International Journal of Software Engineering
and Knowledge Engineering
Vol. 30, No. 10 (2020) 1525–1550

DOI: 10.1142/S0218194020500382

World Scientific
www.worldscientific.com

Design and Implementation of Phylotastic, a Service Architecture for Evolutionary Biology

Abu Saleh Md. Tayeen*, Thanh Hai Nguyen†,
Van Duc Nguyen‡ and Enrico Pontelli§

Department of Computer Science
New Mexico State University
Box 30001, MSC CS
Las Cruces, NM 88003, USA
**tayeen@nmsu.edu*
†*thanhnh@nmsu.edu*
‡*vannd@nmsu.edu*
§*epontell@nmsu.edu*

Received 1 May 2019
Revised 14 May 2020
Accepted 26 June 2020

Access and reuse of authoritative phylogenetic knowledge have been a longstanding challenges in the evolutionary biology community — leading to a number of research efforts (e.g. focused on interoperation, standardization of formats, and development of minimum reporting requirements). The *Phylotastic* project was launched to provide an answer to such challenges — as an architectural concept collaboratively designed by evolutionary biologists and computer scientists. This paper describes the first comprehensive implementation of the Phylotastic architecture, based on an open platform for Web services composition. The implementation provides a portal, which composes Web services along a fixed collection of workflows, as well as an interface to allow users to develop novel workflows. The Web services composition is guided by automated planning algorithms and built on a Web services registry and an execution monitoring engine. The platform provides resilience through seamless automated recovery from failed services.

Keywords: Bioinformatics; phylogenies; Web services; services composition.

1. Introduction

Phylogenetic trees are useful in all areas of biology (and beyond, e.g. in linguistic studies), both to provide instruments to organize knowledge in taxonomic forms and for process-based models, which allow scientists to make robust inferences from

§Corresponding author.

comparisons of descriptive characters (e.g. DNA, phenotypes) of evolved entities (e.g. genes, species). Indeed, intense efforts have been invested in the assembling of a *Tree of Life* (ToL), a phylogeny covering 10^7 or more species [19]. The first draft of a grand phylogenetic synthesis — a single synthetic tree with 2.5×10^6 species (tree.opentreeoflife.org) — emerged from the *Open Tree of Life* (*OpenTree*) project.

Though useful, neither this tree nor any other individual phylogenetic tree, will be the sole authority on phylogenetic knowledge. Indeed, several components of the OpenTree are synthetic and not yet grounded in actual biological knowledge. Therefore, when researchers refer to the "ToL" they often do not mean any single tree, but the dispersed set of available trees that represent the current state of ToL knowledge. While experts continue expanding the ToL, addressing gaps and conflicts, an unexpected challenge has appeared: how to *disseminate* and *reuse* such knowledge, i.e. how ToL knowledge can be effectively placed in the hands of researchers, educators, and the broad public. The vision of this project, referred to as *Phylotastic,* is to enable easy online and programmatic access and use of biological phylogenetic knowledge (e.g. species trees).

The premise of disseminating knowledge is that it will be *reused.* Studies conducted show that the level of reuse of phylogenetic trees is limited; indeed, it is more common to see phylogenetic trees being inferred de novo for each specific study instead of being used as-is from previous investigations [34]. Yet, with a greater emphasis placed on the construction of large species trees and the creation of authoritative phylogenies, we are seeing an increasing interest toward reuse of phylogenetic knowledge. In a sample of 40 phylogeny papers, the authors of [34] found six cases in which scientists obtained a desired tree by extraction from a larger species tree. We refer to this important pattern of reuse as *subtree extraction* from *ToL source trees.* The potential impact of facilitating the process of subtree extraction from authoritative phylogenetic trees is huge, and has been discussed in the seminal paper on the Phylotastic project [33].

While simple in principle, this mode of reuse currently presents technical barriers, which justify the need of a dedicated informatics infrastructure. The vast majority of users simply do not have the ability to handle a tree with more than a 1000 species, even if they know how to locate the right tree — a challenge, as only 4% of trees are archived [34]. Tree files generally lack machine-processable metadata on sources and methods, crucial for quality evaluation and for documentation of the scientific investigation being conducted. The largest, most valuable species trees often provide a topology without branch lengths, yet users often need branch lengths in downstream analysis steps; proficient users may create crude branch lengths with specialized software. Even matching a list of species names with a source tree is problematic, given the proliferation of aberrant names, due to spelling errors and lexical variants [22]. Whereas subtree extraction is conceptually simple, real-world uses are surrounded by complications, requiring a combination of expert skills, hands-on attention, and specialized software. To achieve the *Phylotastic* vision of simple and

flexible reuse of phylogenies, this paper proposes an open Web-based system that enables on-the-fly delivery of phylogenetic knowledge.

The novelty of this work is the development of the first ever complete architecture implementing the Phylotastic vision. Phylotastic provided an ideal architecture but the original design, presented in [33] failed to recognize the technological challenges to be addressed to establish a Phylotastic architecture that is neutral with respect to content and that is usable and useful to final users. The objective of the architecture is to reliably deliver data that traces to expert sources. The quality of outputs will depend on the state of inputs, relying on sources like name services (e.g. GNR, iPlant) and Web service accessible species tree repositories, such as TreeBASE [23], an archive of about 10,000 gene and species trees, the Phylomatic service [42], supplying the APG tree and derivatives, and OpenTree [12], providing ~ 4000 source trees, plus synthetic tree of 2.5×10^6 species. The proposed developed architecture is both fronted by a client with a graphical interface as well as a programmable Web services interface. Novelty of the architecture includes the use of advanced automated planning techniques to provide Web service composition and to automatically repair an execution workflow in case of failure and the use of natural language generation (NLG) techniques to automatically generate textual explanation of workflows and their execution.

2. Background

2.1. *Evolutionary biology*

Phylogenies are abstractions that have been used extensively in the field of evolutionary biology to describe the relationships among entities (e.g. biological entities like proteins or genomes) derived from a process of *evolution.* This paper refers to the entities studied in a phylogeny as *taxonomic units* (TUs) or, simply, as *taxa.*

The field of *Phylogenetics* developed from the domain of biology as a powerful instrument to investigate similarities among entities that arose as a result of an evolutionary process. Evolutionary theory provides a powerful framework for comparative biology, by converting similarities and differences into events reflecting evolutionary processes. As such, evolutionary-based methods provide more reliable answers than the traditional similarity-based methods, since they employ a theory of evolution to describe changes, instead of relying on simple pattern matching. Indeed, evolutionary analyses have become the norm in a variety of areas of biological analysis (e.g. [8, 11, 16, 35, 38–40, 43]).

Phylogenetic analysis has found applications in domains that are outside of biology. For example, a rich literature has explored the evolution of languages (e.g. [6, 10, 30, 36]). The definitions and techniques employed are analogous — of course the notion of "observable property" will be different. Starting from genes one notice differences using string matching algorithms. But differences (to be analyzed and

explained) can be more macroscopic, such as the presence/absence of a tail in an animal or the way one says "father" in a language.

2.2. *Evolutionary informatics*

The literature on *informatics support* for evolutionary biology is extensive. In broad strokes, the existing proposals can be organized as follows: (a) data representation, storage, and management; and (b) operations and applications development (automation, run control, composition, reuse and re-purposing).

2.2.1. *Data*

Representation of DNA or protein sequences, phylogenetic, and other classes of biologically relevant data has traditionally relied on a large variety of data representation formats, each typically concentrated about a particular aspect, e.g. nucleotide sequences (e.g. GenBank, GFF), protein structures (e.g. PDB/mmCIF), species-specific data formats (e.g. FlyBase), and multiple alignments (e.g. Phylip, MEGA). Recently, we witnessed a move toward XML-based formats to create at least an underlying syntax similarity (e.g. XEMBL, SWISS-PROT, PDBML). Data representation formats are typically coupled with appropriate data storage formats, ranging from flat file storage schemes, to relational models, to more recent object-oriented and XML-based models.

