Biologie

Software Components for increased Data Reuse and Reproducibility in Phylogenetics and Phylogenomics

Inaugural-Dissertation
zur Erlangung des Doktorgrades
der Naturwissenschaften im Fachbereich Biologie
der Mathematisch-Naturwissenschaftlichen Fakultät
der Westfälischen Wilhelms-Universität Münster

vorgelegt von
Ben Christoph Stöver
geb. in Düren
2018

Dekanin: Prof. Dr. Susanne Fetzner
Institut für Molekulare Mikrobiologie und Biotechnologie
Westfälische Wilhelms-Universität Münster
Corrensstraße 3
48149 Münster

Erster Gutachter: Prof. Dr. Kai F. Müller
Institut für Evolution und Biodiversität
Westfälische Wilhelms-Universität Münster
Hüfferstraße 1
48149 Münster

Zweiter Gutachter: Prof. Dr. Dietmar Quandt
Nees-Institut für Biodiversität der Pflanzen
Rheinische Friedrich-Wilhelms-Universität Bonn
Meckenheimer Allee 170
53115 Bonn

Tag der mündlichen Prüfung: 7.11.2018

Tag der Promotion: 13.12.2018

Contents

The following tables of content and abstracts are available:

Additional abstracts on the single publications and manuscripts can be found on the first page of each respective chapter.

A Short Contents

B Publications and Manuscripts

Three **additional manuscripts** describing work that provided a necessary basis for this thesis are described in the **chapters 10** (page 165), **11** (page 168) and **12** (page 171). These are not planned to be published in biological journals, since their focus has a technical nature and lies outside of biology. Still, they are included in this thesis, since they describe work that was done in it and is closely related to the biological chapters.

C Detailed Contents

D List of Figures

E List of Tables

F List of Algorithms

G List of Equations

H List of Definitions

I Abstract

Many parts of the life sciences, including phylogenetics, phylogenomics or ecology, have become data-intensive due to increasingly cheaper high-throughput sequencing technologies, the digitization of large biological collections or data contributions from citizen science. An increasing number of available and computationally accessible methods for downstream analysis that produce derived data (e.g., phylogenetic trees or character data automatically extracted from images) further contributes to the production of large quantities of potentially reusable data. This opens up new opportunities for big data studies, but also creates new challenges for infrastructure and method development. To fully use the potential of available data, cyberinfrastructure for sharing and maintenance of scientific data and policies of journals and funding agencies that encourage its publication are necessary. In part, this has already been addressed by databases like *Dryad* and recommendations of an increasing number of journals and funding agencies, although additional measures still need to be taken. Equally important as the availability of scientific data is its reusability. This includes the use of open and well-defined formats, as well as the semantic annotation of data with metadata of different kinds, for an unambiguous description and links to related information and resources. These annotations should ideally be machine-interpretable to allow reliable automated data collection for large-scale studies. In addition to its value for data reuse, proper annotation can also increase the reproducibility of studies, if methods used and steps of workflows are directly documented using attached metadata. Ideally, this would be done by the researchers that produce the data, who, however, often are unfamiliar with the necessary annotation technologies like the *Resource Description Framework* (*RDF*), advanced file formats like *NeXML* or biological ontologies. Therefore, software that makes this process more convenient is a key requirement in the age of big data and the semantic web.

To address these needs regarding phylogenetic data types, two approaches are followed in this thesis, which cater to both developers of bioinformatical software and researchers from any discipline dealing, e.g., with multiple sequence alignments or phylogenetic trees at any step of their workflows. First, programming libraries are introduced that provide required reusable software components. *JPhyloIO* allows reading and writing phylogenetic data from and to various file formats through a single memory-efficient interface, while making full use of the metadata model of each format. *LibrAlign* provides flexible and easily extendible GUI components for displaying and editing biological sequences and multiple sequence alignments (MSAs) closely together with any type of attached metadata. Second, these libraries form the basis of applications newly developed here that address the described needs of researchers. At the same time, new functionality exposed through these libraries is available to all developers and enables creation or extension of software for diverse biological applications that simplify data reuse through efficient annotation.

