Metadata++: A Scalable Hierarchical Framework for Digital Libraries[1]

[*]Mathew Weaver, [*]Lois Delcambre, [**]Timothy Tolle

[*]Computer Science & Engineering
OGI School of Science & Engineering
Oregon Health & Science University
{mweaver, lmd}@cse.ogi.edu

[**] Resource Planning and Monitoring
Pacific Northwest Region
USDA Forest Service
ttolle@fs.fed.us

Abstract:

Metadata++ is a digital library system that we are developing to serve the needs of the United States Department of Agriculture Forest Service, the United States Department of the Interior Bureau of Land Management, and the United States Department of the Interior Fish and Wildlife Service to support natural resource managers, scientists and publics as they analyze issues and make decisions. The system provides access to institutional knowledge consisting of formal and informal agency reports and documents – including Environmental Assessments, Decision Notices, Appeal Decisions, specialist reports, and so forth. Metadata++ uses a set of hierarchically structured controlled vocabularies – with synonyms and associations – as the primary organizational framework. Users browse the hierarchy to select search terms and see the search results directly in the context of the hierarchy. In order to be useful as a digital library infrastructure, this hierarchy must be implemented in an efficient and scalable manner. This paper introduces the Metadata++ system and evaluates the performance of four different approaches to managing the hierarchy. We present a novel approach that uses a common file system with an associated indexing engine to store terms as directories (with narrower terms as subdirectories) and show how we achieve both scalability and efficiency.

Keywords:

Information technologies for digital libraries; Development of digital libraries

1 Introduction

Resources in a digital library are typically described by metadata, including an indication of the subject(s) or topic(s) (also called *keywords* or *terms*) of the resource. In this work, we are particularly interested in keywords that have been predefined in a *controlled vocabulary* with a hierarchical structure relating broader/narrower terms. Using a controlled vocabulary produces more meaningful and useful metadata. In addition to providing a finite set of keywords, a controlled vocabulary often includes a structured organization of the keywords – as in a traditional thesaurus [1] and Dewey Decimal [5]. Most common metadata standards – such as Dublin Core [6] and FGDC [7] – require or permit the use of one or more controlled vocabularies.

```
Places
  USFS
  BLM
Forestry
  Agriculture
  Botany
  Silviculture
Air
  Air quality
    Air pollution
    Ozone
  Weather
    Air pressure
    Evaporation
```

Figure 1: Excerpts from several CVs

Controlled vocabularies are an essential tool within natural resource management. Scientists, governing agencies, and natural resource managers use specific vocabularies to describe their work. Our environmental research partners [3] have identified approximately twenty-eight domains of interest, such as location, hydrology, climatology, forestry, and planning, as central to the organization of documents and have identified, evaluated, and selected one or more controlled vocabularies for each domain. Figure 1 shows excerpts of controlled vocabularies from three domains. Each controlled vocabulary organizes terms according to the way that specialists in a given domain view the concepts. For a forester, "Forestry" is a primary topic, with "Agriculture" and "Botany" as sub-topics. The climatologist, on the

[1] This work is supported in part by the National Science Foundation, grant number EIA 9983518. Any opinions, findings, conclusions, or recommendations expressed here are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