There have been attempts to develop formats that address the needs of more than one class of applications, providing support to represent different classes of data within the same paradigm, as is the case of NEXUS [18] and NeXML [41] in the context of evolutionary biology. Nevertheless, the issue of interoperation and interchange of biomedically relevant data is still open, and further complicated by the poor formalization of many of these data formats (as exemplified by the variety of flavors of NEXUS adopted by different applications).

Data formats mostly focus on encoding the "appearance" of data (i.e. syntax), while the true challenge in modern phylogenetic approaches requires access to the semantics of the data and the ability to conduct different types of "reasoning" and transformation. For example, these are needed to enable data integration, to facilitate reuse of domain knowledge, its exchange, and interoperation between separate stages of evolutionary analysis. This leads to the need for *ontologies*. Some efforts have attempted to cross the barriers of narrowly focused ontologies to provide controlled vocabularies that can span the needs of wider branches of bioinformatics applications. In the immediate context of evolutionary biology, a consortium of researchers developed the *Comparative Data Analysis Ontology* (CDAO) [4]. CDAO provides a formal ontology for describing phylogenies, their associated characters, and character state matrices. 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: (1) *Operational*

Taxonomic Units (OTUs); (2) *characters and character states*; (3) *phylogenetic trees*; and (4) *transitions.* A phylogenetic analysis starts with the identification of a collection of 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 called *character states.* In phylogenetic analysis, it is common to collect the characters and associated states in a matrix, the *character state matrix,* where the rows correspond to the OTUs and the columns correspond to the characters. 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. Different types of representations of evolutionary knowledge are available, differentiated based on structure of the representation (e.g. resolved trees, unresolved trees, rooted trees), the nature of the encoded knowledge (e.g. phylogenies versus taxonomies) and the methods used to derive them.

2.2.2. *Software*

The traditional view of software support for bioinformatics research relies on the development of independent, task-specific services and applications (e.g. Clustalw, RAP), using different input and output formats (often vaguely specified), and frequently not designed to interoperate. While data formats employed by many such systems are not standardized, the types of data that are generated are often very similar, e.g. most analysis methods applied to sequence alignments (or other character data) generate column-wise attributes of character data; node-wise and branch-wise attributes of trees; or node-and-character-wise attributes. Many of them also estimate parameters for evolutionary transition models that can be represented in a standard form (when there are N character states, an $N \times N$ matrix suffices to represent rates of transition). While most analysis programs limit the user to a few pre-specified types of analysis, there are now generalized systems for evaluating phylogenetic likelihood models that have a clean interface and that can be adapted to many different uses, such as HyPhy and PAL. This suggests that generalizations of data formats (e.g. through ontologies) and of transformations are feasible.

A step forward is represented by the development of multi-component packages, such as Phylip [9], consisting of many task-specific applications that can work together, and integrated systems such as MEGA [14]. Note, however, that these approaches do not represent arbitrary open systems (where new components can be introduced with ease) and none of these approaches has been specifically designed with the issue of programmable interoperability in mind.

To aid in the development of custom software systems for large-scale evolutionary analyses, a number of programming libraries and toolboxes have been proposed. For

instance, BioPerl [2] provides input/output of alignment objects in Phylip format and tree objects in Newick format; `Bio::Tools::Phylo::PAML` provides a PAML [44] output parser. Extensions of traditional programming languages with modules and libraries for evolutionary analyses can be found in BioJava [26] and R [13]. These frameworks address the issue of programmable composition of tasks, but they are still relatively closed system; in particular, they make quite cumbersome the inclusion of new transformations (implemented by existing applications) and they force scientists to explicitly handle low-level implementation details.

3. The Phylotastic Infrastructure

3.1. *Motivations*

The primary motivation underlying the Phylotastic principle is to deliver expert ToL knowledge in a computable, convenient, and credible way. The system must deliver computable information, including trees encoded using standard formats, and metadata encoded using available standards (e.g. for citations), based on formal ontologies or controlled vocabularies. The user interaction must be convenient, in the sense of the following:

(1) Enabling interfaces that align with user expectations;
(2) Returning results in seconds or minutes (not hours, days or weeks, as for de novo tree inference); and
(3) Returning the form of result that integrates into downstream steps.

The system must provide a credible alternative to de novo inference for quantitatively important use cases. In phylogenetics, where the external standard of truth for an inference — actual evolutionary history — is inaccessible, trees are judged by how they are produced, or who produces them, which means that the system must generate a description of sources and methods, sufficient to satisfy users who may wish to include such a description in the methods section of a scientific paper. The second design criterion is to foster a sustainable and adaptable community infrastructure. The main implication of this criterion, in the context of an ever-changing landscape of resource-providers and funding [32], is that the system must be distributed, flexible and open. Ultimately, Phylotastic envisions a community of practice in which different groups of experts can add resources to the system independently, via modular components that interact through common standards.

In our initial design, based on the general principles explored in [33], a *controller* satisfies the user's query by composing a workflow from available *Web services.* A client is any program that uses one or more component services, even indirectly. A controller is a client that mediates workflow operations (e.g. linking the output of X to the input of Y). It may have a user interface or be a library module included in client applications. Our main controller will access a registry of services and invoke

planning algorithms for workflow composition, but other controllers may be simpler. In a typical flow of operations (see Fig. 1), the controller:

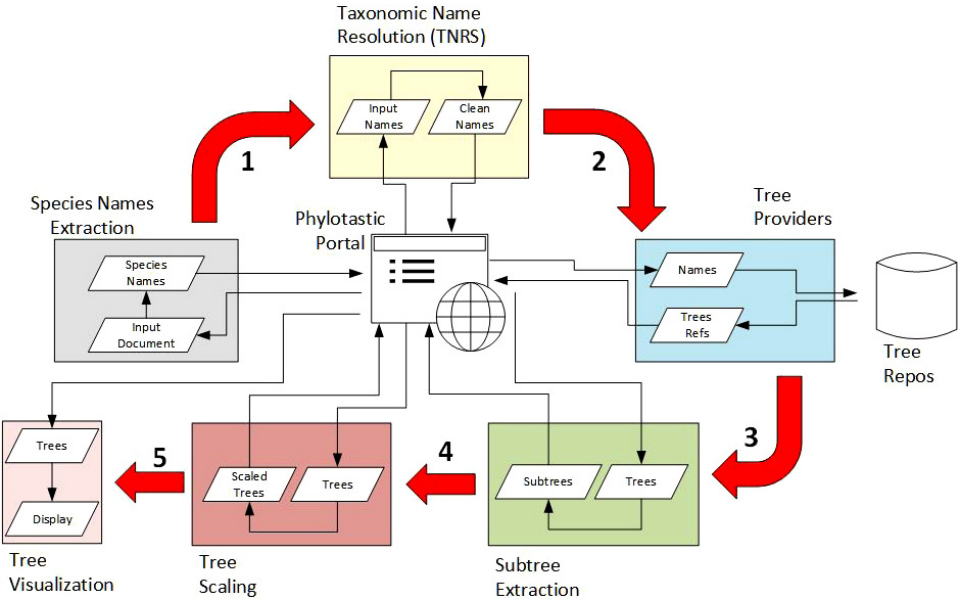

Fig. 1. Typical flow of operations.

(1) Receives a query from user (through a client), issues ticket;
(2) Invokes a Taxonomic Name Resolution Service (TNRS) to clean up names;
(3) Contacts repositories of phylogenies to discover trees that satisfy the query with optimal coverage;
(4) Invokes an extractor to get the relevant subtrees;
(5) Invokes a scaling service to add branch lengths or dates;
(6) Provides the resulting subtrees to the user, with metadata including a report on provenance.

Separate from the development of a production service, a critical contribution of this project is the development of an abstract architecture, composed of standards (e.g. Application Programming Interface (APIs), ontologies), methodologies, and libraries to enable the creation of new services and their integration. This offers several advantages. Phylotastic provides a complete set of production services for the canonical workflow, but the system is open. Anyone with a Web server can implement an API, register their service, and begin processing any requests they receive. Metadata aids users in the selection of services (e.g. in terms of Quality of Services (QoS) or "authority" of data sources).