Among the developed applications, the *Taxonomic Editor* of the *EDIT Platform for Cybertaxonomy* models taxonomic workflows and persistently links all data elements to the specimen they were derived from. This a major advantage over the traditional approach of linking all information to a taxon, because data remains reusable and interpretable if the assignment of specimens to taxa changes in taxonomic revisions. In this thesis, the *Taxonomic Editor* is extended to support molecular sequence data with help of the functionality provided by *LibrAlign* and *JPhyloIO*. The two main phylogenetic data types are addressed by *PhyDE 2* and *TreeGraph 2*, editors for multiple sequence alignments and phylogenetic trees, respectively. *PhyDE 2* is a reimplementation of the currently used version of *PhyDE* based on *LibrAlign* and *JPhyloIO*. Although it currently is in a proof-of-concept state and does not yet offer the full feature set of the previous version, its new codebase is much easier to maintain and extend and significantly simplifies the future development towards advanced metadata modeling and using the potential of the new libraries. *TreeGraph 2* offers versatile formatting and editing options in a user-friendly way and models any type of metadata associated with tree nodes and branches, while

offering a variety of options to visualize these annotations. It makes use of *JPhyloIO* to read and write phylogenetic trees and their metadata.

In addition to fostering data reuse, allowing to compare and combine results from alternative methods is another major goal of this thesis and is also closely linked to metadata modelling and increased reproducibility of studies. Many alternative methods to construct MSAs or phylogenetic trees are available and is choosing among them is usually non-trivial. As a result, researchers often need to carefully check for agreements and conflicts between results from alternative approaches and possibly also present a synthesis across alternatives. *AlignmentComparator* implements different algorithms to visually compare alternative MSAs of the same dataset in detail and allows to identify and annotate differently and identically aligned regions. It can also be used to track subsequent automatic or manual alignment changes in workflows. *TreeGraph 2* completes the required functionality by providing an interactive comparison feature for phylogenetic trees and allows to map statistical support values derived from alternative methods onto a single reference topology, thereby highlighting topological conflicts.

Together, the developed applications support visualizing, editing and comparing all major data types of phylogenetics and related fields and have the potential to allow convenient and complete modeling of necessary metadata across complete phylogenetic workflows that produce optimally reusable data in an easily reproducibly way. Easy reuse of the developed functionality is ensured by providing key functionality in separate libraries that simplify the development and extension of more tools to provide features for easier data reuse and increased reproducibility. All developed products are freely available at http://bioinfweb.info/Software.

J Zusammenfassung

Viele Bereiche der Lebenswissenschaften, darunter auch Phylogenetik, Phylogenomik und Ökologie, haben sich zu Daten-intensiven Disziplinen entwickelt. Gründe sind u.a. zunehmend günstigere Sequenzierungsverfahren mit immer höherem Durchsatz, die zunehmende Digitalisierung von großen biologischen Sammlungen und umfangreiche Datenerhebungen durch wissenschaftlichen Laien (citizen science). Die stetige Entwicklung neuer Analysemethoden, die durch steigende Rechenleistung begünstigt wird, trägt zusätzlich zur Menge von wissenschaftlichen Daten bei, die für eine Wiederverwendung interessant sind. Dies eröffnet neue Möglichkeiten für Studien auf der Basis von sehr großen Datenmengen (big data), bringt aber ebenfalls neue Herausforderungen an die Entwicklung von notwendiger Infrastruktur und Werkzeugen mit sich. Um das Potential der großen vorhandenen Datenmenge optimal nutzen zu können, braucht es digitale Infrastruktur um wissenschaftliche Daten zu teilen und langfristig verfügbar zu halten, aber auch Bestimmungen von Verlagen und Drittmittelgebern, die zu deren öffentlicher Zugänglichmachung motivieren. Datenbanken wie *Dryad* und entsprechende Empfehlungen von einer zunehmenden Zahl wissenschaftlicher Zeitschriften und Drittmittelgeben sind erste Schritte in diese Richtung, aber weitere sind notwendig. Genauso wichtig wie die Verfügbarkeit von wissenschaftlichen Daten ist auch ihre Wiederverwendbarkeit. Dies umfasst v.a. die Verwendung von offenen und eindeutig definierten Formaten und die semantische Annotation mit unterschiedlichen Arten von Metadaten um diese eindeutig zu beschreiben und relevante Informationen und Ressourcen zu verknüpfen. Solche Annotationen sollten idealer Weise automatisiert interpretierbar sein, um eine zuverlässige Datensammlung durch automatisierte Systeme für Arbeiten mit einer breiten Datengrundlage zu ermöglichen. Darüber hinaus kann die Reproduzierbarkeit von Studien erhöht werden, indem Ergebnisse unmittelbar mit Metadaten verknüpft werden, die Informationen über die verwendeten Methoden und Arbeitsabläufe zur Datengenerierung enthalten. Diese Annotation kann am besten von den Wissenschaftlern, die diese Daten produzieren, durchgeführt werden. Allerdings sind diese oft nicht vertraut mit notwendigen Annotationstechnologien, wie dem *Resource Description Framework* (*RDF*), leistungsfähigen Dateiformaten wie *NeXML* oder biologischen Ontologien. Deshalb ist wissenschaftliche Software die einen solchen Prozess vereinfacht und dabei trotzdem sicherstellt, dass Annotationen, die eine optimale Wiederverwendbarkeit ermöglichen, eine dringende Notwendigkeit im Zeitalter von Big Data und dem semantischen Web.

