Towards a Visual Query Interface for Phylogenetic Databases

Hasan M. Jamil
Department of Computer Science
Mississippi State University
jamil@cs.msstate.edu

Giovanni A. Modica
Department of Computer Science
Mississippi State University
gmodica@cs.msstate.edu

Maria A. Teran
Department of Computer Science
Mississippi State University
teranm@cs.msstate.edu

ABSTRACT

Querying and visualization of phylogenetic databases remain a great challenge due to their complex tree type semi structured nature. Naturally, successful phylogenetic databases such as the *Tree of Life* database at the University of Arizona are implemented as Web documents in HTML. While Web implementation of such databases facilitate the representation, and in part, visualization of their contents, querying remains an issue. The interoperability of Web-based phylogenetic databases with other similar databases such as TreeBase and RDB which are implemented using traditional database management systems, has not been possible due to the impedance mismatch between the underlying query and data representation framework. In this paper, we present a novel approach to phylogentic database management using existing database technologies without compromising potential opportunities for visualization and interoperability. We present a Web-based tool for the creation, querying and visualization of phylogenetic databases. We demonstrate the functional capabilities and strengths of our system by recreating the *Tree of Life* database in our system and performing queries that are not possible in the original *Tree of Life* database.

Keywords

Phylogenies, tree and semi structured data, visualization, query language, information retrieval, relational database, web

1. INTRODUCTION

Research into genomics is producing an unprecedented amount of data. Consequently, we are witnessing a proliferation of databases or data repositories all around the world. These databases allow the researchers to access and analyze data with the tools they support. New, more accurate, and more efficient tools and techniques are being developed to analyze, interpret and investigate these data. In order to be useful, these data must be indexed and stored, which leads to the issue of effective data analysis and abstraction that still poses a formidable challenge. From this point of view, the creation and development of advanced database technologies for storage, curation, retrieval and analysis of biomedical data has become extremely important in all biomedical disciplines. As genomics and biology become more of a data driven science rather than the traditional workbench and test tube based discipline, new developments will both generate and require volumes of data far more vast than what these fields are currently prepared to utilize and assimilate. Thus, it behooves us to prepare for envisioned challenges by preparing a technological base which is equipped to handle such rich and new data with their own unique requirements and limitations.

Closely aligned to genomics, a relatively new discipline, phylogenetics (the science of studying structural and behavioral relationships of organisms), has been instrumental in discovering new relationships and explaining unknown science, and is becoming increasingly popular as the study of phylogenies often bear directly on basic understanding in molecular biology or of human health issues. For example, phylogenetic techniques have been used to trace contact histories for infectious diseases [14, 5] and identify geographic origins of new outbreaks, as in the case of *West Nile Virus* [15], and even the timing of new introductions [7], suggesting their broad explanatory power in epidemiology [1]. Phylogenetic trees depict the hierarchical pattern of common ancestry of species, genes, sequences or other entities (taxa). These trees capture information that are vital in the analysis of molecular sequence variations. In order to study phylogenies and to explore their predictive properties, large-scale tree databases have been created using various formats (e.g., *TreeBase* [3], *Tree of Life* [8], *RDB* [9], *Green Plant Phylogeny Research Coordination Group* [4]). At the same time, work is advancing on the synthesis of extremely large phylogenetic trees from these large scale databases of smaller trees. The quality of diagnostic predictions strongly depends upon the completeness of a set of samples of a particular group of organisms or genes, and the correctness of the connectivity information between the taxa. Therefore, precise navigation and querying tools to interact with the tree structures and the structure of relationships between trees is critical.

Past experiences with genomic and phylogenetic repositories and the tools for accessing and analyzing the data during the last decade or so serve as important lessons for professionals working with these dynamic fields. In the light

.

of the observations and recommendations made in the documents [13, 12], with a few not so notable exceptions, the experiences can be *partially* summarized as follows:

- Accessing data through Web-based forms are limiting and often frustrating simply because they do not allow queries outside the scope of the interface, and because they force the users to follow too many predefined navigational steps or pointers.
- The lack of a suitable query language for genomic or phylogenetic databases acts as a barrier in extracting biomedical information from various databases in a flexible way.
- It is simply impossible to make databases interoperable because developing the technology (such as middle layer, wrappers, mediators, etc.) are either too expensive or too complicated.

In this paper, we address the issue of representing, querying and visualization of phylogenetic databases using traditional technologies. Specifically, we aim to answer the question if such databases can be developed in conventional ways (perhaps with extensions) without compromising the visualization and browsing aspects of their vast contents, features that biologists find useful and attractive. We answer positively by developing a mapping of the University of Arizona's acclaimed *Tree of Life* Web-based database to a traditional database and by developing a Web-based graphical query interface that uses SQL and a relational database as underlying query and storage engines. The resulting system is more robust, and supports enhanced visualization and improved query capabilities. It also now makes it possible to interoperate other genomic and phylogenetic databases with our database exploiting existing techniques for interoperability of relational databases.