There is not just one way to use ToL knowledge. The canonical workflow (Fig. 1) takes advantage of four key steps. However, if the problem is merely to

discover all available trees with species A, B, and C, the workflow might have one step (tree discovery) or two (name reconciliation then tree discovery); if the problem is merely to validate names, then only the TNRS is needed, etc. Similarly, the four-component workflow can be expanded with additional operations (e.g. to enrich trees with additional metadata) or embedded in larger analysis workflows. Furthermore, given many workflows, there may be many controllers. For the same workflow, there may be different clients that cater to different cases, e.g. our client for Web resources will be distinct from that for analyzing high-value comparative data sets.

Finally, for purposes of testing, the system presented here implements the minimal redundancy (at least two of each component). Community adoption will result in multiple components of each type, e.g. repositories of phylogenies like TreeBASE [23] and ToLWeb [17] may offer their content via the TreeStore API.

3.2. *Overall architecture*

The Phylotastic project proposes a flexible system for on-the-fly delivery of custom trees, that would support many kinds of phylogenies reuse, and be open for both users and data providers. The overall structure of the Phylotastic architecture is illustrated in Fig. 2. This architecture is an open architecture, composed of a

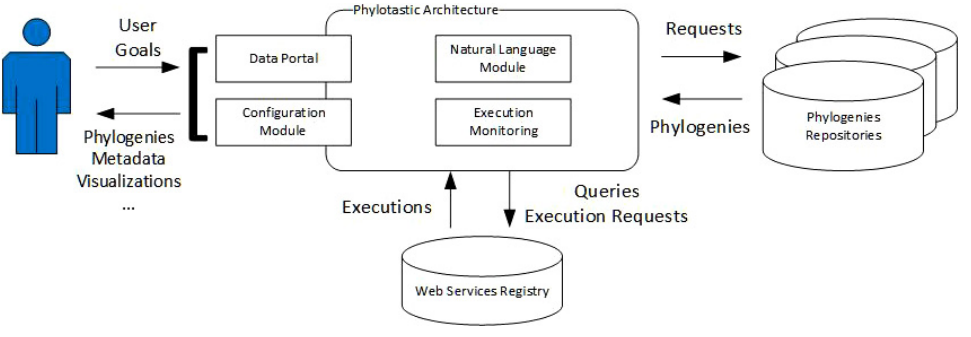

Fig. 2. Overall Phylotastic architecture.

collection of *Web services* relevant to access and reuse of phylogenetic knowledge. Web services are supported by external providers, and novel Web services can be added. Web services available for use within Phylotastic are registered in a Web Services Registry. In addition, Phylotastic consists of the following:

(1) Web Services Execution Monitoring;
(2) a Workflow Configuration Module;
(3) a Data Portal.

The flow of execution of the architecture starts with the Workflow Configuration Module — a graphical user interface that allows the user to provide information about the desired requirements of the phylogenetic trees extraction process. The configuration module is capable of automatically assembling the Web services necessary to implement the requirements set by the user, leading to an executable workflow. The result of such configuration can be visualized and refined; an NLG module can be used to produce English descriptions of the workflow and of the workflow execution. Finally, the executed workflow will be enacted and monitored by the Execution Monitoring module. Alternatively, to the configuration module, the Phylotastic Data Portal provides a number of custom-built workflows, implementing the most commonly used user queries; the portal automatically assembles such fixed workflows using available Web services, provides redundancy, easy to use access, and visualization.

3.3. *Phylotastic data portal*

The Phylotastic data portal is developed to bridge the gap between the technical difficulties of using Web services and the application of these Web services to access and reuse phylogenetic knowledge. The portal supports a collection of predetermined execution workflows, composed of the following steps (Fig. 3):

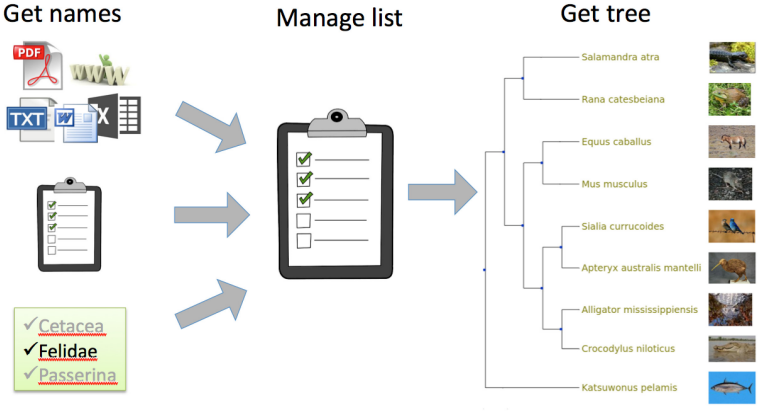

Fig. 3. Overall workflow portal.

(1) The input is aimed at identifying a collection of species names; these names can be either provided directly as a list of names, or they can be extracted from a given document (e.g. a PDF document or a Web page). The portal allows the user to store and retrieve lists of names. In addition, the portal allows the user to request a sample of species belonging to a given group (e.g. a genus, a family) of a

given size and meeting additional user-selected criteria (e.g. belonging to the same geographic region, selected among the most popular species);
(2) The given species names are cleared through a name resolution service to correct misspellings and map to canonical scientific names;
(3) The final list of species names is used to access one of the several repositories of phylogenies and extract a phylogeny that covers the selected species;
(4) If requested, the resulting phylogenetic tree can be scaled (using one of several alternative scaling methods);
(5) Finally, the resulting phylogenetic tree can be visualized; the visualization can be manipulated, e.g. by including common names, requesting images to be added to the tips, drawing boxes around subtrees, etc. The final image can be exported as well as it is possible to export the tree with its supporting metadata.

While the set of possible steps is fixed, the portal executes them via on-the-fly assembling of available Web services; redundancy is present to provide transparent recovery from possible failure of services during the execution.

The Phylotastic data portal is built using a combination of technologies, aimed at ensuring efficiency and reliability. The portal is written in Ruby on Rails (RoR), a model-view-controller (MVC) framework. RoR itself is a Web development framework and it is written in the Ruby programming language. Ruby was influenced by Perl, Smalltalk, Eiffel, Ada, and Lisp.[a] The design of Ruby is like a simple Lisp language at its core, with an object-oriented structure inspired by Smalltalk, blocks inspired by higher-order functions, and practical utilities like that of Perl.[b] Ruby follows the *principle of least astonishment* (POLA), meaning that the language should behave in such a way as to minimize confusion for experienced users. The philosophy behind Ruby's design is for programmer productivity, leading to code which is self-documented, intuitive and expressive. Finally, RoR is an open-source framework, which provides access to a large ecosystem of pre-built code packages (Gem) written in Ruby, covering applications from databases to Web servers.

The portal depends on the following packages (Fig. 4):

- PostgreSQL for database management, to store names generated by users;
- Paperclip for managing file attachments;
- TwitterBootstrap, JQuery, and FontAwesome for front-end development;
- Devise for authentication management — each user of the portal can create an account to maintain lists and trees generated;
- Wicked PDF for PDF generation;
- Capybara, Poltergeist, and Minitest for automated testing;
- Phusion Passenger, Docker, and Kubernetes, for deployment;
- Sidekiq for background job processing.

[a] http://www.ruby-lang.org/en/about/.
[b] http://blade.nagaokaut.ac.jp/cgi-bin/scat.rb/ruby/ruby-talk/179642.