Um diese Anforderungen in Bezug auf phylogenetische Datentypen zu erfüllen, werden in dieser Arbeit zwei Ansätze verfolgt, die sowohl bioinformatischen Softwareentwicklern, als auch Wissenschaftlern aus allen Bereichen, die z.B. mit Multisequenzalignierungen oder phylogenetischen Bäumen arbeiten, zugutekommen. In einem ersten Schritt wurden Programmbibliotheken entwickelt, die notwendige wiederverwendbare Komponenten bereitstellen. Eine davon ist *JPhyloIO* und erlaubt das Lesen und Schreiben unterschiedlicher phylogenetischer Dateiformate über eine einheitliche Speicher-effiziente Schnittstelle. Es wurde dabei besonderer Wert auf die vollständige Unterstützung der Metadatenmodelle aller Formate gelegt. *LibrAlign* ist eine weitere Bibliothek, die flexible und leicht erweiterbare Komponenten für grafische Oberflächen zur Verfügung stellt, die das Anzeigen und Bearbeiten biologischer Sequenzen und Multisequenzalignierungen in direkter Kombination mit entsprechenden Metadaten erlauben. Diese Bibliotheken bilden in einem zweiten Schritt die Basis für neue Anwendungen, die die beschriebenen Bedürfnisse von Forschern erfüllen. Gleichzeitig steht durch sie die neue Funktionalität auch anderen Entwicklern zur Verfügung und erlaubt diesen, Software für viele weitere biologische Anwendungen zu schreiben oder zu erweitern, die ebenfalls Datenwiederverwendung durch Annotation mit Metadaten erleichtert.

Der *Taxonomic Editor* der *EDIT Plattform für Cybertaxonomie* modelliert taxonomische Arbeitsabläufe, wobei alle Datenelemente darin dauerhaft mit dem Beleg verknüpft werden, aus dem sie ursprünglich erzeugt wurden. Dies ist ein entscheidender Vorteil gegenüber der klassischen Herangehensweise, bei

der Informationen lediglich einem Taxon zugeordnet werden. Denn nur so bleiben Datenelemente weiterhin verwendbar und interpretierbar, wenn sich Zuordnungen zwischen Belegen und Taxa durch taxonomische Revisionen später ändern. Der *Taxonomic Editor* wurde in dieser Arbeit um Komponenten zur Verarbeitung molekularer Sequenzdaten, auf der Basis von *LibrAlign* und *JPhyloIO*, erweitert. Die beiden wichtigsten Datentypen der Phylogenetik werden von *PhyDE 2* und *TreeGraph 2* modelliert, bei denen es sich jeweils um einen Editor für Multisequenzalignierungen (MSAs), bzw. phylogenetische Bäume handelt, die beide in dieser Arbeit entwickelt wurden. *PhyDE 2* ist eine Neuimplementierung des bislang verwendeten *PhyDE* auf der Basis von *LibrAlign* und *JPhyloIO*. Obwohl erst eine basale Version von *PhyDE 2* zur Verfügung steht, die noch nicht den vollen Funktionsumfang ihres Vorgängers erreicht hat, ist die neue Implementierung deutlich besser wartbar und erweiterbar. Sie erleichtert die zukünftige Weiterentwicklung von *PhyDE* hin zu einer umfangreichen Metadatenmodellierung und erlaubt die neuen Bibliotheken optimal zu nutzen. *TreeGraph 2* bietet umfangreiche und nutzerfreundliche Funktionen zur Bearbeitung und Formatierung von phylogenetischen Bäumen und modelliert beliebige Arten von Metadaten, die an Äste oder Knoten eines Baums gebunden sein können. Es ermöglicht weiterhin die Visualisierung solcher Daten auf vielfältige Weise und verwendet die Funktionen von *JPhyloIO* zum Lesen und Schreiben phylogenetischer Bäume zusammen mit ihren Metadaten.

