CDAOStore: A Phylogenetic Repository Using Logic Programming and Web Services

Brandon Chisham, Enrico Pontelli, Tran Son, Ben Wright
Department of Computer Science
New Mexico State University
{bchisham | epontell | tson | bwright}@cs.nmsu.edu

January 14, 2011

Abstract

The *CDAOStore* is a portal aimed at facilitating the storage and retrieval of data and metadata associated to studies in the field of evolutionary biology and phylogenetic analysis. The novelty of CDAOStore lies in the use of a semantic-based approach to the storage and querying of data. This enables CDAOStore to overcome the data format restrictions and complexities of other repositories (e.g., TreeBase) and to provide a domain-specific query interface, derived from studies of querying requirements for phylogenetic databases.

CDAOStore represents the first full implementation of the *EvoIO stack*, an inter-operation stack composed of a formal ontology (the Comparative Data Analysis Ontology), an XML exchange format (NeXML), and a web services API (PhyloWS). CDAOStore has been implemented on top of an RDF triple store, using a combination of standard web technologies and logic programming technology. In particular, we employed Prolog to support some of the format transformation tasks and, more importantly, in the implementation of several of the domain-specific queries, whose structure is beyond the reach of standard RDF query languages (e.g., SPARQL). CDAOStore is operational and it already hosts over 90 million RDF triples, imported from TreeBase or submitted by other domain scientists.

1 Introduction

The explosive growth of stock-piled information in the biological and earth sciences presents a wealth of opportunities for expanding bioinformatics-based analyses with respect to both the amount of data incorporated and the diversity of data types and sources to be integrated. While large-scale or integrative analyses of such data may use generic methods of machine learning, there is a theory-based comparative approach to the analysis of diverse types of biological data, in which the similarities and differences between compared things are interpreted as *evolved differences* that have arisen by a process of descent-with-modification from common ancestors. This evolutionary comparative approach, used throughout biology and paleobiology, depends fundamentally on "phylogenetic trees" representing paths of descent. While powerful tools exist for the inference of phylogenies, and while evolutionary approaches are increasingly recognized as effective, the lack of interoperability in tree-based data and services hinders large-scale and integrative analyses.

To address this overarching problem, a collaboration among computational scientists and evolutionary biologists has been established—within a working group (EvoInfoWG) sponsored by the National Evolutionary Synthesis Center—leading to the development of an *interoperation stack* (*EvoIO Stack*) for the exchange of evolutionary structures. The EvoIO Stack comprises of *(i)* an ontology for the description of data (the *Comparative Data Analysis Ontology*), *(ii)* a data exchange format (*NeXML*), and *(iii)* a web service interface for data querying (*PhyloWS*).

The use of components of the EvoIO Stack has been gaining momentum—e.g., NeXML is now a supported format by several analysis tools (e.g., Mesquite[1], DendroPy[2], DAMBE[3]), CDAO and PhyloWS are supported by TreeBase[4]), the largest repository of phylogenetic trees. While these efforts adopt components of the EvoIO Stack as adds-on to the existing features of the tools and repositories, the EvoInfoWG has initiated the development of a novel data repository completely built around the EvoIO Stack and capable of providing forms of access to evolutionary data that are beyond the syntactic and relational access forms offered by traditional repositories, like TreeBase. The

[1] `http://mesquiteproject.org/mesquite/mesquite.html`

[2] `http://packages.python.org/DendroPy/`

[3] `http://dambe.bio.uottawa.ca/dambe.asp`

[4] `http://www.treebase.org`

CDAOStore is the first effort in this direction.

CDAOStore is a triple store that implements the complete EvoIO Stack. As a triple store, it maintains a semantic-based repository for phylogenetic data, as RDF triples; it provides the ability to import and export the content in NeXML and other commonly used formats, and it supports querying through a PhyloWS web service interface. CDAOStore offers a domain-specific querying interface supporting classes of queries relevant to phylogenetic investigation, as identified by domain experts. The implementation of the CDAOStore combines established web services and ontology technologies with Prolog; logic programming is employed to provide an effective implementation of several classes of domain-specific queries, which are beyond the reach of traditional ontology-based query languages (e.g., SPARQL [17]). In particular, Prolog enables a very elegant encoding of operations that manipulate collection of phylogenetic trees, performing selection of branches and transitive closures.

2 Background

2.1 Phylogenetics and Interoperation

Phylogenetic trees (a.k.a. phylogenies) have gained a central role in modern biology. Trees provide a systematic structure to organize evolutionary knowledge about diversity of life. Trees have become fundamental tools for building new knowledge, thanks to their explanatory and comparative-based predictive capabilities. Evolutionary relationships provide clues about processes underlying biodiversity and enable predictive inferences about future changes in biodiversity (e.g., in response to climate or anthropogenic changes). Phylogenies are used with increase frequency in several fields, e.g., comparative genomics [3], metagenomics [23], and community ecology [21]. Comparative phylogenetic analyses have the potential to have enormous impact in even more diverse research areas such as paleobiology [10] or epidemiology [18], linking evolutionary biology with the earth sciences and medical communities. Realizing the full potential of repurposing phylogenies across research communities and disciplines requires interoperability and accessibility of phylogenetic data, as well as interpretability of their meaning.