1.1 Paper Organization

The remainder of the paper is organized as follows. In section 2, we introduce of the *Tree of Life* database along with a discussion of its structure and properties that serve as the representation and querying requirements of our target database. The mapping process is presented in section 3 followed by a discussion on design issues and systems architecture in section 4, and a description of the systems implementation in section 5. We present some performance results related to the mapping of the *Tree of Life* database in section 6 before we summarize our discussion in section 7.

2. THE TREE OF LIFE DATABASE

Tree of Life (TOL) is one of several recently created phylogenetic databases at the University of Arizona for the study of phylogenies and evolution of organisms. Unlike most of its siblings, TOL is Web-based and thus, can be queried only by means of Web search engines. It also provides a hierarchical structure that a user or a crawler could navigate one node at a time. Researchers in various parts of the world contribute to this database by remotely adding a sub tree in the form of an HTML document. All nodes are reviewed by the peers for correctness and accuracy, and are connected via hyperlinks rooted at the TOL home page at the University of Arizona.

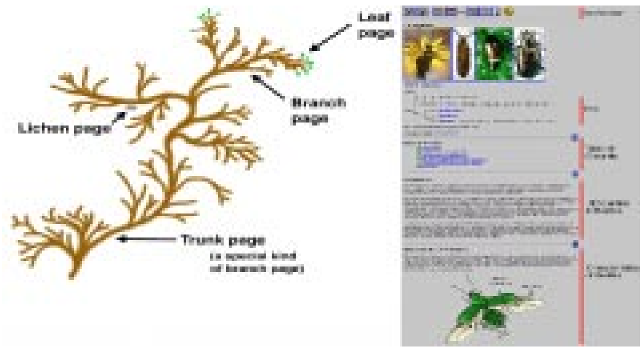


Figure 1: *Tree of Life* database tree hierarchy and node structure.

The TOL database structure is essentially a huge hierarchical tree of hyperlink connected Web documents. Each document in the hierarchy can be considered as a node of the tree that follows a loosely defined format. Every child node is a sub group and a specialization of a living organism of the parent node. For example, the species *Homo Sapiens* is a child node of genus *Homo*, which is a child node of the family *Hominidae*, and so on. In other words, these are ordered from kingdom to species with intermediate classes, namely, phylum, class, order, family and genus. However, the actual classification is more elaborate and often-times debated (because archaeological and genomic evidences often suggest contradictory classifications). We will discuss some of the representation details and the influence of the classification scheme on the representation shortly.

The nodes in the TOL structure are classified in the following four categories as shown in figure 1, i.e., *trunks*, *branches*, *lichens*, and *leaves*. The trunk and branch pages are similar except that a branch page contains the leaf pages. Usually, a branch page contains information for a particular class of organism such as Metazoa, or Homo, etc., and may have links to the sub tree rooted at that class. On the other hand, the trunk pages usually contain links to several branch pages. Lichen is a specialization of a class at any level that leads to a species that can be reached directly from that page. In other words, a lichen page is a node that describes a species of a given class.

Navigation through the entire structure is facilitated with several buttons in every page. Users can visit the nodes of the tree by clicking these buttons to reach the next taxon, the previous taxon and to dive down to the immediate lower node in the classification (the down button) hierarchy or to the parent class (the deep button). Note that there may be several intermediate nodes between a class and its parent class.

3. MAPPING TOL STRUCTURES TO A RELATIONAL DATABASE

The HTML document format of every page and its contents provide enough information for the design of a mapping function from TOL to a relational database. In general, every page contains pictures of species belonging to that class, the sub tree under that class (only one level deep), references to documents that are relevant for that class, discussions, and other relevant information and links.

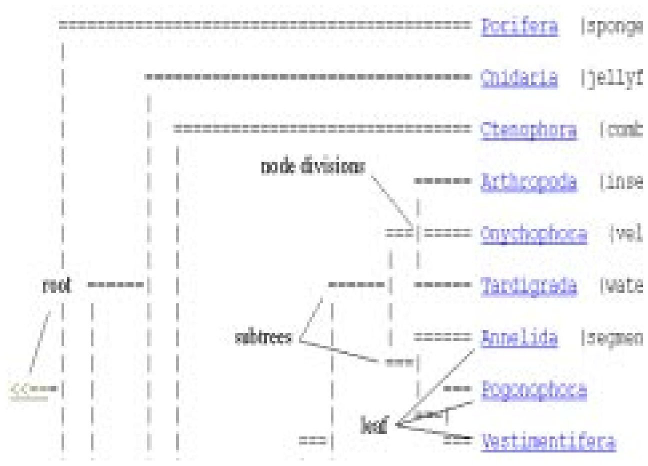


Figure 2: Textual representation of a tree in a page.

The mapping process is divided into several steps and involves a mixture of manual and automated activities. In the following sections, we discuss two major components of these steps.

