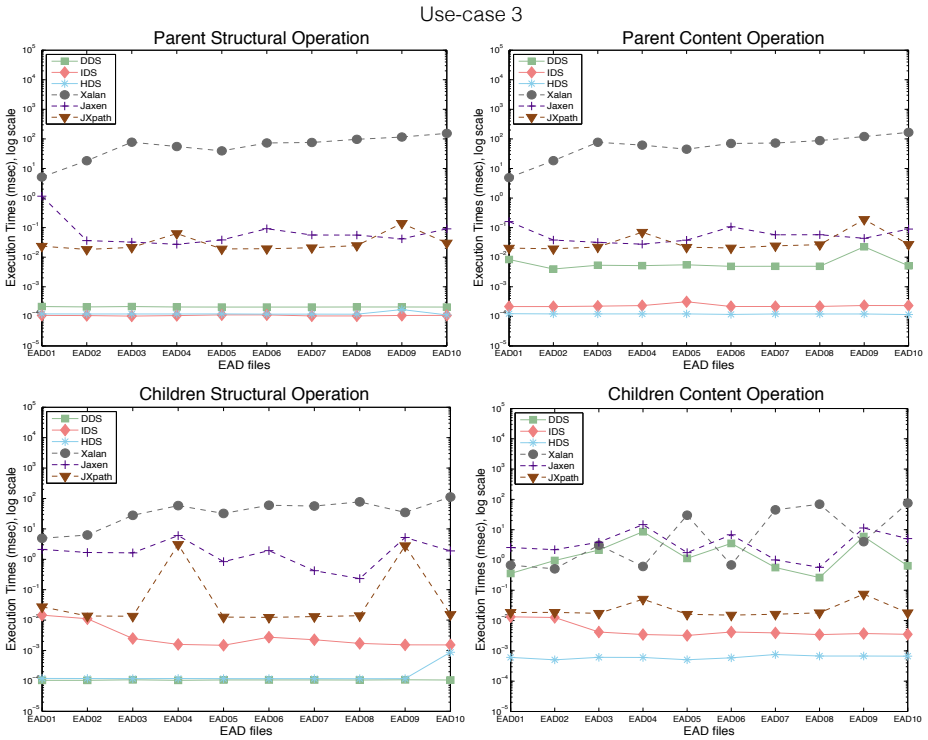


Figure 8: Execution times of the parent and children structural operations.