The major obstacle hindering broad availability and repurposing of phylogenies has been, for a long time, the lack of effective standards and a

community-driven process for adopting and extending them. Existing file formats allow for representation of trees using a simple string and also the molecular or morphological character data used to infer the tree. There are no widely accepted standards for annotating tips, internal nodes or branches, and different applications have adopted unique methods for modifying the tree strings, meaning that annotations from one program may generate errors or be misinterpreted when import into another program. Other types of data/metadata, such as descriptions of evolutionary models or metadata annotations for provenance, have not seen any attempts at standardization.

2.2 The EvoIO Stack

The *EvoIO Stack* [19] (Fig. 1) has been proposed as a platform that coherently combines support for exchange of data and their semantics and predictable programmatic access. The EvoIO Stack is seeded with a triplet of emerging interoperability standards, developed by the EvoInfoWG—NeXML, CDAO, and PhyloWS.

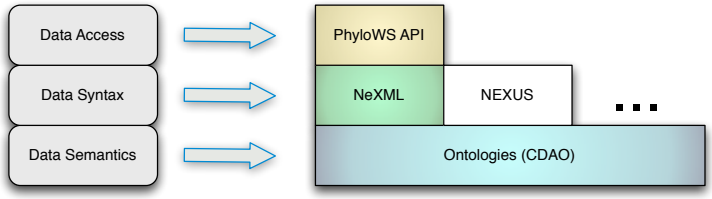

Figure 1: The EvoIO Stack

NeXML [20] is an emerging exchange standard for phylogenetic trees, data matrices, and arbitrary metadata. NeXML is an XML schema for comparative biology that draws on the successful high-level block structure of NEXUS [12], but takes advantage of widespread support for XML, and harnesses the W3C-proposed RDFa standard to embed semantically rich metadata in a way that can be extracted by general purpose tools developed for the semantic web.

The *Comparative Data Analysis Ontology (CDAO)* [16] provides a formal ontology for describing phylogenies and their associated character state matrices. CDAO forms the base of the EvoIO stack, defining the semantics for NeXML files. CDAO is implemented as a formal ontology encoded in OWL. It provides a general framework for talking about the relationships

between taxa, characters, states, their matrices, and associated phylogenies. The ontology is organized around four central concepts (see also Figure 2): OTUs, characters, character states, phylogenetic trees, and transitions. The key concepts and their mutual relationships within CDAO are illustrated in Figure 2. A phylogenetic analysis starts with the identification of a collection of *Operational Taxonomic Units* (OTUs), representing the entities being described (e.g., species, genes). Each OTU is described, in the analysis, by a collection of properties, typically referred to as *characters*. The values that characters can assume are referred to as *character states*. In phylogenetic analysis, it is common to collect the characters and associated character states in a matrix, the *character state matrix*, where the rows correspond to the different OTUs and the columns correspond to the characters.