3.1 Understanding TOL Page Structures

Before we can proceed to map the TOL database into a relational structure, it is necessary to recognize the page structure of every node and develop an accurate understanding so that a relational structure can be designed based on the page template.

As mentioned previously, in addition to several descriptive information, each node page contains a textual diagram of the one level deep sub tree rooted at that node. An example of such a tree is shown in figure 2. Since these are trees represented using text, it is difficult for any existing automated agent to understand the structure. It is however possible to identify important regularities in such trees and develop a specialized agent for automated mapping. For example, a root is denoted by the symbol '`<<==`', and its children are denoted by line drawn with symbols '`|`' that leads to branches represented with lines drawn with the symbol '`=`'. Every branch is labeled (marked as leaf in figure 2) and potentially is a hyperlink.

The structures represented in such trees have special significance that must be preserved, e.g. all the branches have the same parent, but 'closeness' of branches are determined by their position in the tree. Consider the sub tree shown below rooted at node *X* (node *X* not shown). In this tree there are four nodes (leaves) *A*, *B*, *C* and *D*, all children of node *X*. However, nodes *B* and *C* are at a lower level than the nodes *A* and *D*. Hence, the similarity between *B* and *C* are higher than that between *A* and *B*. This 'closeness' is maintained in the tool, as we describe later.

```
    |====== A
    |
    |   |== B
<<==|===|
    |   |== C
    |
    |====== D
```

The tree structure in each page is described using HTML `pre` tags, thus it is identifiable. The algorithm we have developed, presented below, plays a critical role in developing a complete and automated mapping process. Any identified tree can be processed using this algorithm to construct an actual Java `JTree` tree structure.

Algorithm 1. Text2TreeConverter
Input: text describing the tree to convert
Output: a `JTree` tree structure

```
JTree Text2TreeConverter(String text)
{
    matrix=buildMatrix(text);
    root=findRoot(matrix);
    tree=new JTree();
    buildTree(root,matrix,tree)
    return(tree);
}

JTree buildTree(root,matrix,tree)
{
    //identify all the branches of the root
    branches[]=findBranches(root,matrix)
    for(every branch in branches[])
    tree.addChild(buildTree(branch,matrix,tree))
}
```

As the reader can see, to build a tree we first need to create a matrix containing all the characters in the text. Using a matrix, we can move around the tree using coordinates like `x` and `y`, or columns and rows. The building of the matrix is straightforward, all the lines (separated by end-of-line characters) in the text are mapped to a separate row in the matrix, and in order for the matrix to have the same number of columns for each line, we calculate the longest line and pad the others with spaces until the length of all the lines reach the length of the longest one.

Now we can identify the root in the first column using the character `<` (or `=` in the case of the root of all the trees). The root is expressed using coordinates `(x,y)`. We start building the tree recursively identifying the branches for the root first and then treat each branch as a new root for a subtree. In general, is natural to use recursive algorithms when working with trees. We will see that in order to store a tree in a database we will also use a recursive algorithm, as explained later. To identify the branches we use the `|` character and then we move along the vertical axis in the matrix trying to identify new nodes. The process ends when we reach a leaf (identified by a `=` character and then a space, as shown in figure 2).

3.2 Crawling the TOL Structure

Using the algorithm presented in the previous section as a backbone, a Web crawler is developed to parse the TOL structure and gather necessary information to construct the relational structure. As the structure is crawled, pertinent information is stored in a relational database with the structure shown in Figure 3. In this figure, the primary keys are identified using bold letters. The schema contains four tables `pageInformation`, page, `pageReference`, and `treeNode`. The relationship cardinalities among these tables is shown below.

The description of the tables are as follows. The table `treeNode` in Table 1 represents a node of the tree. The `nodeID` attribute is the primary key, the `name` represents the name of a group, the `description` is the example species of the organism that belongs to this node (class) and `parentID` is a foreign key to the table `treeNode` itself that represents the parent of this node. Hence, these relationships are recursive, since a parent of the node is another node in the tree.

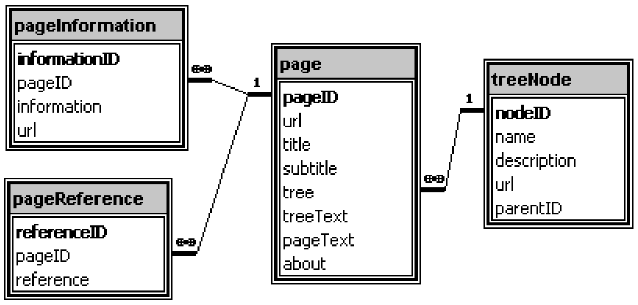


Figure 3: ER Diagram for the relational structure.

Table 1: Relational scheme of the `treeNode` table.