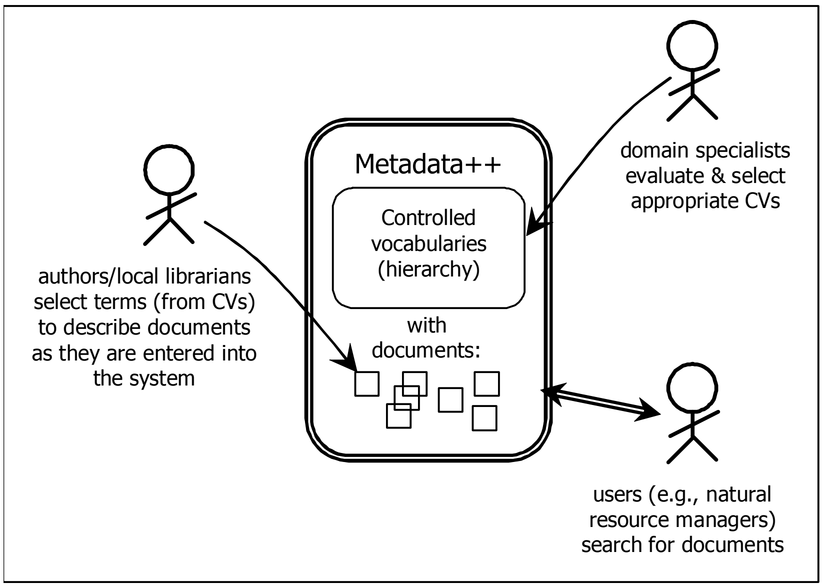
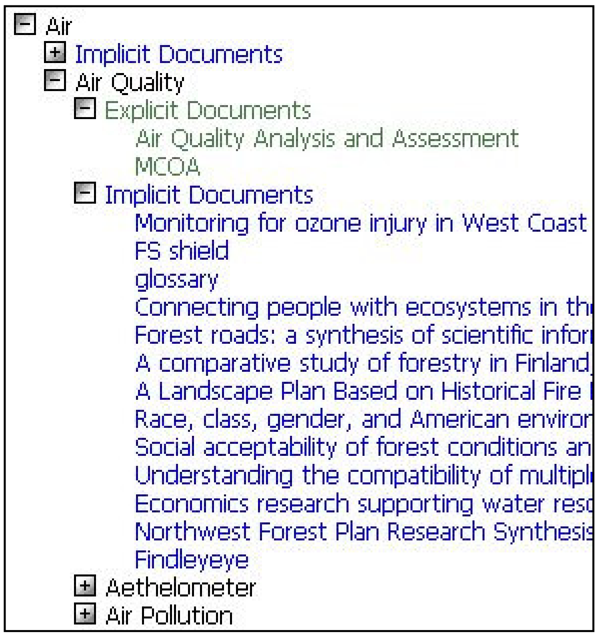

Metadata++
Controlled vocabularies (hierarchy)
with documents:
authors/local librarians select terms (from CVs) to describe documents as they are entered into the system
domain specialists evaluate & select appropriate CVs
users (e.g., natural resource managers) search for documents

Figure 2: Overview of Metadata++

other hand, considers "Air" as a primary topic, with "Air quality" and "Weather" as sub-topics.

The hierarchical relationship among terms represents various relationships including spatial containment (e.g., watersheds), administrative jurisdiction (e.g., national forests and ranger districts), classification of the plant and animal kingdom (e.g., taxonomic description of species), or simple broader/narrower term (e.g., *air* vs. *air quality*) as shown in Figure 1. In our model, we use one hierarchical relationship as an abstraction of these – called *broader/narrower term.* Like a standard thesaurus [1], Metadata++ includes *synonym* and *association* relationships among terms, as well.

As shown in Figure 2, *domain specialists* select appropriate controlled vocabularies for Metadata++ (upper right portion of the figure) and *authors* or *local librarians* supply metadata and select terms from the controlled vocabularies to describe documents within Metadata++ (left portion of the figure). *Users* are then able to easily browse the controlled vocabularies and search for documents (shown on the lower right of the figure). This paper explains how Metadata++ uses controlled vocabularies and evaluates four alternative approaches to implementing a large-scale, dynamic hierarchy.



```
⊟ Air
   ⊞ Implicit Documents
   ⊟ Air Quality
      ⊟ Explicit Documents
           Air Quality Analysis and Assessment
           MCOA
      ⊟ Implicit Documents
           Monitoring for ozone injury in West Coast
           FS shield
           glossary
           Connecting people with ecosystems in th
           Forest roads: a synthesis of scientific infor
           A comparative study of forestry in Finland
           A Landscape Plan Based on Historical Fire
           Race, class, gender, and American enviror
           Social acceptability of forest conditions an
           Understanding the compatibility of multipl
           Economics research supporting water reso
           Northwest Forest Plan Research Synthesis
           Findleyeye
      ⊞ Aethelometer
      ⊞ Air Pollution
```

Figure 3: Partial Screenshot of Metadata++ Hierarchy

2 Using Controlled Vocabularies in Metadata++

Controlled vocabularies provide an intuitive framework for cataloging and retrieving natural resource information. Users may explore the hierarchy of controlled vocabularies and see documents related to the terms of interest. Figure 3 shows a portion of a screen shot of the Metadata++ application while browsing the hierarchy. Explicit documents are documents that an author or librarian explicitly attached to the relevant term (by selecting the term as a keyword for the document). For example, the author of the document titled "Air Quality Analysis and Assessment" selected the term "Air quality" as a keyword for that document. Implicit documents, on the other hand, contain one or more occurrences of the term, but have not been explicitly attached. For example, the document "Monitoring for ozone injury in West Coast" actually contains the term "Air quality" – but the term was not selected as an explicit keyword. As the library grows, librarians may modify the hierarchy by adding new vocabularies or changing existing vocabularies. The concept of explicit documents and implicit documents allows authors, librarians, and users to immediately benefit from new vocabularies – without having to rebuild metadata for existing documents.