Neben der Erhöhung der Wiederverwendbarkeit von wissenschaftlichen Daten, stellt die Entwicklung von Software zum Vergleich und zur Integration von Ergebnissen aus alternativen Analysemethoden ein weiteres wesentliches Ziel dieser Arbeit dar. Dieses ist ebenfalls eng mit der Modellierung von Metadaten verbunden und trägt zur Erhöhung der Reproduzierbarkeit wissenschaftlicher Studien bei. Es existieren zahlreiche verschiedene Methoden zum Generieren einer Multisequenzalignierung oder zur Rekonstruktion eines phylogenetischen Baums und es ist meist nicht direkt zu entscheiden, welche Methode die besten Ergebnisse für welchen Anwendungsfall liefert. Folglich müssen Wissenschaftler oft unterschiedliche Ergebnisse detailliert vergleichen und auf Übereinstimmungen und Konflikte untersuchen und möglicherweise auch die Ergebnisse mehrerer Verfahren kombiniert darstellen. Dazu wurde u.a. *AlignmentComparator* im Rahmen dieser Arbeit unter Verwendung der Komponenten von *LibrAlign* entwickelt. Dieser implementiert unterschiedliche Algorithmen, die einen detaillierten visuellen Vergleich von alternativen Multisequenzalignierungen ermöglichen, wobei unterschiedlich und identisch alignierte Bereiche schnell identifiziert und annotiert werden können. *AlignmentComparator* kann ebenfalls verwendet werden, um schrittweise automatische oder manuelle Modifikationen einer Alignierung über einen Arbeitsablauf nachzuverfolgen. *TreeGraph 2* komplettiert die entwickelte Funktionalität durch eine interaktive Vergleichsfunktion für phylogenetische Bäume und erlaubt außerdem Stützwerte aus unterschiedlichen Verfahren an einem Baum darzustellen und mögliche topologische Konflikte hervorzuheben.

Gemeinsam erlauben die entwickelten Anwendungen die Visualisierung, Bearbeitung und den Vergleich der wichtigsten Datentypen der Phylogenetik und verwandter Disziplinen und bieten das Potential für eine vollständige Modellierung notwendiger Metadaten über komplette phylogenetische Arbeitsabläufe hinweg, die einfach wiederverwendbare Daten erzeugen und leicht reproduzierbar sind. Die Verfügbarmachung weiter Teile der entwickelten Funktionalität in separaten Bibliotheken wird darüber hinaus die Entwicklung und Erweiterung weitere Software fördern, die Funktionen zur erleichterten Datenwiederverwendung und erhöhter Reproduzierbarkeit bietet. Alle entwickelten Produkte sind unter http://bioinfweb.info/Software frei verfügbar.

1 General Introduction

This thesis presents the development and implementation of related pieces of software that allow researchers to store, visualize, edit and compare the major types of phylogenetic and taxonomic data and model meaningful metadata annotations. The following major goals guided the development:

1. Foster the annotation of phylogenetic data with **meaningful and unambiguous metadata,** which is a key requirement to **increase** its **reusability and** the **reproducibility** of studies.
2. Allow researchers to consider results from **alternative analysis** methods by providing applications to **efficiently compare, combine and present** them.
3. Ensure maximal **reusability of the developed functionality** by making the components of general use flexible and accessible to the bioinformatical community as independent libraries.

To ensure that the developed software can be of maximal use for the scientific community, easy access and log-term availability (via the bioinfweb portal, chapter 10, page 165) as well as extensive documentation were other important criteria. All developed software is provided under open-source licenses to allow efficient reuse, while ensuring that possible derived products remain freely available.

The following sections (1.1 and 1.2) elaborate on the motivation for this thesis and how the developed software contributes to achieve the described goals, while section 1.3 provides an overview on how the different chapters and the applications and libraries described there are related and built-up on each other. Figure 1.1 further illustrates this relation and provides a condensed overview.