Name	Type	Comment
`nodeID`	`long integer`	PK
`name`	`string`	
`description`	`string`	
`url`	`string`	
`parentID`	`long integer`	FK to `treeNode.nodeID`

Table 2 shows the scheme for the table `page`. This table has all the information pertaining to a page in the TOL database. The `pageID` attribute is auto generated and identifies the page as its primary key. The `url` attribute is the web address where this page is located, `title` and `subtitle` describe the name of the group and example organism, `tree` is a foreign key to the `treeNode` table and relates to the information of the node, `treeText` stores the tree information contained in this page, `pageText` stores the characteristics and information of the group and the `about` records information such as the page creation date, the creator of the page, etc.

The `pageInformation` scheme is shown in Table 3. In this table, information related to links that have additional information about a specific class is recorded. The primary key of this scheme is the attribute `informationID`. `pageID` is a foreign key to the table `page`, and the `information` attribute represents the title of the information content.

The scheme of the last table `pageReference` is shown in Table 4. This table contains the references contained in a page. A page may contain several references. In this table, `referenceID` is the primary key while `pageID` is the foreign key to the table page where it has been cited. Finally, `reference` is the explicit reference text.

Once the information is stored in the database, a suitable user interface can be developed for users to query and visualize its contents. We will address this issue in a later section.

Table 2: Relational scheme of the `page` table.

Name	Type	Comment
`pageID`	`long integer`	PK
`url`	`string`	
`title`	`string`	
`subTitle`	`string`	
`tree`	`long integer`	FK to `treeNode.nodeID`
`treeText`	`string`	
`pageText`	`string`	
`about`	`string`	

Table 3: Relational scheme of the `pageInformation` table.

Name	Type	Comment
`informationID`	`long integer`	PK
`pageID`	`long integer`	FK to `page.pageID`
`information`	`string`	

Table 4: Relational scheme of the `pageReference` table.

Name	Type	Comment
`referenceID`	`long integer`	PK
`pageID`	`long integer`	FK to `page.pageID`
`reference`	`string`	

4. SYSTEM ARCHITECTURE AND DESIGN ISSUES

The architecture for our system is presented in Figure 4. The major components in our system are the modules *Property Identification Module* and the *Navigation Module*. The other two modules are *Visualization Module* and *Database Manager*. We now present a brief discussion on each of the modules.

4.1 Navigation Module

The navigation module is composed of two agents responsible for the interaction between the system and the Internet. The first agent, the *page retriever*, is responsible for establishing network connections for a given URL. The *page retriever* is also responsible for building appropriate URLs based on relative URLs found in the pages, i.e., if a page has a relative (local) URL as a link then the page retriever agent will use the page information (`base` tags and page domain) to build an absolute URL. When a network connection is established, the agent starts retrieving the HTML text from the page. The *page retriever* agent is also used to automatically navigate through the hierarchical structure of the page to follow links to the internal pages in the web site provided by the *external link identification* agent. Because the number of links to crawl usually grows exponentially to the crawling depth, the *page retrieval* agent must be limited only to crawling pages in the application domain using a predefined crawling (which is implemented as a system property that can be adjusted, if necessary, as shown in figure 5).

The second agent, the *external link identification agent*, uses a `DOM` (Document Object Model) structure to identify external links to other pages in the application domain (identified by attributes in the HTML code). In general, all the leaf nodes in a sub tree contain links to other sub trees (other pages) in the TOL structure. All these links are encapsulated as a list of new pages that will be passed to the *page retriever* so that the contents of the new pages can be retrieved and parsed.

4.2 Parsing Module

Once the page contents are retrieved, the *parsing module* is activated. The *parsing module* contains an HTML parser that is able to identify the different HTML tags and their