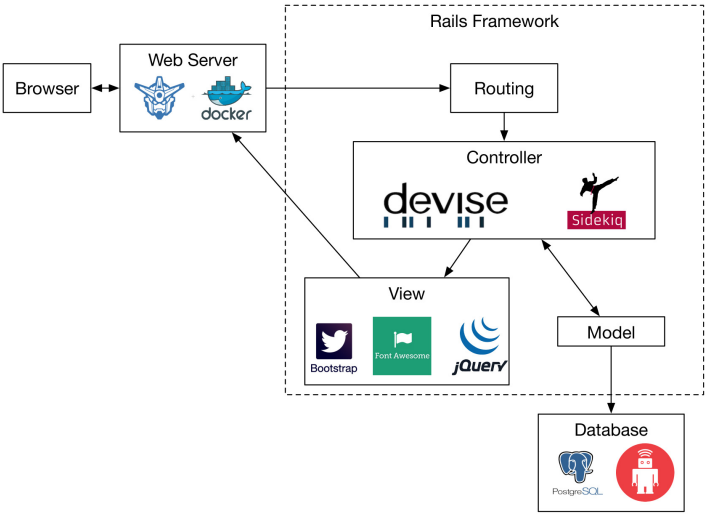

Fig. 4. Some technologies in the portal.

The test suite covers model tests, controller tests, and interactive tests (simulated in Poltergeist, which mimics user interactions).

3.4. *Services and service registry*

3.4.1. *Service type*

There are two types of Web services in the Phylotastic project: synchronous and asynchronous. Synchronous Web services are services with which clients invoke a request and wait for a response. Synchronous services are suitable for applications that require quick responses from the server. Most of the Web services in the Phylotastic project fall into this category. In asynchronous services, clients initiate a request and then resume its processing without waiting for a response from the server. The client can retrieve the response at some later point when the server has finished handling the request. The client can also check the status of the request while it is still being processed. Asynchronous services are well suited for tasks that have high computational overhead and require long time to finish.

Figure 5 shows the workflow of an asynchronous service. Clients make requests and retrieve responses through the service API. Upon receiving a new request from the client, the service API creates a new task and passes it to the broker. The broker places the task into a queue and maintains the task queue. The broker is responsible for taking the tasks from the task queue and distribute them to worker processes or threads. Workers execute the tasks and put the results in a back-end storage. When the client makes a request to check the status of a running task, the service API receives the current progress from the back-end storage and reports back to the

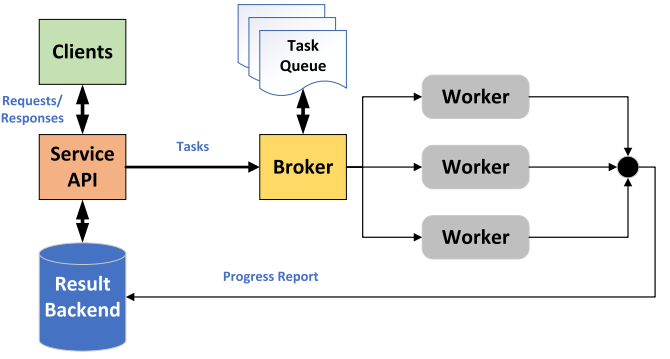

Fig. 5. Asynchronous service workflow.

client. In the Phylotastic project, we adopt Celery [3] to support the distributed task queue and RabbitMQ [25] to implement the broker.

3.4.2. *Web services design*

The Web services for the Phylotastic project have been designed as RESTful Web services, i.e. services that are built using the Representational State Transfer (REST) architectural style: data and functionality are considered as resources, such resources are exchanged between clients and servers using standardized communication protocols, such as HTTP. The key elements of a RESTful implementation are as follows:

(1) *Resource representation*: A REST resource can be any Phylogenetic tree data. This resource can be represented by a plain text file, an XML document or an image file;
(2) *Uniform Resource Identifier* (*URI*): A URI is a sequence of characters to identify a Web resource. A client can use the URI to determine the data access method and parameters to locate and retrieve the specific resource;
(3) *Resource access methods*: Resources are manipulated using a set of well-defined methods — such as creating or deleting a resource. For example, in the case of the HTTP protocol, four different methods can be used to access resources: GET, POST, PUT, and DELETE.

3.4.3. *Services implementation*

Currently, there are 40 Web services registered in the Phylotastic project. Some of these services are wrappers of existing external services, while others have been developed specifically for this project. Most of the service implementations and wrappers have been coded using the Python 2.7 language and they make use of the CherryPy [37] application server, in conjunction with the Nginx [5] Web server. All of these services can be divided into the following broad functional categories.

Taxonomic Name Resolution. These services are used to resolve input strings to standard taxonomic names. Based on the resolution procedure, these services can be of two types. The first type of service resolves misspelled or incorrect scientific names to correct taxonomic names by performing exact or fuzzy matching of strings against known taxonomic sources, such as the *Catalogue of Life*[c] or the *National Center for Biotechnology Information* (NCBI).[d] For example, *GNR_TNRS_wrapper* accepts scientific names as input and returns resolved names with identifiers using the Global Names Resolver.[e] The second type of service resolves common names (vernacular names) of organisms into the corresponding scientific names based on a standard taxonomy. For example, *NCBI_common_name* takes a list of common names and suggests their scientific names depending on the NCBI taxonomy.[f]

Scientific Names Extraction. These services are used to extract scientific names of organisms embedded in text documents, Web pages, PDF documents, Microsoft Office documents, and even images. For instance, the *GNRD_wrapper_URL* service of Phylotastic accepts the URL of a Web resource (which may be an HTML page or an electronic file) and identifies the scientific names using the *Global Names Recognition & Discovery* (GNRD) tool.[g]

Sampling of a Taxon. The services in this category accept a higher taxon name as input and, based on some criteria, return a sample of species belonging to that taxon. For example, the *Taxon_genome_species* service takes a taxon name as input and provides in output a subset of species within that taxon for which there is a genome sequence available in NCBI.

Retrieval of Taxon Information and Image Links. The services of this category retrieve information and image links of input species or higher taxon names from different sources. For example, the *EOL_Habitat_Conservation* service takes a list of species as input and returns the habitat and conservation status, using the *Encyclopedia of Life* (EOL)[h] traits bank. The *Image_url_species* service receives a list of species names as input and obtains image URLs (along with corresponding license information), using the service API provided by EOL.

List Management. These services are used to publish, access, update, or remove lists of species owned by a Phylotastic Web application/services user. For example, the *Remove_list* service allows any valid user (e.g. users with a Gmail address) to remove any public or private list of species from the Phylotastic list server.

[c] http://www.catalogueoflife.org/.
[d] https://www.ncbi.nlm.nih.gov/.
[e] http://resolver.globalnames.org/.
[f] https://www.ncbi.nlm.nih.gov/taxonomy.
[g] http://gnrd.globalnames.org/.
[h] http://eol.org/api/.

Phylogenetic Tree Retrieval. The services in this category accept a list of scientific names as input and return phylogenetic trees as output — the trees are expected to have the given species (or at least a significant subset of them) as tips. For example, the *OToL_wrapper_Tree* service takes a list of input taxa, uses the *Open ToL* (OToL) *match_names* method API to get the Open Tree Taxonomy (OTT) identifiers of the input list, and then uses the *induced_subtree* method API of OToL to retrieve the relevant phylogenetic tree (in Newick format).

Phylogenetic Tree Scaling. These services can be used for fitting a phylogenetic tree to a geologic time scale. For example, the *Datelife_scale_tree* service takes a phylogenetic tree (in Newick format) in input and produces a tree with assigned branch lengths, determined using the Datelife R package.[i]

Phylogenetic Tree Comparison. These services are used to compare two phylogenetic trees. For instance, the *Compare_trees* service accepts two phylogenetic trees (in Newick format) and returns true if the trees are equivalent. This service is implemented using the *DendroPy*[j] Python library and uses the unweighted Robinson–Foulds distance to perform the symmetric comparison between the trees.

3.4.4. *Service registry*

A service registry is required to discover all available Web services. A service registry serves as a centralized repository where clients can find information about the functionality of the services and how to invoke them. In order to publish, sponsors of Web services need to properly describe the services using XML-based documents, e.g. coded using Web Service Description Language (WSDL) [1], and store them persistently through the service registry.