In addition to exploring the hierarchy, the user may issue a search for a specific term. The results of the search include documents related directly to the search term as well as documents related to descendant terms. For example, a search for "Air" would include the document "Air Quality Analysis and Assessment," because the term "Air quality" is a descendant of the term "Air". Instead of returning a ranked list of documents, Metadata++ returns structured search results – so the user can view the documents in context of the hierarchy. Metadata++ uses the SUBTREE function to compute all descendants of a given term.

In addition to finding descendant terms, the user must also be able to find one or more terms given a specified string value. For example, a novice user may be unfamiliar with the hierarchy and have difficulty finding a term of interest. Instead of manually browsing the entire hierarchy to find the desired term, the user may type in a string – such as "evaporation". A function called FIND returns all terms that contain the specified string. The FIND function supports two modes – exact match (which returns only terms that exactly match the string) and wildcard (which returns all terms that contain the string).

PARENT	CHILD
Places	USFS
Places	BLM
Forestry	Agriculture
Forestry	Botany
Forestry	Silviculture
Air	Air quality
Air quality	Air pollution
Air quality	Ozone
Air	Weather
Weather	Air pressure
Weather	Evaporation

Figure 4: Parent-Child

3 Implementing Hierarchical Controlled Vocabularies

Metadata++ uses a hierarchy of controlled vocabularies to provide an intuitive framework for building a digital library, as described above. In order to be useful as a digital library framework, this hierarchy must be implemented in an efficient and scalable manner. Specifically, the SUBTREE and FIND functions must be performed in real-time. We must also accommodate concurrent additions and modifications to the controlled vocabularies. This section describes four different approaches to implementing the hierarchy for Metadata++ using various combinations of database, XML, and file system technology. We implemented and evaluated the advantages and disadvantages of each approach.

3.1 Parent-Child Binary Relation

The simplest approach uses a relational database table that represents a binary relationship between terms – one row in the table for each parent-child relationship. This approach (illustrated in Figure 4) is similar to the edge relation described in [4, 8]. It is important to note that this approach (as well as the next two approaches) actually uses term identifiers that act as foreign keys to a separate table of terms. The figure shows the actual text of the term (instead of the term identifier) to improve readability.

This approach is functional, but fails in terms of scalability and performance. Because each parent-child relationship is a separate row in the table, executing the SUBTREE function (i.e. to do a search or display the hierarchy to the user) requires an additional query for each term. Even with relatively few terms (i.e. 10^2 terms), the performance of the SUBTREE function is noticeably slow. The FIND function also performs slowly. It does not take long to find the matching terms (exact match or wildcard) – the majority of the time is spent finding the ancestors in order to return the precise location in the hierarchy. Despite poor query performance, adding or deleting leaf nodes is very efficient – simply add rows or delete rows from the relation. Modifying internal (non-leaf) nodes requires more time to correctly process the affected sub-trees. The underlying database management system supports concurrent updates to the hierarchy.

3.2 Breadth-first Path

Our second implementation also uses a relational database, but includes a more novel approach for storing the hierarchy (similar to Dietz's numbering scheme as described in [11]). Instead of storing the

ID	PATH
1	Places
2	Forestry
3	Air
4	Places\USFS
5	Places\BLM
6	Forestry\Agriculture
7	Forestry\Botany
8	Forestry\Silviculture
9	Air\Air quality
10	Air\Weather
11	Air\Air quality\Air pollution
12	Air\Air quality\Ozone
13	Air\Weather\Air pressure
14	Air\Weather\Evaporation