1.1 Fostering data reuse and reproducibility by providing software that simplifies annotating phylogenetic data with necessary metadata

1.1.1 Metadata annotation is important to make data accessible and reusable

Similar to fields like high energy and nuclear physics or geosciences, many parts of the life sciences have become data-intensive fields [1]. This includes organismic and biodiversity-related disciplines, such as phylogenetics, taxonomy or ecology [2,3] as well as related fields of molecular biology, especially genomics or genome evolution. Making phylogenetic data more reusable is important for these fields and every discipline where data is analyzed in a phylogenetic context, evolutionary aspects are part of studies, and alignments by homology or phylogenetic trees are needed at some point during analysis. Understanding biodiversity and the relationships between organisms is data-intensive due to the number of species [4] and the complexity of their interactions in combination with the increasing amount of available data due to increasingly cheaper high-throughput sequencing technologies [5], data contributions of barcoding initiatives [6–8], the ongoing digitization of biological collections [9,10], and large-scale data acquisition (e.g., related to monitoring biodiversity) in citizen-science [11–14]. The availability of faster processing units to perform more advanced downstream analyses and to try multiple alternative methods and parameter sets leads to even more (derived) data based on the increasing amount of primary data, potentially multiplying the amount with value for reuse in subsequent studies. The increasing success of deep learning approaches, e.g. combined with computer vision methods, also for biological purposes [15–17] is made possible by the availability of large amounts of data and such methods could be even better utilized in more types of studies if more structured metadata would be provided with the available data. The derived data produced by such automated approaches increases the amount of data available for potential reuse even further. Mass-extraction of, e.g., morphological character states from the increasing variety of digitalized specimens or organism photos generated by citizen-science is a possible example, that could also bring morphology to the big data age and make it accessible to some computational methods, previously only applicable to molecular data.

All these developments open up new perspectives for studies making use of big data, but only if the produced data is both accessible and reusable. Cyberinfrastructure for long-term storage and public access together with policies of journals and funding agencies enforcing the publication of data together with each study are necessary [18,19], but equally important is the proper annotation of published data with meaningful and unambiguously formulated metadata [20–22] to make it searchable and efficiently reusable. Ideally, annotations should be machine readable and interpretable, while describing the data with respect to a variety of different aspects. This would enable automated collection of large amounts of data that exactly fit the purpose and requirements of a new study, to avoid time-consuming large-scale manual data selection that would significantly hamper such data-intensive studies or even make them impossible. Therefore, data that is not published in appropriate formats and properly annotated is inaccessible for a growing fraction of future studies and will be less useful to contribute to scientific progress.

1.1.2 Metadata annotation is important to increase reproducibility

Beyond its importance for increasing reusability, proper metadata attachment can also help in increasing the reproducibility of studies. This is true both for primary and derived data. Examples of relevant information to be attached includes links from sequences to raw data, e.g., from different sequencing technologies, unambiguous taxonomic IDs (e.g. *NCBI Taxonomy* [23]), or information on the specimen the data was gained from. Linking back to an individual specimen instead of, e.g., a taxon is beneficial if taxonomic revisions happen that change the assignment of that specimen to a taxon. Keeping such links is possible using the *EDIT Platform for Cybertaxonomy* described in chapter 4 (page 59, [24]).

For more derived data, like multiple sequence alignments (MSAs) or phylogenetic trees, annotations can, e.g., be used to document the workflow required to produce it. That may include the algorithm that was used and the software it was implemented in, together with its version number and specified parameter values. Possible subsequent alignment modifications can also be indicated by metadata. (See also section 1.2 and chapter 7.) A detailed list on how to make computational studies more reproducible can also be found in [25]. Available technologies that are useful for the documentation of workflows include platforms like *MyExperiment* [26] or different workflow managers [27–30].

Using these tools or completely documenting workflows using, e.g., batch files, is advisable and increases reproducibility. Additionally, providing respective metadata directly within a created MSA or phylogenetic tree is even more helpful, since the necessary information is then directly linked to the data and easily accessible for both humans and automated data collection systems, even if the data file has been copied somewhere else and the respective workflow description was not. (Workflow documentation or batch files can additionally be linked using metadata annotations directly in the data document.)