The Web services registry for the Phylotastic project has been implemented as a Web application. It has mainly two layers: the application layer and the data layer. In the application layer, the Drupal[k] technology Content Management System (CMS) is used to manage the content objects, which are Web services, and a Nginx Web server to serve the service registry to the clients. In the data layer, a MySQL Database Management System (DBMS) is used to store the WSDL file locations, the metadata of the Web services and the description of the services. The Phylotastic service registry supports the following basic features:

(1) Accounts management for both users and Phylotastic administrators;
(2) The upload of WSDL files and addition of descriptions of Web services;
(3) The edit of descriptions of Web services;
(4) The removal of WSDL files and relevant descriptions.

[i] http://datelife.org/.
[j] http://dendropy.org/.
[k] https://www.drupal.org/.

3.5. *Execution monitoring*

The outcome of the Web services composition framework and the operations underlying the portal are complete workflows, composed of a sequence of Web services, whose execution is expected to lead to the result requested by the users. Each Web service in the workflow is described by its WSDL profile. In order to execute the workflow, the Phylotastic architecture provides an *Execution Monitoring* module. Such module makes use of the information in the WSDL profiles (extracted from the registry) and the structure of the workflow (e.g. connections between inputs and outputs of the services) to execute the services [20]. The goals of this module are to enact the workflow, producing an execution, and also to monitor the execution to identify possible failures and take appropriate recovery actions. In the current implementation, the recovery process is based on repeating the configuration phase with an added constraint that excludes the failed service. The *Execution Monitoring* module has multiple components. The first component is a WSDL parser; its goal is to parse the WSDL information into data structure models, such as Web services object model, operations object model, parameter components, and elements.

The second component is the *Execution Program* that can execute a concrete Web service operation based on its WSDL profile. This software consumes several parameters: the WSDL URI, the *name of the operation*, and the *list of input components data*. The execution module performs the following steps:

- Given the URI pointing to the service WSDL profile, the execution modules calls the WSDL parser to interpret the profile.
- The detailed profile of the selected service operation is extracted from the WSDL data. This profile includes service endpoints, input and output parameters, content encoding, protocol, etc.
- The execution module issues HTTP/SOAP requests based on the service endpoints and the operation information derived during the previous step. The body of the HTTP/SOAP request includes the inputs to the service (the third parameter mentioned earlier, *list of input components data*), arranged according to the description of the input parameters obtained in the previous step.
- After receiving the response from the service host, the execution module parses the response based on the structure of output parameters of this operation, and analyzes the response to determine how to continue.

The third component is a *Combination Program* that automatically performs repeated calls to the *Execution Program*, in order to execute the entire workflow. The calls follow the structure of the workflow, properly matching inputs and outputs of the different services executed.

The last component is a *Recovery Program* that will be activated when a failure or error in the processing of a single *Execution Program* occurs; this is observed from the status of the Web service (e.g. unavailable, failed, timeout exception, network

problem, forbidden access, etc.). As mentioned earlier, in the current implementation, the *Recovery Program* performs the following steps:

- Detects the failed/unavailable Web service;
- Activates the configuration phase, adding a constraint into *Planning Engine* to exclude the *failed service* from the workflow; this phase results in a new workflow without *failed service*.
- Calls the *Execution Monitoring* with the new workflow.

3.6. *Natural language generation*

From the user perspective, it is handy to extract phylogenetic trees using the features of the Phylotastic portal, but the way the portal brings the results back to the users is potentially confusing. In order to address the problem of clarifying the methods used to determine and extract the phylogenetic tree, Phylotastic provides a NLG module; such module is responsible for the automated development of a readable description of the workflow and its execution. The NLG module is integrated with a visual display which describes the workflow being executed as a diagram; the diagram displays the component of the workflow (e.g. inputs, outputs, services). The diagram is interactive, allowing the user to either abstract the components of the workflow (e.g. describe the workflow in terms of general classes of operations being performed) or to drill down to the details of each specific service being executed. The process of abstracting/concretizing the components of the workflow is based on the Phylotastic ontology describing the Web services. The diagram is synchronized with an automatically generated English description of the content of the diagram, created by the NLG module.

A part of the workflow is illustrated in Fig. 6 (left). The boxes represent the input of a service, the service itself which has green background, and the output of that

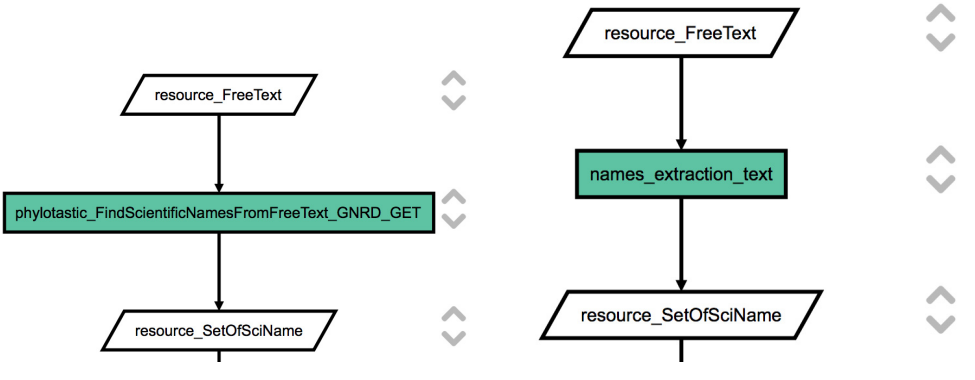

Fig. 6. A part of generated workflow before (left) and after (right) navigating.

service. We can infer that the service *phylotastic_FindScientificNamesFromFreeText_GNRD_GET* requires only one input, which is a text document, and it will return a possible set of scientific names as output. On the right-hand side is the arrow for navigating the level of abstraction of the workflow description. Because *phylotastic_FindScientificNamesFromFreeText_GNRD_GET* is a service which belongs to the class of services *names_extraction_text* in the ontology, when the user clicks on the up arrow on the right-hand side of service box, the new diagram will be generated as in Fig. 6 (right). The generated text is updated corresponding to the change in diagram. Figure 7 provides the explanation of a workflow along with the image of the generated phylogenetic tree.

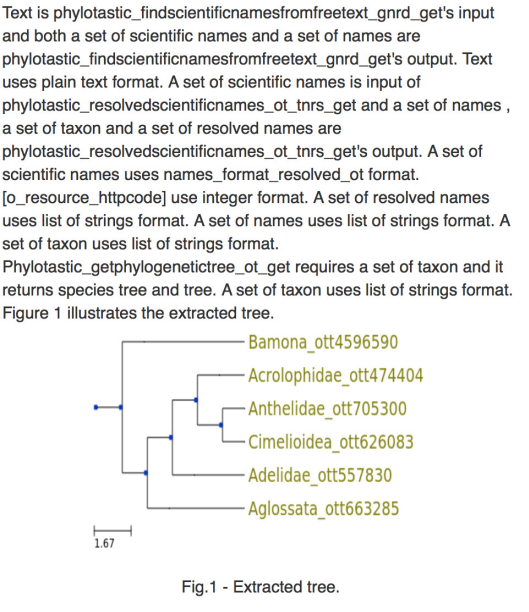

Fig. 7. Description generated for a workflow.

The NLG module is designed as a standalone service which is powered by the *Grammatical Framework* (GF) [27] for language encoding. In this version, the NLG module is developed manually following three major processing phases: (1) document planning (content determination); (2) micro-planning; and (3) surface realization as described in [29]. The result after processing these phases is a set of grammar rules and needed vocabulary which, combined with information from the Phylotastic ontology, form the desired language for encoding in the GF. The language is encoded using the *portable grammar format* (pgf) and can be loaded in any server that uses the GF Runtime API.

4. From Predefined Workflows to Automated Configuration

While the Phylotastic portal allows the execution of predefined workflows, which should satisfy common users interested in the extraction of phylogenetic trees (e.g. in an educational context), we envision situations where the user may want to go beyond these fixed workflows and develop her/his own analysis pipelines. The workflow configuration component provides an infrastructure to achieve such objective. The configuration component receives as input a partially specified workflow designed using a graphical user interface; the specification may include an identification of inputs (e.g. the URL of a document), description of a desired result (e.g. a scaled species tree), and even classes of operations that should be part of the workflow (e.g. include a call to a name resolution service). The problem of mapping the set of requirements specified by the user into a satisfactory workflow is an instance of the *Web services composition problem* [7, 15, 28]. The results of the Web service composition process comprise a workflow that can be executed by the *Execution Monitoring module*. A workflow is a directed acyclic graph, whose nodes represent the services at different levels of granularity and whose links represent the data flow between the services as well as the constraints on the execution of the workflow.