Figure 5: Path

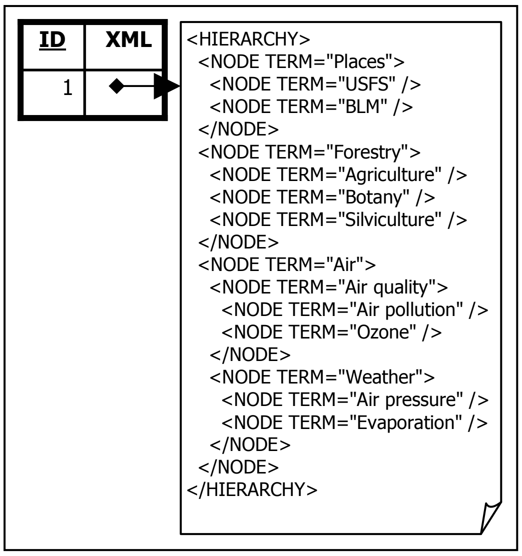
hierarchy as a set of parent-child relationships, a structure is used based on the breadth-first ordering of the tree. Each node of the tree is stored – in breadth-first order – with its full path in the hierarchy (as illustrated in Figure 5). (As noted previously, the path is actually stored using term identifiers, but the figure shows the term text for improved readability). The hierarchy can be constructed using a single SQL query. This approach offers a great improvement over the previous approach when executing the SUBTREE function. The nodes in the SUBTREE can be determined by matching the prefix of each path – and the structure of the SUBTREE can be created by re-building the tree in breadth-first order. This implementation requires only a single SQL query – instead of several SQL queries or expensive joins. The FIND function is also efficient – doing a string comparison on the path. Because the relation stores the full path, the FIND function does not need to issue additional queries to determine the path (as with the Parent-Child approach).

As the number of terms increases (i.e., 10^3 terms), modifications to the hierarchy take unreasonable amounts of time. Whenever a new term or vocabulary is added to the hierarchy – or an existing term is moved to a new location – the entire hierarchy must be saved. This process requires computing the breadth-first index and full path for each node – and saving the complete hierarchy to the database. With a large number of terms, this process takes several minutes to complete – an obvious failure in performance. Because the hierarchy must be saved in its entirety (after computing the breadth-first index), this solution does not easily support concurrent modifications. Deleting existing terms is significantly faster – because it does not require rebuilding the index. When a node is deleted from the hierarchy – it simply requires deleting the appropriate row(s) from the table.

3.3 XML BLOB

The flexible structure of XML provides an ideal representation for the hierarchy of terms. Our third approach for managing the hierarchy in Metadata++ uses a single XML document to represent the hierarchy, stored as text in a relational database (using a BLOB field). This approach is illustrated in Figure 6. (As with the previous approaches, the XML in the figure shows the text of the terms, but the actual implementation uses term identifiers). Other data (synonyms, associations, document metadata) are also stored in relational tables. Metadata++ uses the XML to build (and save) the hierarchy and uses the relational tables (i.e. SQL) to query the related information.

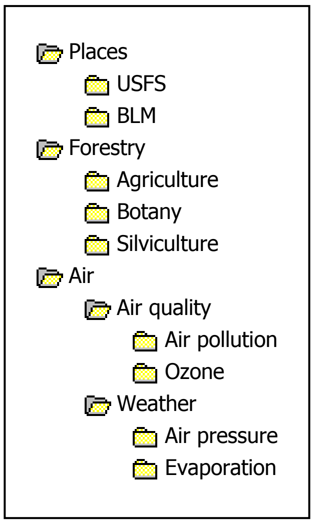
Because the hierarchy is in XML, it is easy (using XPath) to execute both the SUBTREE and FIND functions. One drawback of this approach is that the entire XML document must be retrieved from the database and parsed into main memory. This process adds *significant* initial processing overhead. Additionally, this solution does not support modifications from concurrent users. Suppose one user is adding terms to one portion of the hierarchy, while a second user is adding terms to a different portion of the hierarchy. Because the entire XML string is treated as a single value – and the updates to that string are serialized by the relational database system – only the latest modifications are retained in