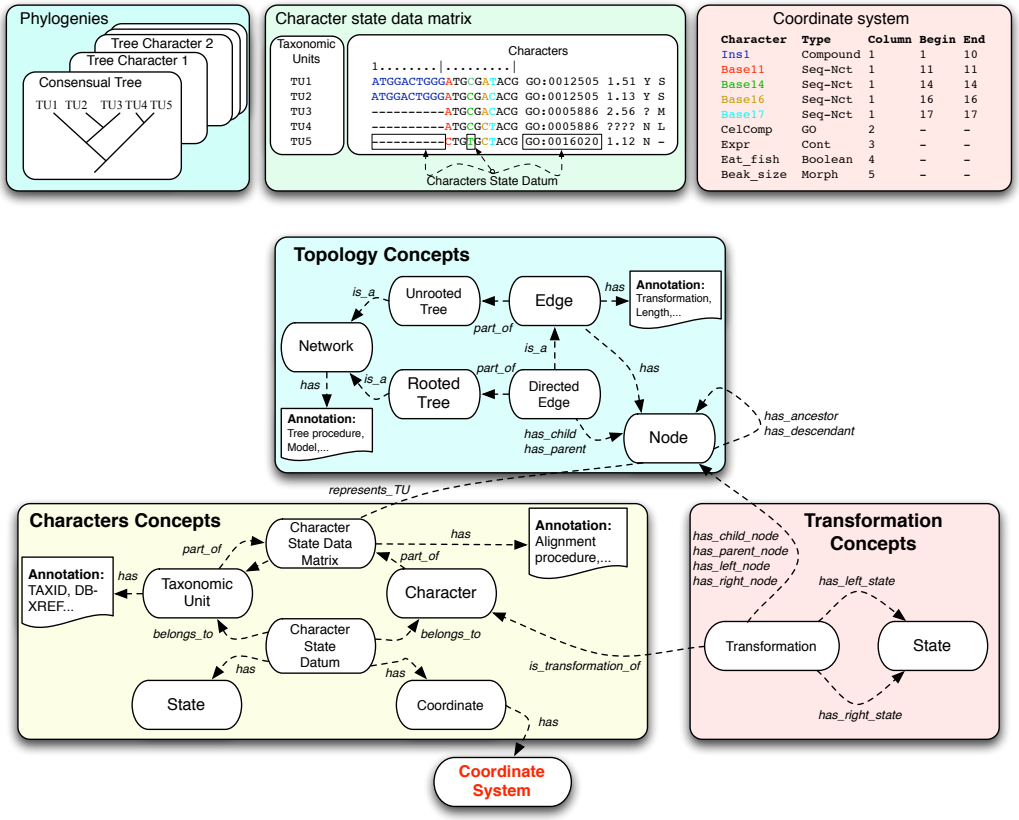

Figure 2: CDAO Core Concepts

Phylogenetic trees and networks are used to represent paths of descent-with-modification, capturing the evolutionary process underlying the considered OTUs. Since evolution moves forward in time, the branches of a tree are typically directed. The terminal nodes are anchored in the present, as they represent observations or measurements made on existing organisms. The internal nodes represent common ancestors, with the deepest node as the root node of the tree. The restriction that each node has at most one immediate ancestor reflects the assumption that evolutionary lineages, once separated, do not fuse (e.g., because of the assumption of reproductive isolation). Branching is considered to be a binary process of splitting by speciation (or gene duplication, in the case of molecular sequences). Even with terminal nodes anchored in the present, it may be impossible to infer the direction of each internal branch, in which case the tree may be referred to as an unrooted tree or as a network. Even the restriction of single parentage maybe occasionally abandoned (e.g., in the case of lateral transfer or reticulate evolution).

As a general framework, CDAO supplies general classes and relations, that can be further specialized to meet the needs of a specific application—e.g., *Beak length* might be defined as a specialization of CDAO's Standard character type.

PhyloWS [9] is a web-services standard for searching, addressing, and accessing phylogenetic trees, data matrices, and their associated metadata in a predictable and programmable way from online phylogenetic data providers. At the physical level, it includes a specification of a RESTful (Representational State Transfer) programming interface to trees and associated data and metadata. The API specification is currently focused on data retrieval, but specifications for data manipulations (such as removing tips, adding an annotation, changing the topology) have been started. All PhyloWS URI's begin with /phylows/ as the standard delimiter. Then based on the phylogenetic information being queried a data structure will be given, such as taxon, tree, or study. This is followed by any specific identifiers needed for the query. For example, `http://purl.org/phylo/treebase/phylows/tree/TB2:Tr3099?format=rdf` is a way to access information from TreeBase using PhyloWS. When this URL is accessed, it returns the tree with the TreeBase ID equal to 'Tr3099' in RDF format.

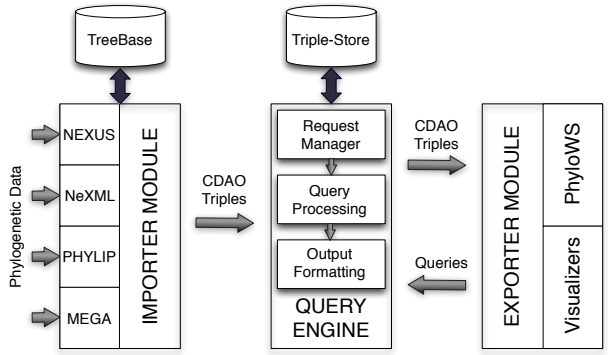

Figure 3: Overall structure of CDAOStore

3 CDAOStore

The CDAOStore is a novel repository and portal aimed at facilitating the storage and retrieval of phylogenetic data; it is the first instance of a repository built on the EvoIO stack. The novelty of CDAOStore lies in the use of a semantic-based approach to the storage and querying of data, building on CDAO for the semantic annotation of data. This approach enables scientists to overcome the restrictions imposed by the use of specific data formats—thus, facilitating inter-operation among phylogenetic analysis applications. This is different, e.g., from the main existing repository for phylogenies, TreeBase, which requires submissions in NEXUS format; there is experimental evidence (e.g., [14]) that, due to the vague specification of NEXUS and the different implementations of this format, data reuse and inter-operation has been hard to achieve, and TreeBase has high rates of rejected submissions due to incorrect data formats (e.g., a study showed that, over a period of time, 9% of the submissions to TreeBase were complete, and 11% of them could not be parsed). Furthermore, the use of a semantic-based repository makes it possible to design and implement more meaningful domain-specific queries.

The overall structure of CDAOStore is illustrated in Fig. 3. The system is organized in three modules. The *Importer* module enables the submission of new phylogenies, data matrices, or metadata for existing phylogenies into the repository. The *Query* module is in charge of supporting all the queries to the repository. The *Exporter* module implements the user interface to the CDAOStore. In the following subsections, we review the functionalities and

implementation of these three modules. The details of the *Query* modules are discussed extensively in the following sections, as this module represents the core of the system.

3.1 Importer Module

The purpose of the Importer module is to import phylogenies and their associated data into the repository, automatically extracting their representations in terms of instances of CDAO. The data importer module can process phylogenetic data encoded in several commonly used data formats—currently, the Importer supports NEXUS [12], NeXML [20], PHYLIP [4], and MEGA [8] formats. The Importer has been realized by developing a semantic characterization of each data format in terms of CDAO concepts and relations. The Importer module includes a permanent link to TreeBase—i.e., the CDAOStore repository provides a semantic mirroring of the complete content of TreeBase, and new updates to TreeBase are immediately reflected into CDAOStore.

The various parsing sub-modules have been developed from scratch, using combinations of C++, Prolog, and XSLT. In particular, Prolog has been used to implement a parser for NEXUS—accommodating for differences in the interpretations of the NEXUS specification [12] through non-determinism, see [7] for details. The Importer maps data from the input files to an object model that mirrors CDAO classes, producing RDF/XML triples that can be deposited in the CDAOStore triple store. The data importer module is also capable of mapping the object model back into any of the acceptable data formats; this enables the use of CDAOStore for conversion among data formats.