1.1.3 Phylogenetic data formats

The *Nexus* format [31] was traditionally, and still often is used to store phylogenetic datasets that consists of taxon/OTU lists, character matrices/multiple sequence alignments, phylogenetic trees and different sets of, e.g., alignment columns or tree nodes. The combination of these types of data is sometimes also referred to as the *Nexus data model*. *Nexus* is supported by a variety of phylogenetic software and allows to include custom blocks to store, e.g., application-specific data. Other available software though only accepts simpler formats as input, such as *FASTA* or *Phylip* [32,33] for MSAs and *Newick* [34] for phylogenetic trees. (See also chapter 2.2.3 on page 38 for further details on the different formats.)

Today, *XML*-based formats like *NeXML* [35] and *phyloXML* [36] with more advanced metadata models are increasingly used, although their support in different software is still limited compared to the mentioned classic formats. *XML* documents are text files that are structured using markups (which are

called *XML* tags) delimited by the characters "<" and ">". (See Figure 5.3 on page 82 and Figure 7.7 on page 118 for example *XML* documents.) Using a technology called *XML Schema*, the structure of each type of *XML* document, i.e., the way tags may be nested and used, can be formally defined, which allows easy automated validation also of *NeXML* of *phyloXML* files. Functionality to process and manipulate *XML* is available in all major programming languages, which makes such formats easily accessible.

phyloXML models phylogenetic trees and a predefined set of metadata annotations to nodes, branches, and the tree as a whole. Additionally, custom *XML* elements (*XML* tags) can be inserted to model custom metadata. The other two parts of the *Nexus* data model, i.e., separate taxon lists or MSAs, are not directly supported, but sequences and taxonomic information can be attached to the nodes of a tree.

NeXML is more closely modeled along the *Nexus* format and supports the same set of data types. It additionally offers structures to attach any type of metadata to all its data elements, including OTUs, MSAs, trees, sequences, nodes, branches and others. *RDF* [37] (see below) is used to link data and metadata. This is a major advantage over the *Nexus* format that only allows limited textual annotations in its `NOTES` block and annotations to certain data elements were usually done using informal extensions of the format, such as different kinds of hot comments. (See chapter 2.2.4 on page 41 for further details.)

Although starting to use *NeXML* may be more complex than *phyloXML*, since new users need to be or become familiar with the concepts of *RDF* and externally defined ontologies (see next section), it provides more powerful ways of annotating more elements of phylogenetic data in a formally defined and unambiguous way.

1.1.4 Linking metadata using ontologies

As mentioned, *NeXML* allows to attach metadata to phylogenetic data elements using *RDF*. (Figure 5.3 on page 82 and Figure 7.7 on page 118 contain examples of *NeXML* documents, including metadata annotations.) *RDF* stands for *Resource Description Framework* and is a family of specifications of the *World Wide Web Consortium* (*W3C*). It allows to formulate logical statements of the form *subject-predicate-object*. The subject of a statement can, e.g., be the node of a tree in a *NeXML* or other document or any other data element, while the object may be any metadata annotation like a support value or a taxonomic identifier. The predicate must be a globally unique URI (e.g., a web address) and describes the relation between the subject and the object. A predicate with the meaning "carries bootstrap value" could, e.g., link a tree node and the numeric value "98". In contrast to a hot comment with the content "98" attached to a tree node in *Newick* or *Nexus*, it is unambiguously defined what the meaning of "98" is. (In *Newick* or *Nexus* the age of the node or any other numeric annotation might also be indicated by the same hot comment. While human users may or may not be able to guess the meaning from the context, requiring such guessing makes interpreting the data unnecessarily difficult, especially for automated systems trying to make use of the information.)

Technically predicates are just URIs. To unambiguously define the meaning for a set of URIs, an ontology is necessary. Ontologies, in this case, are sets of controlled vocabularies that allow to formulate *RDF* statements, usually regarding a certain field. Since predicates are identified by globally unique URIs, ontologies (predicates and their meaning) can be defined outside of the document that uses them, which is referred to as "externally defined ontologies".

Numerous ontologies, including many for use in different parts of the life sciences, exist to date. The *National Center for Biomedical Ontology* (*NCBO*) host a database containing descriptions of a large part of existing bioontologies and an ontology recommender that proposes useful ontologies related to a

paragraph of text or a list of keyword provided by the user [38]. It is also involved in developing and extending some of the ontologies [39]. The development of new and the extension of existing ontologies is and must be a community process that sometimes starts with the definition of minimal information standards (e.g. [40]) and then leads to the development of ontologies for certain purposes that fulfill the needs of the community. A list with examples of phylogenetically relevant ontologies is provided in Table 9.1 (page 153). It should though be noted that the community process of defining ontologies to describe phylogenetic data sufficiently under different aspects to enable effective reuse for specific questions is still ongoing and further work remains to be done to further improve the current information and to keep track with future developments.