ID	XML
1	◆

```
<HIERARCHY>
  <NODE TERM="Places">
    <NODE TERM="USFS" />
    <NODE TERM="BLM" />
  </NODE>
  <NODE TERM="Forestry">
    <NODE TERM="Agriculture" />
    <NODE TERM="Botany" />
    <NODE TERM="Silviculture" />
  </NODE>
  <NODE TERM="Air">
    <NODE TERM="Air quality">
      <NODE TERM="Air pollution" />
      <NODE TERM="Ozone" />
    </NODE>
    <NODE TERM="Weather">
      <NODE TERM="Air pressure" />
      <NODE TERM="Evaporation" />
    </NODE>
  </NODE>
</HIERARCHY>
```

Figure 6: XML BLOB



- Places
  - USFS
  - BLM
- Forestry
  - Agriculture
  - Botany
  - Silviculture
- Air
  - Air quality
    - Air pollution
    - Ozone
  - Weather
    - Air pressure
    - Evaporation

Figure 7: NTFS/IS

the database.

3.4 NTFS and Microsoft Index Server

Allowing concurrent access to native XML would be a suitable solution. This would provide efficient execution of all of the necessary functions and support modifications from multiple users. Although some native XML databases do exist [10, 15], few are actually available for use. We found none that include all of the important features found in mature relational databases (such as concurrency). In order to achieve the concurrency needed to support multiple users and obtain the necessary performance requirements, we implemented a simple solution based on the file system (Microsoft© NTFS) and Microsoft Index Server© [12].

The hierarchy is stored and modified as folders in the file system as illustrated in Figure 7. It should be noted that there is no concept of term identifier in this implementation – the folder names are the actual terms. The file system is naturally hierarchical – and already has familiar tools for creating, editing, and deleting folders. The file system also provides the concurrency necessary to allow multiple users to modify the hierarchy simultaneously. Different users can work in different parts of the hierarchy and all of the changes will persist. The SUBTREE function looks at the directory structure to find all descendant terms. (It should be noted that scalability limitations with user interface tools – such as Windows Explorer© – are not necessarily limitations of the file system itself).

We configured Microsoft Index Server – a full-text indexing and search service – to catalog all of the file system folders defined in the hierarchy. The index is searchable by folder name – providing support for the FIND function (both exact match and wildcard). Microsoft Index Server provides an efficient query interface for finding folders within the hierarchy. Microsoft Index Server also provides full-text indexing of all documents in Metadata++, thus enabling the identification of implicit documents, as described above.

4 Performance Results

We implemented each approach described in the previous section on an IBM© xSeries 220 eServer containing a single Intel© Pentium III (1.0 GHz) processor, 896 MB of RAM, and IBM ServeRAID storage. The server is running Microsoft Windows© Server 2003 Standard Edition and Microsoft SQL Server© 2000 Enterprise Edition. The fourth solution uses Microsoft Index Server© (included in the Windows Server 2003 operating system). We conducted the experiments using code written and compiled with the Microsoft .NET Framework. The hierarchy consisted of numerous controlled vocabularies provided by various application domain experts – with a combined total of more than 70,000 terms. Figure 8 summarizes the depth (number of levels) and width (number of children per node) of the hierarchy.

	Max	Avg
Depth	14	6.4
Width	9896	*19.5

*does not include leaf nodes

Figure 8: Hierarchy Statistics

	A	B	C	D	E	Avg
ParentChild	2.3947	2.4259	0.3556	0.0040	0.2618	1.1103
Path	0.6407	0.9376	0.4493	0.2150	0.2657	0.5009
XmlBlob	0.4220	0.6603	0.7501	0.0040	0.0470	0.4204
NTFS/IS	0.3712	0.6642	0.3673	0.0001	0.0118	0.1837

Figure 9: Average SUBTREE Execution Times for 5 queries

For each approach, we implemented and evaluated both the SUBTREE and FIND functions. We executed each function multiple times (with inputs and outputs varying in size and complexity) and averaged the execution times. Figure 9 shows the results of the SUBTREE function for five different queries. Figure 10 compares the maximum, minimum, and average SUBTREE execution times for each approach.

Figure 11 shows the results of the FIND function for four different queries. We executed each query in both modes (exact match and wildcard) and averaged the results. Figure 12 compares the maximum, minimum, and average execution times for each approach. The NTFS/IS approach is an order of magnitude faster (both maximum and average).