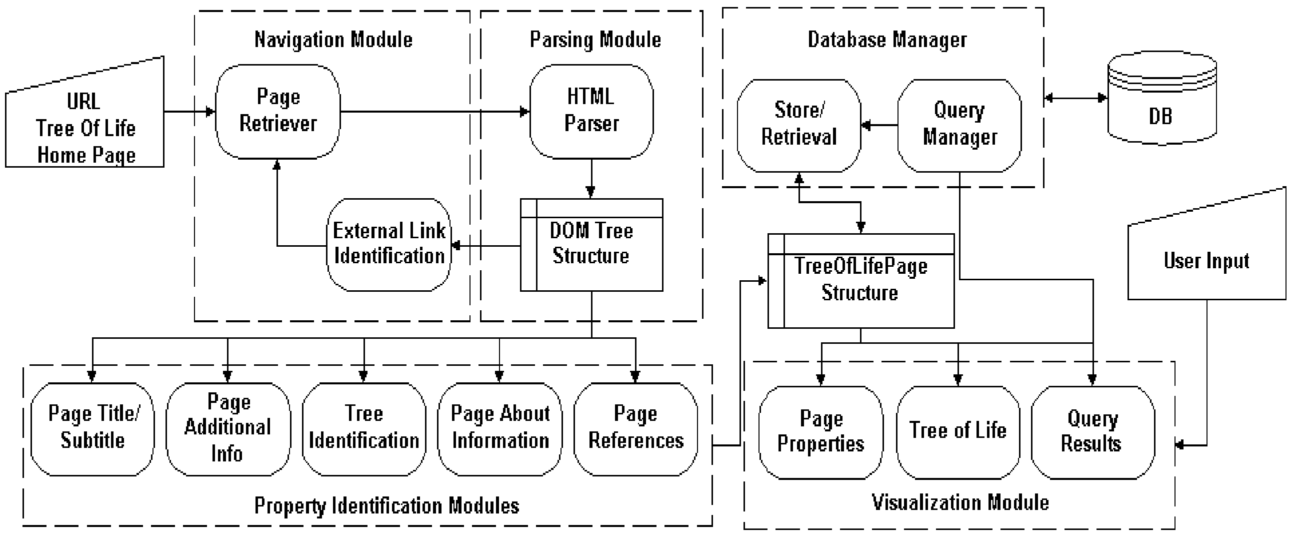


Figure 4: System architecture.

contents, and to map the HTML page into a tree called the DOM (Document Object Model) tree. This tree structure is described in W3C standard [2] and is used to describe the structure of an HTML page. The use of DOM structure and construct greatly simplifies the identification of the elements in an HTML page. While there are several HTML parsers available, most of them are designed to parse XHTML/XML documents and thus cannot be used to parse the TOL database HTML pages since most of these documents are not well formed, a critical requirement for those tools to be effective [10]. However, `HTML Tidy`, also a W3C standard [2] tool can ignore such discrepancies and is used in our system. In addition to the parsing and creation of a DOM structure, `HTML Tidy` is able to identify and fix most of the common problems present in HTML pages such as omission of closing tags when required, etc. Another good feature of `HTML Tidy` is that it can use any parser as long as the parser is able to produce a DOM structure that represents the page (most of the parsers do).

4.3 Property Identification Module

The *property identification module* receives a DOM structure representing a specific page and identifies its attributes (title and references, metadata contained in the page, etc.) as well as page information such as description of the species represented in the page, family, descendents in the taxonomy, etc. This module is comprised of specific submodules, each of which is responsible for the identification of a specific property in the page. The use of an array of sub modules supports extensibility in case new properties that are not currently supported need to be recognized. Our system currently supports five sub modules for the identification of the following five properties listed in Table 5.

The input to the sub modules is a DOM structure created by the *parsing module* for a page. These sub modules use the tag template presented in table 6 for the identification of specific properties in the page.

4.4 Visualization Module

Users interact with the system through the *visualization module*, the query interface of our system. The primary function of the visualization module is to provide the user with a graphical representation of the pages and the illusion of the TOL structure. The query interface is shown in the figure 6 in section 5.

There are basically two choices in mapping and querying the TOL database. In our application, we have used our crawler to map the TOL database offline and store all the information in the relational database for efficiency reasons. In this way, the querying of the relational database is extremely fast. The compromise is that the currency of information is lost in the event of an update in the source TOL database, a common, albeit rare, practice. Our other option was to crawl the pages at query time and store the information as we crawl. In this way, the mapping could be done at query time. As we need to map the entire database in any event, an offline retrieval seemed logical.

Table 5: Submodules for identification of page properties.

Module	Property
Page Title/Subtitle	The title and optionally, a subtitle (e.g. title: Animals, subtitle: Metazoa)
Additional Info	Every page has an "Additional Info" section with external links in the web
Tree Identification	This submodule identifies the text that describes the tree for the current page
About Information	Every page has an "About" section with information about the author of the page
Page References	Every page has a "Reference" section with references used by the contents of the page

4.5 Database Manager

The *database manager* module is responsible for inter-

Table 6: Template for property identification based on HTML tags.

Property	HTML Tag
Tree	`<pre>`
References	`<pre>`
Title	`<h2>`
Subtitle	`<h3>`
Section boundaries	Between two `<hr>` tags
Links	`<a>`

acting with the data repository used to store the mapped information. The database manager makes use of the JDBC wrappers supported by Java to abstract the database operations from the type of database in use. The *database manager* module has two sub modules: *Store/Retrieve* to store and retrieve the pages parsed, and the *Query Manager* to handle queries against the data repository using the query forms in the query interface. The *query manager* is just a general module that receives as input an SQL query and returns the results using a multicolumn table that is later displayed by the *visualization module*. The database manager uses the relational structure described in section 3 as the database schema to store the page information.

5. SYSTEM IMPLEMENTATION

This section addresses issues related to the systems implementation. We present a brief discussion about the language and platforms used to develop and deploy the tool, as well as some details about the main two structures used by the tool: the DOM and the TreeOfLifePage structures. We also explain some of the features presented in the tool like the configuration panel (to configure database access and inherent addresses to the source of the information) and the query manager (to perform queries against the data collected).