Since *RDF* allows to formulate statements about distributed data using distributed (externally defined) ontologies, it is very well suited to be used in rapidly evolving, diverse and data-intensive disciplines [41]. It allows rapid and discipline-specific development without the need of a single (potentially slow) authority that defines one central standard, while still allowing to keep all produced data easily reusable, accessible and semantically interpretable. *NeXML* allows to make efficient use of *RDF* for describing phylogenetic data. Together, these two technologies can play a central role in letting phylogenetics benefit from the advances of the semantic web.

1.1.5 Public databases

Different public databases have been established in the past, contributing to the necessary cyberinfrastructure for data reuse. The sequence databases in the *International Nucleotide Sequence Database Collaboration* (*DNA Data Bank of Japan* (*DDBJ*), *European Nucleotide Archive* (*ENA*), and *GenBank* at *NCBI*) were among the pioneers in the field. They are commonly used and the publication of data in them is enforced by many journals and funding agencies. The *Dryad* data repository [42] was more recently established and allows to submit sets of any type of data files linked to a scientific publication. These files are recommended to be in open formats (as, e.g., listed in the *FairSharing* database [43]). *TreeBASE* [44] is more specific for phylogenetics and allows to upload character matrices or multiple sequence alignments together with phylogenetic trees inferred from them in *Nexus* format [31]. Data also must be linked to a scientific publication.

The publication of data that is more complex and derived than simple sequences (e.g., in databases like *Dryad* and *TreeBASE*) is less frequently enforced by journals but some start recommending it, e.g., [45,46]. As a result, the fraction of published data regarding, e.g., phylogenetic trees is still smaller than for sequence data [19,22,47] and most phylogenetic trees are still published only as images in publications. Although software exists that tries to extract such trees [48,49], this way of providing phylogenies is far from optimal and allows no reliable metadata interpretation at all. Data that is not uploaded to public databases but is stated to be available upon request by the authors of a publication is frequently not accessible [50] after authors cannot be contacted anymore since they have left academia or simply do not reply to requests.

Since *Dryad* does not impose many constraints on the format of the uploaded data, annotating it (e.g., in *NeXML* format) with any kind of relevant metadata to increase its reusability is possible, although having such data in a database does not automatically mean that it is searchable, but it is a necessary step. Databases that model the uploaded data more closely, as *TreeBASE* does by explicitly focusing on MSAs, phylogenetic trees and a set of attached metadata like taxonomic IDs and publication DOIs are more easily searchable for the modeled data but at the same time less flexible with regard to new types of data and metadata. In *TreeBASE* it is currently not intended to upload *NeXML* files that provide additional metadata not modeled in the database (e.g., on the workflow used to infer a phylogenetic tree), although it can output *Nexus* uploads in *NeXML* format that contain a set of modeled and partly automatically obtained metadata. Both these and other database are very helpful contributions to

making phylogenetic data more reusable and *Dryad* in principle already allows to annotate uploaded data with any type of metadata. In the future development, a greater focus on metadata annotation to allow more efficient identification of relevant datasets could further improve the cyberinfrastructure. Providing a triple store [51] for *RDF* annotations (e.g., contained in uploaded *NeXML* files) that allows semantic queries within repositories such as *Dryad* could be an important step further in that direction.

Phylotastic! is a web service that returns a tree containing a set of queried taxa that is based on known phylogenies. This allows efficient data reuse in studies that require a phylogenetic context in principle, but the project is currently in a proof-of-concept state [52]. The *Open Tree of Life* project tries to provide a comprehensive phylogeny of all organisms on earth, based on all available phylogenies in databases and additionally allows to upload published phylogenies directly to the project's own database [53]. It suffers from the fact that a large part of published phylogenies are not easily reusable, e.g., since they are only available as images [53]. Both services provide access to the trees available in other databases and *Open Tree of Life* also provides some metadata, such as links to studies with supporting or conflicting trees. They also significantly contribute to a better reusability of phylogenetic trees but currently do not focus on a general flexible metadata annotation for reuses beyond the phylogenetic tree topology (and probably cannot do this due to the too limited amount of available properly annotated data).