As described in the previous section, each approach performed differently with regards to concurrent updates. The Parent-Child approach efficiently handled updates at the leaf level (adding or deleting leaf nodes), but not when modifying entire sub-trees. The Path approach efficiently handles deletes – by simply deleting the appropriate rows in the table – but additions require saving the entire hierarchy. The XML is easily modifiable while in-memory, but does not support concurrent updates (and requires serialization with each modification). The NTFS/IS approach has the overall best performance. The file system natively supports concurrent updates to different parts of the directory structure, and Index Server watched for changes in the hierarchy and efficiently updated its index for each modification.

5 Related Work

The natural choice for representing a hierarchy is XML. Recent languages such as XQuery and XPath make it easy and efficient to query hierarchical data stored in XML. However, the large and dynamic nature of the Metadata++ hierarchy is not adequately supported by current XML tools. A full-featured XML database would be an ideal solution, but the work in this area [10, 15] has yet to produce a native XML database with all of the necessary features.

An alternative to building a native XML database is to map the XML to relations. The first two implementations described in this paper (Parent-Child and Path) are attempts at mapping XML-like data into a relational database. The Parent-Child

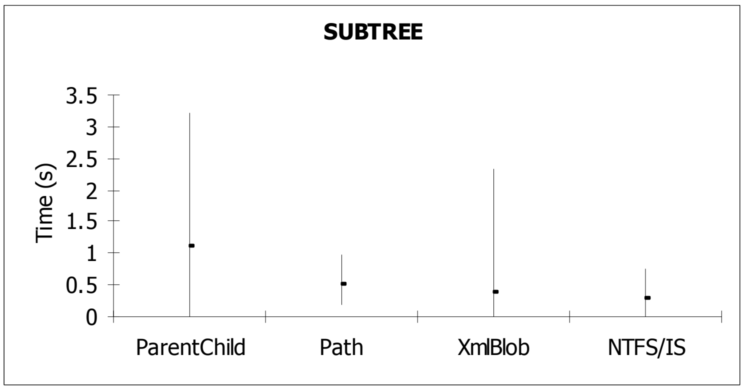


Figure 10: SUBTREE Max-Avg-Min Comparison Chart

	A	B	C	D	Avg
ParentChild	88.4766	2.9297	0.7110	50.4219	35.6348
Path	16.7266	0.4454	0.2579	13.5782	7.7520
XmlBlob	54.0625	1.9688	0.5079	44.1016	25.1602
NTFS/IS	0.6250	1.7344	0.5625	0.5623	0.8711

Figure 11: Average FIND execution times for 4 queries
(average of exact match and wildcard searches for each query)

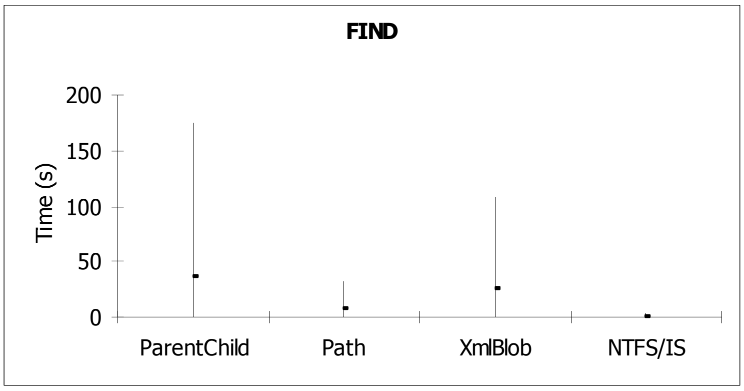


Figure 12: FIND Max-Avg-Min Comparison Chart

relation is similar to the edge relation described in [4, 8]. LegoDB [2] takes a novel approach to mapping XML into relations by doing a cost-based evaluation of different relational representations. Because the hierarchy in Metadata++ is a simple tree of arbitrary width and depth, the LegoDB algorithm would also result in a simple edge-based relation. Metadata++ requires functionality (i.e. SUBTREE and FIND) that is not efficiently supported by this type of representation. Our Path approach is similar to XML indexing techniques described in [11]. This technique works well for static data, but updates are more difficult. The authors propose leaving gaps in the ordering to handle updates, but this is still not suitable for frequent and arbitrary updates.