5.1 Language and Platform

The tool was developed in Java for portability, abundant predefined library and tool support, and because Java supports better abstraction capabilities. As an example, we used libraries such as the `util` library for container structures like vectors and dynamic arrays, the AWT and Swing libraries for GUI development, and JDBC library for database access, to name a few [11]. The choice of Java was also motivated by the fact that future versions of the system may now include the same functionality in a Java Applet so that we can run the system from any Java-enabled web browser like Netscape or Microsoft Internet Explorer.

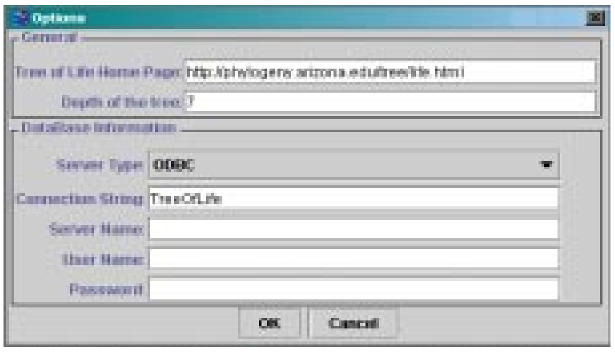
5.2 Data Structures

The two most important data structures used are the DOM structure and the TreeOfLifePage structure, also identified in figure 4. As already mentioned in a previous section, the DOM structure is used by the parsing libraries to represent an HTML page as a hierarchical tree structure. The template in table 6 is used in conjunction with the DOM structure to analyze and understand a page according to its structure, and use its contents as needed.

The DOM structure is based on a Node structure, representing all the nodes in the hierarchy. HTML tags are represented by Element structures (a subclass of Node). The library to manipulate DOM structures allows identifying all the tags in the page by name, and for an Element to retrieve its children, or its attributes, as well as the text contained inside a tag (like in the case of the p tag) [10].

The second structure, the TreeOfLifePage structure is used to store all the parsed information relevant to a page. It resembles the database structure presented in section 4. The structure contains data members for each property modeled, and in the case of multivalued attributes like references (a page can contain multiple references) and additional information, it contains dynamic arrays for storing these properties. It is worth noting here that the TreeOfLifePage structure contains a reference to a TreeOfLifeNode structure that represents the root node for the tree described in the page. This node structure contains data fields to describe each node in the tree, including the node name, a description, and a URL (in case of leaf nodes that point to other pages in the TOL structure).

The TreeOfLifePage structure uses a set of utility methods to maintain the structure. Methods like `saveToDB` and `retrieveFromDB` are used by the database manager to store and retrieve a page to/from a database while the methods `addInformation` and `removeInformation` are used to add/remove information found in the "Additional Information" section of each page. The methods `addReference` and `removeReference` are used to add/remove references found in the "References" section of each page. The structure also provides methods to retrieve the set of links present in the tree to be used as input to the navigation module to crawl the tree.



Tree of Life Home Page: http://phylogeny.arizona.edu/tree/life.html
Depth of the tree: 7
Server Type: ODBC
Connection String: TreeOfLife
Server Name:
User Name:
Password:

Figure 5: Configuration options available to the user.

5.3 System Configuration Panel

In order to make the system independent of the underlying data structure and database management system, a set of configuration options are supported that allows it to connect to almost any database manager that supports JDBC. If the database is changed, all the users need is to specify the new database in the configuration options as shown in Figure 5.

For systems with support for ODBC drivers (such as in the case of Microsoft Windows OS), the user only needs to specify the ODBC driver configured for database access in the connection string field, and the ODBC driver takes care of the details regarding security, type of connection, database server, etc. Of course, the user can also specify these parameters as part of the connection in the appropriate fields.

For general options the user can specify the crawling depth used by the navigation module to retrieve new pages for parsing. Due to the potential exponential growth of the tree structures on the Internet, including the in the *Tree of Life* database, we suggest using of a low depth. Otherwise, depending on the type of connection, the process to build the complete tree can be extremely slow. The user can also modify the starting point (i.e. the root of the tree) allowing the construction of partial trees. For example, to build only one branch of the tree of depth six, the user may first build a complete tree up to a depth of three (by specifying crawl depth to be three), selecting only the branches at depth three that are of interest, and then building three more levels from it.

5.4 Query Manager

One of the main goals for this project is to provide the user with extended query capabilities not present in the original source of the phylogenetic tree. The tool provides to the user with query and browsing capabilities as explained in this section. The user interface allows the user to perform queries using different attributes of the page. Currently, four types of queries can be performed in this interface: selection of pages using (i) the title of the page, (ii) the URL of the page, (iii) any keyword in the page text, and (iv) the id of the page in the database. Queries of type (i) and (iii) use substring searches, and multiple strings are ANDed together to constrain the selection. If the query is successful, i.e. at least one page meets the conditions specified, the query manager constructs a list of page IDs identifying the pages displayed as hot links. The user can browse the information in each of these links by clicking the page ID needed. When the user selects a page ID, a new `TreeOfLifePage` structure is created, and its contents retrieved from the database using the database manager module. The page is then displayed in the user interface for the user just like a web document.