The visual interface of the configuration module allows the user to express the requirements by designing a graph; the interface is Web-based, developed using HTML5 and JavaScript. The specific requirements that can be described include the following:

- *Input*: The input of the workflow can be anything that can be consumed by the registered Web services. Often, the input of is a collection of classes of resources and data formats described using the Phylotastic ontology. Since the ontology organizes the classes of resources as a taxonomy, the user interface allows the user to navigate the taxonomy of resources to select the desired class of inputs.
- *Output*: The output of the workflow can be anything produced by the registered Web services. The user can specify the outputs via *tree view* and *dialogs* Web components that allow the navigation of the Phylotastic ontology.
- *Inclusion services*: The module allows the users to request particular Web services (or classes of Web services) to be used as nodes in workflow — i.e. the final workflow needs to include such operations. For example, in order to compute a `reconciliation tree` (desired output) from a `species tree` and a `gene tree` (desired inputs), the user may request the inclusion of a `scaling_tree` service to ensure that trees are scaled before the reconciliation process.
- *Avoidance services*: The module allows the users to request exclusion of Web services from the workflow. For example, a user may request to avoid using `get_phylogeny_tree_TreeBase`, i.e. a tree retrieval service based on the TreeBase repository [23], as this may provide inadequate coverage of certain species.
- *Insertion services*: This is a special case of an inclusion request, which allows the user to specify the relative ordering of such services in the workflow.

After the users requirements are configured, these data will be sent to the *Planning Engine*. This module maps the collection of requirements into a *planning problem* [24, 31], whose resulting plans correspond to the possible workflows satisfying the given requirements. In the case of Phylotastic, the planner has been encoded using *Answer Set Programming*; a thorough description of the planner can be found in [20]. In addition to the direct generation of a workflow from a set of requirements, the Phylotastic configuration module allows the users to refine and modify a workflow, e.g. by requesting addition and/or removal of operations. The modifications are provided using the same interface used to describe the initial requirements, in the form of editing of a graph representation of the computed workflow. Modifications of a workflow will require reactivation of the planner to determine a new workflow that is consistent with the required modifications. In these situations, the planner will determine a workflow that accommodates the requested changes while preserving *as much as possible the original workflow* [21]. In addition, the module provides users with the following capabilities:

- *Configuration*: A workflow can be saved, updated and reused.
- *Feasibility checking*: A workflow is an "underspecified" plan that, ideally, can be expanded to an executable plan. To provide this capability, the *Workflow Configuration Module* sends the workflow to the *Planning Engine* and requests a possible completion. When no completion exists, the *Workflow Configuration Module* notifies the users that the workflow is not executable.
- *Workflow execution*: The module activates the execution of the workflow. After the desired workflow is generated successfully, it will be sent to the *Execution Monitoring* corresponding with data of input to execute and get the final result.

5. Evaluation

5.1. *Sample execution of the Phylotastic portal*

The portal supports various options to extract species names from a biological taxonomy. For example, to query five species of the family Canidae, which includes domestic dogs, wolves, coyotes, etc., the user needs to fill in a form as in Fig. 8:

Figure 9 displays the result from the portal which is a list of species. In this particular example, the species have been selected based on a popularity service. The user can interact with the list to select/deselect any species that the user wants or does not want to include in rest of the analysis. After pushing the *Get tree* button, the user is redirected to the tree visualizer interface, illustrated in Fig. 10. Here, the interface allows the user to modify the appearance of the tree and/or attach more information (such as images for species or show the common names) to the tree. The user can also freely export the tree in multiple formats, read more about the species or query the supporting studies about the extracted tree.

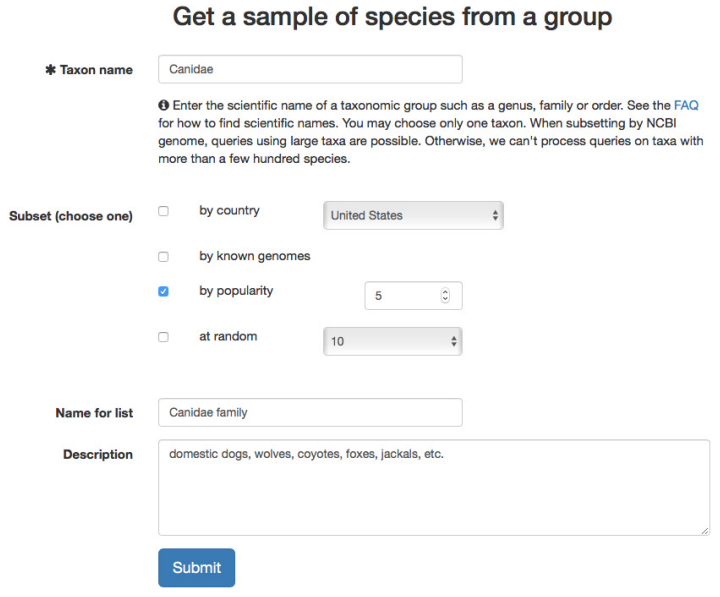

Fig. 8. The form to query five species of biological family Canidae.

5.2. *Sample execution of workflow configuration*

The paper demonstrates the *Workflow Configuration Module* using one of *Phylotastic* use cases: generate a chronogram from Plain Text. In this use case, a workflow for generating a chronogram (i.e. a phylogenetic tree with branch lengths) from a plain text document is created. The first step of configuration is to set up the initial

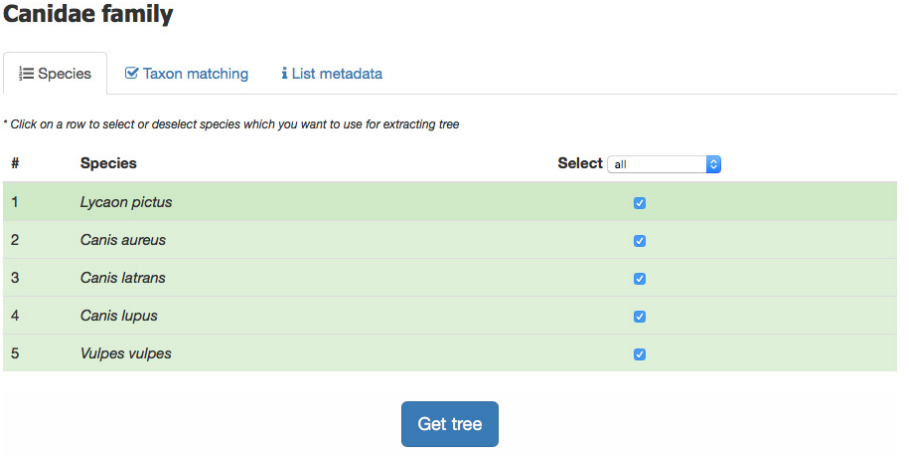

Fig. 9. Five species of biological family Canidae selected by the portal.

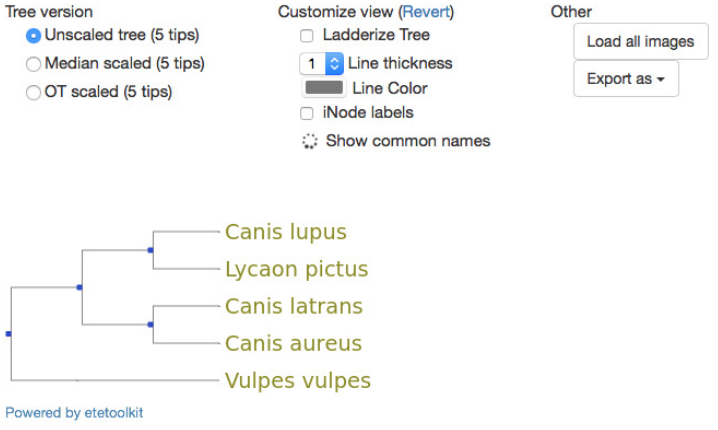

Fig. 10. The tree visualizer for the tree extracted from the list in Fig. 9.

input and the desired output of the workflow. In this step, all input/output resource components attached with data formats are identified from the Phylotastic ontology (Fig. 11). In this use case, the input includes two components: a *free_text* and a *method* associated with *plain_text* and *string* data formats, respectively; the desired output is a *chronogram* with *Newick* format. Figure 12 displays the successful configuration of initial input and desired output.