3.2 Exporter Module

The Exporter modules provides the user interfaces of CDAOStore. It consists of a Web portal, which allows users to submit and query data through fillable fields, and a series of visualization tools, referred to as CDAO Explorer. CDAO Explorer includes an application for visualizing phylogenetic trees, that provides different visualization formats and the ability to highlight or hide parts of the trees according to user-defined criteria, an application for displaying character data matrices, with capabilities to use color coding to highlight character patterns, and a tool to graphically add annotations to existing phylogenies. CDAO Explorer has been implemented using a com-

bination of Java and the Prefuse framework (`prefuse.org`). Additionally, the Exporter module includes the ability to map phylogenies encoded in the triple store to any of a number of formats (e.g., phyloXML, NeXML, Newick, RDF/XML), which can be piped to other visualization tools, such as Nexplorer 3 [5], which accepts CDAO RDF/XML input, and PhyloBox [6], which accepts phyloXML and Newick representations.

3.3 Query Module

At the core of CDAOStore we find a triple store which collects data and metadata associated to phylogenetic analysis studies. All the data and metadata are stored as RDF triples, encoded using CDAO. The repository itself has been implemented using the RDFlib library (`www.rdflib.net`), a Python-based RDF store which uses a MySQL database to maintain the serialized RDF triples representing instances of CDAO. RDFlib was selected for its simplicity and for the flexibility offered in formatting the output of queries posed to the store. The Query module is articulated into three components—a request manager, a query processor, and an output formatter. We briefly discuss the request manager and the output formatter next, while the central component of query processing is discussed in the next section.

Request Manager: The request manager provides the infrastructure for the PhyloWS web service API and prepares the queries for execution on the triple store. The PhyloWS API is the basis for all the data access features of CDAOStore. The other web components and the CDAO Explorer use PhyloWS to access data.

The URI's of the CDAOStore implementation of PhyloWS are divided into three conceptual parts: **(1)** the address of the store site and path prefix `www.cs.nmsu.edu/~cdaostore/cgi-bin/phylows`, **(2)** a query type (e.g., tree, matrix, msc, nca, size), and **(3)** a parameter list, dependent on the specific type of query. For example, the msc (maximum spanning clade) and nca (nearest common ancestor) query types expect a list of taxon id's separated by '/'. The listing query takes optional limit and offset parameters to paginate results. The size query requires parameters describing a *direction* (greater, less, or equal), a *criteria* (node, internal, or leaf) and a *limit* (a numeral). The following are two sample PhyloWS calls:

- Request the nearest common ancestor between the taxa `Ilex anomala` and `Ilex glabra` in the TreeBase tree with identifier `Tree3099`:

`www.cs.nmsu.edu/~cdaostore/cgi-bin/phylows/nca/Tree3099/Ilex_anomala/Ilex_glabra`

- Retrieve all phylogenies with less than 25 nodes:

  `www.cs.nmsu.edu/~cdaostore/cgi-bin/phylows/size/node/less/25`

The request manager handles parsing, sanitizing, interpreting request arguments, and orchestrating the remaining components. The query variables are extracted from the `QUERY_STRING` and/or the `REQUEST_URI` environment variables. Once all the components of the the URI have been parsed, the information are passed to the Query Processor for execution. This process is implemented by a set of scripts, named according to the following convention: `do_FORMAT_QUERY-NAME_query.sh` where `FORMAT` indicates the output format of the script, and `QUERY-NAME` indicates the type of query handled. In particular, the script `do_query.py` is capable of executing a generic SPARQL query—allowing the users to execute arbitrary queries on the triple store. After processing the arguments, the handler script invokes the Query Processing layer, and communicates the result to the Output Formatter.

Output Formatter: The results produced by the execution of a query are returned to the Exporter for output. For most queries, the formatting is largely determined by the format string given to the query processor, and a query-dependent header and footer. For the queries determining phylogenies, the output is filtered through a post-processing layer to reorder its components and make it easier to be processed by the exporter module—e.g., the components of the phylogeny are exported in an order that resembles a breadth-first traversal of the tree.

4 Domain-Specific Queries

4.1 Querying Biological Phylogenies

CDAOStore maps phylogenies and characters state data matrices to instances of CDAO, and stores them as RDF triples in a triple store. The triple store of CDAOStore is exposed to the users via the PhyloWS web service interface, enabling them to submit SPARQL queries [17]. These queries are processed by the generic SPARQL processor implemented by the RDFlib library.

Nevertheless, the use of a SPARQL interface has several drawbacks; SPARQL is a relatively complex query language, which is relatively inaccessible to the average life scientist. Furthermore, there is ample evidence in the literature on uses of phylogenetic analysis in biological investigations that the type of

queries required are often more complex than what standard SPARQL can provide. In particular, SPARQL is relatively weak in handling queries to hierarchical structures of arbitrary depth, transitive relations, and it offers limited support for aggregate functions.

To address these issues, we developed a *domain-specific* query interface to the triple store, providing classes of queries that have been determined to be of general interest to life scientists using phylogenies in their investigations. The classes of queries have been devised through a combination of focus groups with life scientists and investigation of the relevant literature—the problem of storing and accessing phylogenies has been recognized for quite some time (e.g., [15]) and attacked by various research groups (e.g., the Evolutionary Database Interoperability group at the Natl. Evolutionary Synthesis Center). In particular, a seminal paper on desiderata for phylogenetic databases [13] provided a classification of phylogenetic investigations and the associated types of queries.