Additional information about the query capabilities for this tool can be found in [6], where a the framework for a graphical query language is defined to query tree databases (specifically, phylogenetic databases like the *Tree of Life* database described in this paper).

5.5 User Interface

The interface is developed using entirely the Java Swing UI library. As shown in figure 6, the interface is divided in different panels with multiple tabs to organize different information to be presented to the user. Most of the elements present properties (e.g., nodes in the tree) that the user can see using the properties panel. Figure 6 also depicts how the information is stored in the database, how the queries can be performed, and the how results are viewed. It is also possible to inspect/query the underlying database and its meta data. The user can also use the address bar to type in a URL, parse it, and inspect its structure, if needed.

6. SYSTEM VALIDATION

We have tested our system on SUN Solaris and PC Windows 2000/Millenium Edition. We were able to successfully run the system in both these platforms without any need for compilation of either system. We have used three different types of database management systems such as Microsoft Access, Microsoft SQL Server and Oracle 8i with this system without any complication.

As the graph in Figure 7 shows, the time required to parse and map the TOL structure increases quite dramatically

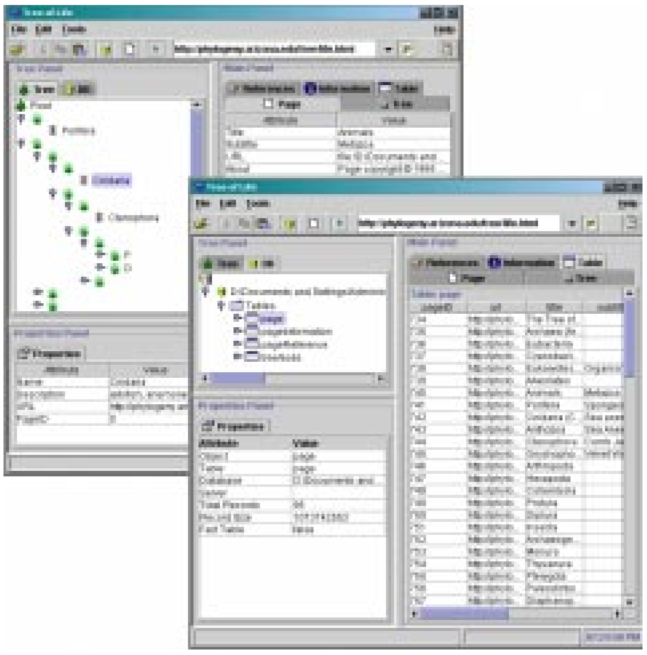


Figure 6: System/User Interface for TOL Relational Database.

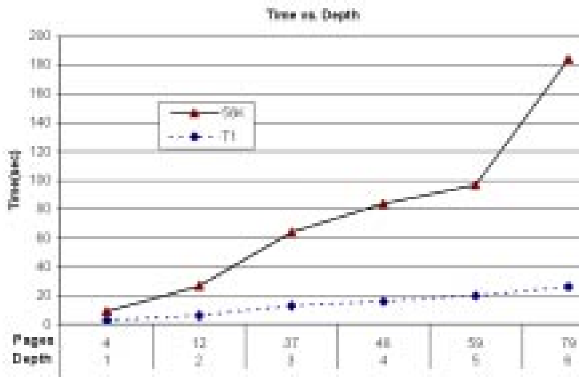


Figure 7: Time vs. Depth (56K & T1).

with the increase in crawl depth. The mapping performance for a 56K and a T1 Internet connection on Microsoft Access DBMS are shown for multiple crawl depth. The number of pages parsed and mapped at each depth is also shown. Notice that the number of pages parsed is a constant for a given crawl depth.

7. SUMMARY AND FUTURE WORK

We have presented a system for mapping Web-based phylogenetic databases into regular relational databases so that they can be queried using improved database query capabilities, yet, can be visualized using graphical interfaces such as the Web. We believe that the framework presented in this paper can be generalized to map most such databases for the purpose of interoperability and scientific analysis.