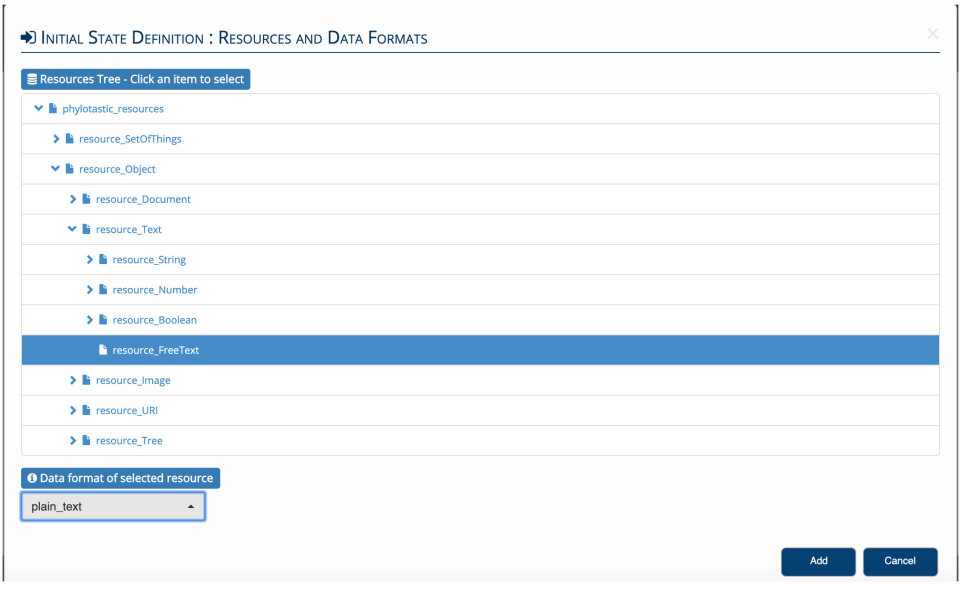

Fig. 11. Configure the resources and data formats of input/output components.

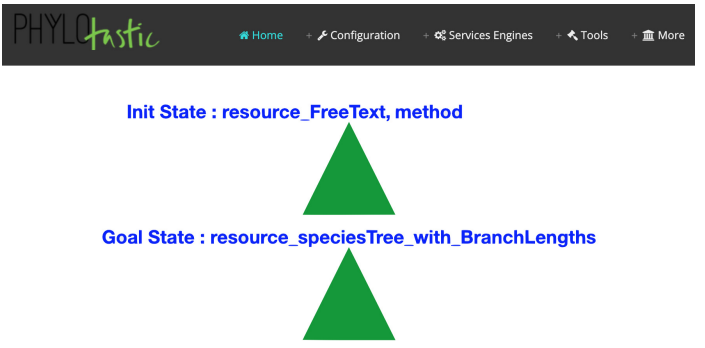

Fig. 12. Initial input and desired output configuration.

Once the input and output components have been determined, the user can use the function *Generate Workflow* of this module to generate the completed workflow. Figure 13 illustrates the generated workflow based on the user's request. In addition,

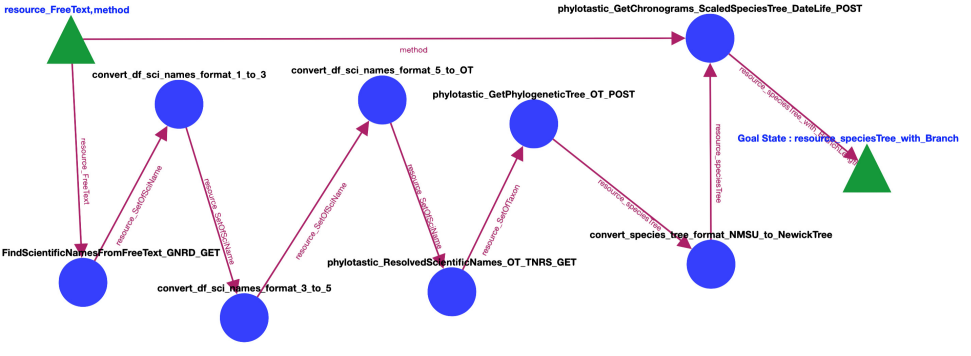

Fig. 13. Original workflow with input/output configuration.

if users want to update or modify the given workflow, they can provide the modification requirements (Sec. 4) using this same module and resubmit it to the system using the *Recomposition Workflow* function, to get the updated workflow. In our example, if the user wants to use the service *ResolvedScientificNames_GNR_TNRS_POST* to provide taxonomic names resolution instead of the service *ResolvedScientificNames_OT_TNRS_GET*, included in the original workflow, they can configure to avoid *ResolvedScientificNames_OT_TNRS_GET* and include *ResolvedScientificNames_GNR_TNRS_POST*, using the *Avoidance services* and *Inclusion services* functions, respectively. Figure 14 displays the successful updated workflow based on this modification requirement.

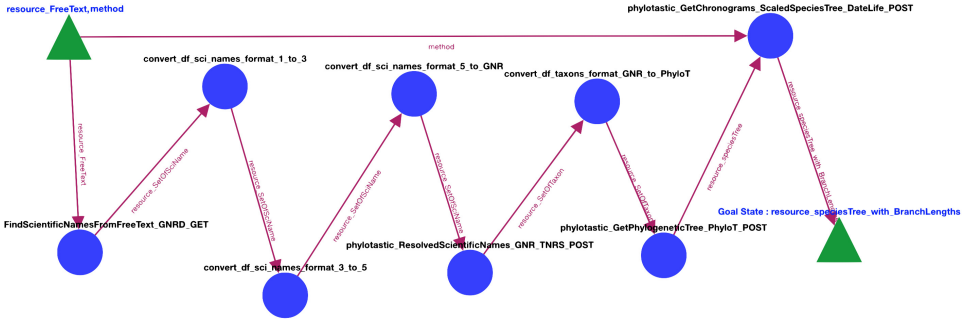

Fig. 14. Updated workflow with modification requests.

6. Conclusion and Future Work

This paper provides an overview of the Phylotastic system. The system has been designed to provide a flexible and extensible architecture to facilitate access and reuse of authoritative phylogenetic knowledge. The system provides easy retrieval of relevant phylogenies for species named in documents and Websites, through a flexible portal. The architecture allows the users to also design their own analysis protocols, through a high-level graphical interface and an automated configuration system, which automatically creates an executable workflow from a high level, and possibly incomplete, specification. The Phylotastic architecture implements both the portal and the configuration system using Web services — supported by a service registry and a comprehensive ontology to describe services and the artifacts manipulated by such services. The system has been developed and validated in collaboration with a community of evolutionary biologists, originated from a working group part of the National Evolutionary Synthesis Center (NESCENT). The working group provided the initial concept [33]. The original idea of Phylotastic is derived from the realization of the limited level of reuse of authoritative phylogenetic knowledge and the complexity of integrating phylogenetic trees within a more complex analysis pipeline. The current emphasis of Phylotastic is to develop educational materials that integrate the use of the Phylotastic portal.