The study in [13] identifies six classes of uses of phylogenetic repositories: *(1)* Casual Uses (occasional search for a phylogeny, e.g., for informational use); *(2)* Visualization Uses (retrieval of one or more phylogeny for graphical display); *(3)* Study Development Uses (contribution of new phylogenies or update of existing ones); *(4)* Super-Tree Uses (for assembly of phylogenies into super-trees); *(5)* Simulation Studies (e.g., to assess performance of models of evolution); *(6)* Comparative Genomics Studies (use of phylogenies to relate different genes/genomes).

The investigation in [13] continued by breaking down the needs of these six application areas in terms of classes of queries, leading to eleven basic types of phylogenetic queries. We revised these classes of queries through our own focus groups and refined them into the following classes of queries:

Q1. Determine all the phylogenies containing a given set of taxa—e.g., locate all trees containing the taxonomic units named `Ilex anomala` and `Ilex glabra`.

Q2. Determine the relationships among a set of taxa in all phylogenies (query not supported).

Q3. Determine the minimum spanning tree/clade for a given set of taxa—e.g., locate the minimum spanning clade in the tree `Tree3099` (TreeBase identifieer) for the taxonomic units `Ilex anomala` and `Ilex glabra`.

Q4. Determine all phylogenies constructed using a given inference method—e.g., locate all the phylogenies constructed using a parsimony method.

Q5. Determine all the phylogenies containing a set number of taxa—e.g., locate all the phylogenies with at most 25 taxa.

Q6. Determine all the phylogenies produced by a given tool or author—e.g., locate all the phylogenies published by W. Piel.

Q7. Determine all phylogenies satisfying a given geometric property—e.g., locate all the phylogenies that have diameter equal to 5.

Q8. Given a phylogeny P, a measure m, and a quantity q, determine all the phylogenies that are at distance q from P according to the measure m (e.g., for the purpose of clustering phylogenies that are "close" to a given tree).

Q9. Given a model of evolution, determine all the phylogenies that have been constructed using such model of evolution—e.g., identify all the phylogenies that have been constructed using Jukes-Cantor model for estimating distance.

Q10. Given a measure, return statistics about the measure in the phylogenies present in the repository—e.g., determine the distribution of tree lengths

Q11. Given a type of data and a set of taxa, determine all the phylogenies on the set of taxa that have been constructed using the specified type of data—e.g., determine all phylogenies built using DNA sequences. sequences.

Two classes of queries are currently not supported—classes **Q2** and **Q8**. The class **Q2**, drawn from the study in [13], is only vaguely specified, and further investigation is in progress to characterize the type of relationships to be considered—the queries in this class are aimed at discovering opportunities for clustering of taxa based on how they are related by the phylogenies in the repository. Queries in class **Q8** are associated to the use of distance measures between phylogenetic trees (e.g., Robinson-Foulds distance, K tree score, Geodesic distance) to determine clusters of trees. For questions of type **Q7**, we currently support the computation of radii and diameters—respectively defined in terms minimum and maximum eccentricity of the phylogeny, where the eccentricity is the maximum distance in the phylogeny among any two nodes.

4.2 Query Implementation

The various classes of queries listed above have been implemented in CDAOStore using a combination of SPARQL and Prolog; some of the queries (e.g., **Q1**) do not require much more than a database search and can be mapped to corresponding SPARQL queries; other queries require more involved reasoning on phylogenies (e.g., computation of nearest common ancestors) and these are implemented by mapping phylogenies into Prolog terms and using Prolog to address the queries.

4.2.1 Preprocessing

In order to use of Prolog in CDAOStore, we developed a pre-processor, which is used to generate a Prolog program that will produce the result of the submitted query. The first step of the pre-processor is to generate SPARQL queries to retrieve from the triple store the phylogenies of interest (i.e., those referred to by the query being executed) and convert the RDF representation of such phylogenies into a collection of ground Prolog facts. The pre-processor combines these ground facts with a set of pre-determined Prolog rules, which describe the computation of the query, and feeds the result to a Prolog engine for execution (in our case, SWI-Prolog). Finally, the pre-processor converts the result back into a RDF format for final output.

The facts used to describe phylogenies include facts of the form:

```
tree(TreeName).
node(TreeName, NodeName).
edge(TreeName, EdgeName, Direction, SourceNode, DestinationNode).
```

For simplicity, the nodes are classified as internal, root, or leaf nodes by adding to the facts the rules:

```
leaf( Tree, Node ) :- node( Tree, Node ), not(edge(Tree, _, _, Node, _)).
root( Tree, Node   :- node( Tree, Node ), not(edge(Tree,_,_, _,Node )).
internal_node( Tree, Node ) :- node( Tree, Node ), edge(Tree,_,Node,_).
```

The set of rules added include those used to determine the ancestors of a node, in the form of a predicate `ancestor_of(Tree, Ancestor, Node)`, as a simple transitive closure of the `edge` predicate.

For queries that involve the entire repository (e.g., **Q10**), the process above is broken down into chunks, where the Prolog system is invoked not

on all the phylogenies at once, but on groups of a given size (established as a system parameter); the results are incrementally aggregated to produce the final result.

4.2.2 Implementation

Queries Q1: This class of queries is used to determine all the phylogenies containing a given set of taxa. This query can be implemented directly in SPARQL:

```
PREFIX study: <http://www.cs.nmsu.edu/~bchisham/study.owl#>
PREFIX contact: <http://www.w3.org/2000/10/swap/pim/contact#>
PREFIX foaf: <http://www.mindswap.org/2003/owl/foaf#>
SELECT ?tree WHERE {
    ?tree has_TU TU1.
    . . .
    ?tree has_TU TUN.  }
```

Note that the various taxa passed as arguments refer to standardized names of taxa within the CDAOStore; work is in progress to adapt the internal nomenclature to satisfy global naming standards being developed by several working groups (e.g., the Darwin Core [22]).