Since XML is increasingly becoming a language of choice for many applications, our current research is focused on developing a mapping technique for phylogenetic databases into XML. The main idea is as follows. We are working on developing a simple DTD schema that will match the ER model presented in previous sections, enabling us to map the phylogenetic information in to XML structures:

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT phylogeny (page)>
<!ATTLIST phylogeny
name CDATA #REQUIRED
url CDATA #REQUIRED
description CDATA #IMPLIED>
<!ELEMENT page (treeText, pageText?,
  about, reference*, information*, page*)>
<!ATTLIST page
url CDATA #REQUIRED
title CDATA #REQUIRED
subtitle CDATA #IMPLIED
description CDATA #IMPLIED>
<!ELEMENT treeText (#PCDATA)>
<!ELEMENT pageText (#PCDATA)>
<!ELEMENT about (#PCDATA)>
<!ELEMENT reference (#PCDATA)>
<!ELEMENT information (#PCDATA)>
```

As an illustration, we present below an XML fragment that encodes the first two levels of the *Tree of Life* database:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE phylogeny SYSTEM "phylogeny.dtd">
<phylogeny name="The Tree of Life"
  url="http://phylogeny.arizona.edu/tree/life.html">
  <page
    url="http://phylogeny.arizona.edu/tree/life.html"
    title="The Tree of Life Project Root Page">
  <treeText>...</treeText>
  <pageText/>
  <about>David R. Maddison</about>
  <page url="eubacteria.html"
    title="Eubacteria"
    description='"True bacteria", mitochondria...'>
    <treeText>...</treeText>
    <pageText>...</pageText>
    <about>Gary J. Olsen</about>
  </page>
  <page url="archaea.html"
    title="Archaea (Archaebacteria)"
    description="Methanogens, Halophiles, ...">
    <treeText>...</treeText>
    <pageText>...</pageText>
    <about/>
  </page>
  <page url="eukaryotes.html"
    title="Eukaryotes (Eukaryota)"
    description="Protists, Plants, Fungi, Animals, etc."
    subtitle="Organisms with nucleated cells">
    <treeText>...</treeText>
    <pageText>...</pageText>
    <about>David J. Patterson</about>
  </page>
</page>
</phylogeny>
```

Once the phylogenetic tree is converted to XML, we will be able to perform more advanced queries using XML query languages like XQL and XPath expressions, or even XSL templates for advanced display and formatting [2]. As an example, an XPath expression to retrieve nodes with title 'Animals', at any level, would be: `//page[@name='Animals']`. We are also working on a API for advanced query capabilities, some of the API commands include the `parent()` and `child()` to query for direct hierarchy relationships in the tree, as well as the `predecessor()`, `succesor()` and `root()` commands for indirect hierarchy relationships.

However, the system presented in this paper is a part of a larger project at Mississippi State University that aims at developing a framework for interoperability of genomic and phylogenetic databases. As a next step to this project, we plan to develop a language for phylogenetic databases, called the *phylogenetic query language* (PQL) to query tree structured data and their intricate properties. A preliminary report on this language may be found in [6].

8. REFERENCES

[1] S. N. e. a. C. Holmes, Paul Bollyky. *New Uses for New Phylogenies*, chapter Using phylogenetic trees to reconstruct the history of infectious disease epidemics, pages 169–186. Oxford University Press, 1996.

[2] W. Consortium. Internet Address: http:://www.w3.org.

[3] B. P. et al. TreeBASE. Internet Address: http://herbaria.harvard.edu/treebase.

[4] G. P. P. R. C. Group. Deep Green. Internet Address: http://ucjeps.berkeley.edu/bryolab/greenplantpage.html.

[5] D. M. Hillis and J. P. Huelsenbeck. Support for dental HIV transmission. *Nature*, 369:24–25, 1994.

[6] G. M. Jamil Hasan and M. Teran. Querying phylogenies visually. In *Proceedings of the Second IEEE BIBE Symposium*, 2001.

[7] G. F. G. e. a. K. McGuire, E. C. Holmes. Tracing the origins of Louping Ill virus by molecular phylogenetic analysis. *Journal of General Virology*, 79:981–988, 1998.

[8] D. Maddison and W. P. Maddison. The Tree Of Life: A multi-authored, distributed internet project containing information about phylogeny and biodiversity. Internet address: http://phylogeny.arizona.edu/tree/phylogeny.html, 1998.

[9] L. T. e. a. Maidak BL, Cole JR. The RDP-II (Ribosomal Database Project). Internet Address: http://rdp.cme.msu.edu/html/. Nucleic Acids Res 2001 Jan 1;29(1):173-4.

[10] B. McLaughlin. *Java and XML*. O'Reilly and Associates, Inc., Sebastopol, CA, 2000.

[11] P. Naughton and H. Schildt. *The complete Java reference: Java 2*. Osborne/McGraw-Hill, Berkeley, CA, 3rd ed. edition, 1999.

[12] W. G. on Biomedical Computing. The biomedical information science and technology initiative. Technical report, National Institutes of Health, June 1999.

[13] N.-U. D. W. on Phyloinformatics. Research needs in phyloinformatics (draft report). Technical report, National Science Foundation, October 2000.

[14] C. Y. Ou. Molecular epidemiology of HIV transmission in a dental practice. *Science*, 256:1165–1171, 1992.

[15] V. D. e. a. R. S. Lanciotti, J. T. Roehrig. Origin of the West Nile virus responsible for an outbreak of encephalitis in the northeastern united states. *Science*, 286:2333–2337, 1999.