The Lightweight Directory Access Protocol (LDAP) [9, 13] defines basic directory services for a hierarchically structured set of directories. LDAP allows a single conceptual directory hierarchy to be stored in multiple, distributed locations. The Metadata++ hierarchy differs from the directory abstraction supported by LDAP in that Metadata++ terms (in the hierarchy) consist only of the word or phrase that makes up the term (the name of the directory); there is no type structure for terms and, in particular, there is no need to have attributes for terms. Thus Metadata++ uses a much simpler structure for the term hierarchy with a rather narrow focus on the SUBTREE and FIND functions.

6 Conclusions and Work in Progress

Metadata++ makes several important contributions. As a model for controlled vocabularies, Metadata++ differs from standard thesauri by (1) allowing terms to appear in multiple locations in the hierarchy, (2) allowing two terms, when they both appear in two different portions of the hierarchy, to have different narrower/broader term relationships, and (3) treating all terms with equal emphasis (i.e., there is no notion of preferred term). As a digital library application, Metadata++ allows users to see search results directly in the context of the hierarchy. That is, documents are not ranked in the search result; they are shown directly with the term or narrower term that they match. Metadata++ also unifies the use of terms attached explicitly by a librarian (for *explicit* documents in a search result) with terms found through text indexing techniques (for *implicit* documents in a search result).

The focus of this paper is on implementation strategies for a large-scale term hierarchy. With over 70,000 terms, with new terms being added and arranged on an ongoing basis, we had to find an implementation that would support the SUBTREE and FIND operations efficiently while still maintaining consistency during concurrent updates. The primary contribution of this paper is the performance analysis of a series of implementations that demonstrates that the directory/subdirectory structure of a file system, when coupled with an index server, provides an ideal solution for storing large-scale term hierarchies. This result may be applicable to other applications that must manage large-scale hierarchical information.

Work in progress includes extensive user testing, with a goal of deploying Metadata++ for use in natural resource management, and formalizing the Metadata++ model. We will also continue our work in integrating Metadata++ with geographic information systems [14].

7 References

[1] ANSI/NISO Z39.19 – 1993. Guidelines for the Construction, Format, and Management of Monolingual Thesauri. NISO Press, 1994.

[2] Bohannon, Philip, Juliana Freire, Prasan Roy and Jerome Simeon From XML Schema to

Relations: A Cost-Based Approach to XML Storage (ICDE 2002)

[3] Delcambre, Lois, Timothy Tolle. "Harvesting Information To Sustain Forests". *Communications of the ACM*, January 2003 Volume 46, Number 1, pp. 38-39.

[4] Deutsch, Alin, Mary F. Fernandez, Dan Suciu. *Storing Semistructured Data with STORED*. (SIGMOD 1999)

[5] *Dewey Decimal Classification.* http://www.oclc.org/dewey/index.htm

[6] *Dublin Core Metadata Initiative*. http://www.dublincore.org/

[7] *Federal Geographic Data Committee.* http://www.fgdc.gov/

[8] Florescu, Daniela, Donald Kossmann: Storing and Querying XML Data using an RDMBS. IEEE Data Engineering Bulletin 22(3): 27-34 (1999)

[9] Howes, Timothy A., Mark C. Smith, *LDAP: Programming Directory-Enabled Applications with Lightweight Directory Access Protocol*, Macmillan Technical Publishing, 1997.

[10] Jagadish, H.V., Shurug Al-Khalifa, Adriane Chapman, Laks V.S. Lakshmanan, Andrew Nierman, Stelios Paparizos, Jignesh M. Patel, Divesh Srivastava, Nuwee Wiwatwattana, Y.Wu and C.Yu. TIMBER: A Native XML Database. The VLDB Journal, Volume 11 Issue 4 (2002) pp 274-291.

[11] Kaushik, Raghav, Philip Bohannon, Jeffrey F. Naughton, Henry F. Korth Covering indexes for branching path queries. SIGMOD Conference 2002: 133-144

[12] Nareddy, Krishna. "Indexing with Microsoft Index Server". Microsoft Corporation, January 1998.

[13] Open LDAP Project. http://www.openldap.org/

[14] Weaver, Mathew, Lois Delcambre, Leonard Shapiro, Jason Brewster, Afrem Gutema, Timothy Tolle. "A Digital GeoLibrary: Integrating Keywords And Place Names", *7th European Conference on Digital Libraries* (ECDL 2003), Trondheim, Norway, August 2003.

[15] XML:DB Initiative for XML Databases. http://www.xmldb.org/