Queries Q3: This class of queries is used to determine the minimum spanning clade for a given set of taxa or the nearest common ancestor of a given collection of taxa. These queries are implemented using the Prolog engine. The nearest common ancestor is an ancestor of all taxa in a given set such that none of its descendants is also an ancestor of all such taxa. The Prolog code is relatively straightforward:

```
common_ancestor_of(Tree, Ancestor, [Node]) :- !,
     ancestor_of(Tree, Ancestor, Node).
common_ancestor_of(Tree, Ancestor, [Node | Nodes]) :-
     ancestor_of(Tree, Ancestor, Node), common_ancestor_of(Tree, Ancestor, Nodes).
distant_common_ancestor_of(Tree, DistantAncestor, Nodes) :-
     common_ancestor_of(Tree, Ancestor, Nodes),
     ancestor_of(Tree,DistantAncestor, Ancestor).
nearest_common_ancestor_of(Tree, Nca, Nodes) :-
     common_ancestor_of(Tree, Nca, Nodes),
     not(distant_common_ancestor_of(Tree, Nca, Nodes)).
```

The minimum spanning clade of a set of taxa is the set of nodes that includes the nearest-common ancestor of all the taxa in the set, and all of its descendants.

```
clade( Tree, Node, Member ):- ancestor_of( Tree, Node, Member ).
clade( Tree, Node, Node):- node( Tree, Node ).
msclade( Tree, Nodes, Clade ):-
   nearest_common_ancestor_of(Tree,NCA,Nodes),
   setof( Member, clade( Tree, NCA, Member ), Clade ).
```

Queries Q4: This class of queries is used to determine all the phylogenies constructed using a given inference method. There are two different "methods" information that can be checked (as from the CDAO specification)—i.e., the type of algorithm used and the specific phylogenetic inference system (i.e., the specific software used). Both cases can be handled in SPARQL:

```
SELECT ?tree WHERE {
  ?study study:has_analysis ?analysis.
  ?analysis study:has_algorithm '$ALGO' .
  ?analysis study:has_output_tree ?tree . }
```

```
SELECT ?tree WHERE {
?study study:has_analysis ?analysis .
?analysis study:has_software '$ALGO' .
?analysis study:has_output_tree ?tree .
```

where *$ALGO* is the variable for the given algorithm/software being checked for.

Queries Q5: This class of queries is used to determine all the phylogenies containing a given number of taxa. This query requires the use of Prolog, to count the number of leaves in the phylogenies:

```
taxa_count( Tree, Count ) :- leaf_count( Tree, Count ).
leaf_count( Tree, Count ) :- setof( LNode, leaf( Tree, LNode ), Nodes ),
                             length( Nodes, Count ).
tree_with_n_taxa( N, Trees ) :- findall(T, (tree(T), taxa_count(T,N)), Trees).
```

Queries Q6: These queries are used to determine all the phylogenies produced by a given tool or author; these can be easily addressed using SPARQL. For example, to search for a specific author:

```
SELECT ?study WHERE {
  ?study study:has_author ?authorid.
  ?authorid foaf:last_name '$LAST_NAME'^^<http://www.w3.org/2001/XMLSchema#string>.
  ?authorid foaf:first_name '$FIRST_NAME'^^<http://www.w3.org/2001/XMLSchema#string>.}
```

where *$LAST_NAME* and *$FIRST_NAME* are the last and first name of the author, respectively. The option to search on just the last name is also available.

Queries Q7: CDAOStore currently supports only the computation of queries requesting phylogenies with a given constraint on either their radii or their diameters. The code for the radius is provided next—the predicate computes the radii of the trees (through a recursive comparison of distances among leaves) and selects those that have a radius equal to the requested value `R`. The code for the diameter is analogous, with the exception that we will maximize the eccentricity instead of minimizing it.

```
%% Query entry point
radius_count( Tree, R, Trees ) :- setof(T, (tree(T), radius(T,R)), Trees).
radius( Tree, R ) :- findall(Leaf, leaf(Tree, Leaf), [L|Leaves]),
   max_distance(Tree, L, Leaves, Curr),
   radii(Tree, Leaves, R, Curr).
radii( _, Leaf, R, R) :- length(Leaf, 1), !.
radii(Tree, [Leaf | Leaves], Radius, Curr) :-
   max_distance(Tree,Leaf,Leaves, Len),
   (Len < Curr *-> radii(Tree,Leaves,Radius,Len);
                        radii(Tree,Leaves,Radius,Curr)).
eccList(_,_,[],[]).
eccList(Tree, Leaf, [LeafNode | Leaves], [E | Rest]) :-
   pathlength(Tree,Leaf,LeafNode,E),
   eccList(Tree,Leaf,Leaves,Rest).
max_distance(Tree, Node, Nodes, Dist) :- eccList(Tree,Node,Nodes,Lens),
   max_list(Lens,Dist).
```

Queries Q9: This query is used to determine all phylogenies that have been constructed using a particular model of evolution; this query is simply mapped to a SPARQL query that filters phylogenies based on the model of evolution property (code omitted due to lack of space). Unfortunately, the TreeBase repository is lacking this type of meta-data, preventing us from experimenting with it.