The portal is accessible at `portal.phylotastic.org` while the overall Phylotastic project has a dedicated site at `www.phylotastic.org`. This research directions for the project include the following:

- Refinement of the Web services registry, to facilitate addition of new services; the registry currently provides also APIs that can be used in a number of programming languages (e.g. Python, R) to allow programmatic access to the services. On the other hand, the API does not support, yet, the search of services in the registry (e.g. using terms of the Phylotastic ontology as search keywords);
- Use of workflow refinements based on soft constraints (e.g. preferences);

- Several phylogenetic Web services tend to be fragile; the team is currently working in developing a more sophisticated approach to handle failures of services during execution. We are exploring solutions that allow the automated generation of new workflows upon failure; the new workflow should achieve the same goal as the failed one and reuse as much as possible the parts of the original workflow that have been successfully executed before failure.

Acknowledgments

The research has been supported by NSF grants 1833630, 1458595, and 1345232. The authors thank Dr. Stoltzfus, Dr. O'Meara, and Dr. Mozzherin for their help.

References

1. WSDL, http://www.w3.org/TR/wsdl.
2. BioPerl, 2018, https://bioperl.org/.
3. Celery: Distributed Task Queue, 2019, http://www.celeryproject.org/.
4. B. Chisham, B. Wright, T. Le, T. Son and E. Pontelli, CDAO-store: Ontology-driven data integration for phylogenetic analysis, *BMC Bioinf.* **12** (2011) 98.
5. D. De Jonghe, *The Complete NGINX Cookbook — 2019 Edition* (O'Reilly, 2018).
6. A. J. Dobson, Lexicostatistical grouping, *Anthropol. Linguist.* **11** (1969) 216–221.
7. S. Dustdar and W. Schreiner, A survey on web services composition, *Int. J. Web Grid Serv.* **1**(1) (2005) 1–30.
8. J. A. Eisen and P. C. Hanawalt, A phylogenomic study of DNA repair genes, proteins, and processes, *Mutation Res., DNA Repair* **435**(3) (1999) 171–213.
9. J. Felsenstein, Phylogeny inference package (version 3.2), *Cladistics* **5** (1989) 164–166, http://evolution.genetics.washington.edu/phylip.html.
10. H. A. Gleason, Counting and calculating for historical reconstruction, *Anthropol. Linguist.* **1** (1959) 22–32.
11. T. Gruber, Toward principles for the design of ontologies used for knowledge sharing, *Int. J. Hum.-Comput. Stud.* **43**(5–6) (1995) 907–928.
12. C. E. Hinchliff *et al.*, Synthesis of phylogeny and taxonomy into a comprehensive tree of life, *Proc. Natl. Acad. Sci. USA* **112**(41) (2015) 12764–12769.
13. W. Huber *et al.*, Orchestrating high-throughput genomic analysis with Bioconductor, *Nat. Methods* **12** (2015) 115–121.
14. S. Kumar, K. Tamura and M. Nei, MEGA: Molecular evolutionary genetic analysis software for microcomputers, *Comput. Appl. Biosci.* **10**(2) (1994) 189–191.
15. A. Lagares Lemos, F. Daniel and B. Benatallah, Web service composition: A survey of techniques and tools, *ACM Comput. Surv.* **48**(3) (2016) 33.
16. D. A. Liberles and M. L. Wayne, Tracking adaptive evolutionary events in genomic sequences, *Genome Biol.* **3**(6) (2002).
17. D. Maddison and K. Schulz, Tree of Life Web Project, 2007, http://tolweb.org.
18. D. Maddison, D. L. Swofford and W. P. Maddison, NEXUS: An extensible file format for systematic information, *Syst. Biol.* **464**(4) (1997) 590–621.
19. C. Mora, D. P. Tittensor, S. Adl, A. G. Simpson and B. Worm, How many species are there on Earth and in the ocean? *PLoS Biol.* **9**(8) (2011) e1001127.

20. T. H. Nguyen, T. C. Son and E. Pontelli, Automatic web services composition for phylotastic, in *Practical Aspects of Declarative Languages* (Springer Verlag, 2018), pp. 186–202.
21. T. H. Nguyen, T. C. Son and E. Pontelli, Phylotastic: An experiment in creating, manipulating, and evolving phylogenetic biology workflows using logic programming, *Theor. Pract. Log. Program.* **18**(3–4) (2018) 656–672.
22. D. J. Patterson, J. Cooper, P. M. Kirk, R. L. Pyle and D. P. Remsen, Names are key to the big new biology, *Trends Ecol. Evol.* **25**(12) (2010) 686–691.
23. W. H. Piel, M. J. Donoghue and M. J. Sanderson, TreeBASE: A database of phylogenetic knowledge, in To the interoperable, "*Catalog of Life*" *with Partners Species 2000 Asia Oceanea*, J. Shimura, K. L. Wilson and D. Gordon (eds.), Research Report from the National Institute for Environmental Studies No. 191, Tsukuba, Japan (2002), pp. 41–47.
24. M. Pistore, P. Traverso and P. Bertoli, Automated composition of web services by planning in asynchronous domains, in *Int. Conf. Automated Planning and Scheduling*, 2005, pp. 2–11.
25. Pivotal Software, RabbitMQ Open Source Message Broker, 2019, https://rabbitmq.docs.pivotal.io/37/topics/intro.html.
26. A. Prlic *et al.*, BioJava: An open-source framework for bioinformatics, *Bioinformatics* **28**(20) (2012) 2693–2695.
27. A. Ranta, Grammatical framework, *J. Funct. Program.* **14**(2) (2004) 145–189.
28. J. Rao and X. Su, A survey of automated web service composition methods, in *Semantic Web Services and Web Process Composition*, 2004, pp. 43–54.
29. E. Reiter and R. Dale, *Building Natural Language Generation Systems* (Cambridge University Press, 2000).
30. D. Ringe, T. Warnow and A. Taylor, Indo-European and computational cladistics, *Trans. Philol. Soc.* **100**(1) (2002) 59–129.
31. E. Sirin and B. Parsia, Planning for semantic web services, in *Semantic Web Services Workshop at ISWC*, 2004, pp. 33–40.
32. C. A. Stewart *et al.*, Cyberinfrastructure Software Sustainability and Reusability: Report from an NSF-funded Workshop. Technical report, Indiana University, 2010.
33. A. Stoltzfus *et al.*, Phylotastic! Making tree-of-life knowledge accessible, reusable and convenient, *BMC Bioinf.* **14** (2013) 158.
34. A. Stoltzfus *et al.*, Sharing and re-use of phylogenetic trees (and associated data) to facilitate synthesis, *BMC Res. Notes* **5** (2012) 574.
35. E. A. Stone and A. Sidow, Physicochemical constraint violation by missense substitutions mediates impairment of protein function and disease severity, *Genome Res.* **15**(7) (2005) 978–986.
36. A. Taylor, T. Warnow and D. Ringe, Character-based reconstruction of a linguistic cladogram, in *Historical Linguistics*, Vol. 1 (Benjamins, 2000), pp. 393–408.
37. The CherryPy Team, CherryPy: A Minimalist Python Web Framework, 2019, https://cherrypy.org/.
38. J. L. Thorne, Models of protein sequence evolution and their applications, *Curr. Opin. Genet. Dev.* **10**(6) (2000) 602–605.
39. E. Tillier, L. Biro, G. Li and D. Tillo, Codep: Maximizing co-evolutionary interdependencies to discover interacting proteins, *Proteins* **63**(4) (2006) 822–831.
40. E. Tillier and T. Lui, Using multiple interdependency to separate functional from phylogenetic correlations in protein alignments, *Bioinformatics* **19**(6) (2003) 750–755.
41. R. Vos, NeXML: Phylogenetic data in XML, 2008, http://www.nexml.org/.
42. C. O. Webb and M. J. Donoghue, Phylomatic: Tree assembly for applied phylogenetics, *Mol. Ecol. Notes* **5**(1) (2004) 181–183.

43. C. R. Woese and N. R. Pace, Probing RNA structure, function, and history by comparative analysis, in *The RNA World* (Cold Spring Harbor Laboratory Press, 1993).
44. Z. Yang, PAML: A program package for phylogenetic analysis by maximum likelihood, *Comput. Appl. Biosci.* **13**(5) (1997) 555–556.