Queries Q10: This class of queries is used to determine statistical information about the phylogenies present in the repository (e.g., distribution of tree lengths). The code has been developed to support the computation of the mean, median, mode, and standard deviation of size of trees, edge lengths,

radii, and diameters of all phylogenies in the repository. These queries are implemented in Prolog. Here below we sketch its main aspects.

```
stat_measures(Type, Mean, Median, Mode, Dev) :-
   select_data(Type, List), msort(List, Sorted),
   find_mean(Sorted, Mean), find_median(Sorted, Median),
   find_mode(Sorted, Mode), find_stddev(Sorted, Dev).
select_data(size, List) :-
   findall(C, node_count(Tree,C), List).
select_data(edge_length, List) :-
   findall(C, (edge(Tree,Name,_,_,_), edge_length(Name,C)), List).
...
findMean(List, Mean):- sum(List, Sum), length(List, N),
   Mean is Sum / N.
...
```

Queries Q11: This class of queries is used to determine, given a type of data and a set of taxa, all the phylogenies containing all the given taxa and whose construction involved data of the given type (e.g., DNA). This type of queries can be addressed using SPARQL, since CDAO provides properties describing all these features; the overall structure of the query is

```
SELECT ?tree  WHERE {
   ?study has_analysis ?analysis.
   ?analysis has_output_tree ?tree.
   ?analysis has_input_matrix ?matrix.
   ?tree has_TU <TU1>.
   . . .
   ?tree has_TU <TUN>.
   ?matrix has_Character ?character.
   ?character rdf:type cdao:AminoAcidResidueCharacter. }
```

5 Preliminary Evaluation

The CDAOStore has been implemented and it is now available for access and use. The implementation has been realized using a collection of publicly available tools. In particular, the triple store has been implemented on top

of the RDFlib Python library, the Prolog components in SWI-Prolog, and various libraries have been adopted to deal with specialized data formats.

The store has already been populated by importing into CDAOStore the complete content of the TreeBase [1] repository, encoded using CDAO and enhanced with several semantic annotations drawn from several associated repositories. At the time of the development of this paper, CDAOStore contains over $4,000$ phylogenies, and the overall size of the repository is now at more than 90 million RDF triples.

We performed some preliminary experiments to evaluate the performance of the system on some sample queries, drawn from scientific publications. Let us observe that performance was not one of the main driving criteria in initial design and implementation of CDAOStore—we focused more on providing an environment that is flexible and provides querying capabilities that are beyond what supported in existing repositories. Furthermore, CDAOStore is viewed as a service provider that will be used by other clients for various types of applications.

The performance in answering the queries has been measured in terms of time to respond, measured using the `time` and `curl` commands. Since our Web portal uses cgi-bin, we evaluated performance by performing a direct call with a specific URL, instead of going through the actual HTML form, in order to get a more accurate time measurement. The queries have been performed on a server running on a HP Intel Core i7 860 machine, with 8GB of memory and making use of SuSE Linux.

Query	**PhyloWS Time**	**Web Portal Time**
T1	2.44	1.20
T2	1.82	3.18
T3	0.91	4.19
T4	6.12	6.18
T5	6.19	6.26
T6	32.58	35.22
T7	5.42	5.04
T8	15.91	*

Table 1: Execution Times for Sample Queries (time in sec.)

Table 2 summarizes the queries and results, while Table 1 summarizes some of the execution times. The timings are expressed in seconds. The different rows correspond to different queries. The first time column reports

the response time while issuing the query through the PhyloWS API, while the second made use of the web portal. Note that occasionally the web portal provides a faster response time as some of the argument parsing is pre-determined by the specific fields in the portal. We do not report results for queries of type 9 because meta-data about models of evolution are missing from the current repository. Queries 10 and 11 have been only very recently integrated in the system. A particular note for query of type 7—the pre-processing of the query required excessive time, leading to a time-out of the web server; we are working on addressing this issue.

Query	Description	Query Type	Answer
T1	Phylogenies containing taxa `Ilex anomala` and `Ilex glabra`	**Q1**	16 phylogenies
T2	Minimum Spanning Clade for taxa `Ilex anomala` and `Ilex glabra` in tree `Tree3099`	**Q3**	1 clade, 14 nodes
T3	Nearest Common Ancestor of taxa `Ilex anomala` and `Ilex glabra` in tree `Tree3099`	**Q3**	1 node
T4	Phylogenies constructed using Parsimony algorithm	**Q4**	3,636 phylogenies
T5	Phylogenies constructed using PAUP*	**Q4**	4,091 phylogenies
T6	Phylogenies with less than 25 nodes	**Q5**	3,120 phylogenies
T7	Phylogenies co-authored by W.H. Piel	**Q6**	3 phylogenies
T8	Phylogenies with radius equal to 10	**Q7**	1 phylogeny

Table 2: Sample queries

Some of the queries denote a relatively larger execution times (in the order of several tens of seconds), due to the fact that the queries check all possible phylogenies in the triple store, rather than looking only at nodes from a particular phylogeny. Some ways to improve these execution times may include maintaining additional summary information of the various phylogenies (e.g., for **Q5** we could maintain summary of the main statistical information of each phylogeny). Also some restructuring of the relationships between author, study, and their associated trees would help to improve query speed. An asynchronous query mechanism would also be useful for running queries such as radius and diameter, avoiding the web server timeout issue mentioned earlier.

6 Conclusion and Future Work

The CDAOStore is a collaborative effort to implement a repository of results from phylogenetic analysis studies in the field of life sciences, built on a formally specified inter-operation stack, the EvoIO stack, composed of a formal ontology, a standard exchange format, and a web service API. The first deployment of CDAOStore has been completed with success, embedding a sophisticated domain-specific querying API and Web interface, implemented using a combination of SPARQL and Prolog. The novelty of CDAOStore lies in the use of a semantic-based approach to the storage and querying of data, building on established ontologies for the semantic annotation of data and on a query language which is domain-specific. These are features that are absent from related existing repositories (e.g., TreeBase [1], Tree of Life Web project [11], Dryad [2]). This approach enables us to overcome restrictions imposed by the use of specific data formats (facilitating interoperation among phylogenetic analysis applications) and makes it possible to formulate more meaningful domain-specific queries.

We are currently working on extending the set of domain-specific queries supported by CDAOStore, paying particular attention at queries used to discover clustering of taxa and clustering of phylogenies, according to different types of distance measures. We are also exploring ways of enhancing the speed of some queries, through better representations and pre-computation of additional meta-data. An important component of our future work will include the evaluation of the suitability of the current platform (based on RDFlib) to sustain the growing size of the triple store. Alternative platforms are available (e.g., Jena, RAP, AllegroGraph), with probably better performance but also with steeper amount of work required to integrate the various system components.

The CDAOStore platform is open-source and is available as a SourceForge project, at `sourceforge.net/projects/cdaotools`. The portal to CDAOStore is available at `http://www.cs.nmsu.edu/~cdaostore`.

References

[1] Treebase. `http://www.treebase.org`, 2010.

[2] 2011.

[3] H. Ellegren. Comparative genomics and the study of evolution by natural selection. *Molecular Ecology*, 17(21):4586–4596, 2008.

[4] J. Felsenstein. PHYLIP: Phylogeny Inference Package. *Cladistics*, 5:164–166, 1989.

[5] V. Gopalan, W. Qiu, M. Chen, and A. Stoltzfus. Nexplorer: phylogeny-based exploration of sequence family data. *Bioinformatics*, 22(1):120–121, 2006.

[6] A. Hill, S. Pick, and R. Guralnick. PhyloBox, 2010.

[7] J.R. Iglesias, G. Gupta, E. Pontelli, D. Ranjan, and B. Milligan. Interoperability between bioinformatics tools: A logic programming approach. In *PADL*, pages 153–168. Springer Verlag, 2001.

[8] S. Kumar, J. Dudley, M. Nei, and K. Tamura. MEGA: A Biologist-centric Software for Evolutionary Analysis of DNA and Protein Sequences. *Briefings in Bioinformatics*, 9:299–306, 2008.

[9] H. Lapp and R. Vos. Phyloinformatics web services api: Overview. `https://www.nescent.org/wg/evoinfo/index.php?title=PhyloWS`, National Evolutionary Synthesis Center, 2009.

[10] N. MacLeod. The role of phylogeny in quantitative paleobiological data analysis. *Paleobiology*, 27(2):226–240, 2001.

[11] D. Maddison, K. Schulz, and W. Maddison. The Tree of Life Web Project. *Zootaxa*, 1668:19–40, 2007.

[12] D. Maddison, D. Swofford, and W. Maddison. NEXUS: an Extensible File Format for Systematic Information. *Syst. Biol.*, 46(4):590–621, 1997.

[13] L. Nakhleh, D. Miranker, and F. Barbancon. Requirements of phylogenetic databases. In *Third IEEE Symposium on Bioinformatics and Bioengineering*, pages 141–148. IEEE, 2003.

[14] NESCent. Supporting nexus. https://www.r-phylo.org/wg_phyloinformatics/Supporting_NEXUS, National Evolutionary Synthesis Center, 2008.

[15] W.H. Piel. Phyloinformatics and tree networks. In *Computational Biology and Genome Informatics*, pages 239–252. World Scientific Press, 2003.

[16] F. Prosdocimi, B. Chisham, E. Pontelli, J.D. Thompson, and A. Stoltzfus. Initial implementation of a comparative data analysis ontology. *Evolutionary Bioinformatics*, 5:47–66, July 2009.

[17] E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. Technical Report REC-rdf-sparql-query-20080115, W3C, 2008.

[18] G.J.D. Smith et al. Origins and evolutionary genomics of the 2009 swine-origin h1n1 influenza a epidemic. *Nature*, 459:1122–1125, 2009.

[19] A. Stoltzfus, N. Cellinese, K. Cranston, H. Lapp, S. McKay, E. Pontelli, and R. Vos. The evoio interop project. http://www.evoio.org/wiki/Main_Page, National Evolutionary Synthesis Center, 2009.

[20] R. Vos. NeXML: Phylogenetic data in xml. http://www.nexml.org, 2008.

[21] C. Webb, D. Ackerly, M. McPeek, and M. Donoghue. Phylogenies and communtiy ecology. *Annu. Rev. Ecol. Syst.*, 33(1), 2002.

[22] J. Wieczorek, M. Döring, R. De Giovanni, T. Robertson, and D. Vieglais. Darwin core. http://rs.tdwg.org/dwc/index.htm, Darwin Core Task Group / TDWG, 2009.

[23] M. Wu and J. Eisen. A simple, fast, and accurate method of phylogenomic inference. *Genome Biology*, 9(10):R151, 2008.