1.1.6 Software components developed in this thesis provide missing functionality to fostering data reuse and increasing reproducibility

Besides improvements and extensions in the other necessary parts of the cyberinfrastructure and policies to allow efficient data reuse as described above, providing software to process phylogenetic data while making the necessary metadata annotation as easy as possible, is a key step. The less effort is required for researchers to provide data in suitable formats annotated with necessary metadata, the more would be willing to follow this practice and the easier respective policies of journals and funding agencies could be implemented. Despite the great usefulness to tackle the challenges of the big data age, only few applications for use in phylogenetic workflows are available to date that provide necessary features by supporting the flexible metadata model of the *NeXML* format or comparable features.

Instead of only developing or extending one or more software products that support a flexible metadata model independently, this thesis focuses on implementing necessary functionality in general software libraries (cf. aim 3 above). This way, the new functionality can be used in different applications developed in this thesis and at the same time by all other bioinformaticians to extend their software and address the described needs. Implementing general libraries requires more effort than directly extending single applications, since flexible strategies covering a maximal number of use cases are required, but that is justified by the number of applications that are made possible or can be improved. (Reusability is not only important for scientific data, it also is for scientific software components.)

The libraries *JPhyloIO* and *LibrAlign* have been developed to provide the metadata-related functionality that is necessary to enable easier data reuse. Based on them, three applications have additionally been created or extended that cover all three parts of the *Nexus* data model. The *Taxonomic Editor* of the *EDIT platform for Cybertaxonomy* models taxonomic workflows with their data derived from specimens and therefore covers the taxa/OTU part, while *PhyDE 2* and *TreeGraph 2* are editors for multiple sequence alignments and phylogenetic trees, respectively. Section 1.3 below describes in detail how these libraries and applications are related and contribute to the aim of simplifying data reuse.

1.2 Providing tools to compare, combine and present results from alternative analyses

Multiple sequence alignments and phylogenetic trees are two core data types of phylogenetics and their creation is at the center of most respective workflows. Beyond that, both play a significant role in nearly all other disciplines of the life sciences. Numerous downstream analyses and (phylo-) statistical tests are based on them (e.g., [54–58], or see Box 1 in [59] for a more complete list of examples). A large variety of alternative methods for both multiple sequence alignment [60–68] and phylogenetic inference [69–74] exist and have different advantages for different purposes, but it is often not trivial to decide which method to use for which problem. The single methods can furthermore produce different results depending on numerous parameters the user can adjust. (Examples include generally used parameters like gap penalties in MSA algorithms or substitution models used by phylogenetic inference software as well as more method-specific parameters like the `F`-option of the MSA algorithm *PRANK* that allows to avoid penalizing gaps resulting from insertions and not from deletions multiple times along the guide tree.)

As a result, researchers often need to compare the results from different alternative methods or parameter sets to inspect how these influence their analyses. Since there is no general agreement on which method produces the best results for which purpose, comparing the results and possibly simultaneously presenting them, indicating where they agree and where they are in conflict, is an advisable practice. Software for conveniently and efficiently comparing MSAs and phylogenetic trees is therefore of great use when dealing with all types of biological problems that involve these datatypes. In addition, software to compare MSAs and trees can be used to track and identify subsequent manual or automatic modifications made during a workflow when comparing the outputs of different steps. Therefore, comparison software can also increase reproducibility of studies when used in this context.

In this thesis, two applications are developed to address these needs. *AlignmentComparator* offers several ways to visualize and annotate differences between alternative MSAs and *TreeGraph 2* provides features to interactively compare alternative phylogenetic trees and map support values from different analyses onto one topology. A more detailed introduction to the comparison functionality of these two applications and how they are related to the other software of this thesis is provided in section 1.3.3 below.

1.3 How the developed software components combine

The chapters of this thesis (i.e., the publications and manuscripts) are grouped into three parts. Part I (page 31) describes the bioinformatical libraries, while Part II (page 57) deals with the applications based in these. As shown in Figure 1.1, the audience of the chapters in Part I are bioinformaticians who develop scientific software, while Part II addresses researchers working in phylogenetics or many other fields, who need such tools. Part III (page 163) contains three additional chapters that describe a web portal and two general-purpose (not mainly bioinformatical) libraries that provide the basis for all other projects. It is located at the end of the thesis, since it has a supplementary character regarding its biological and bioinformatical aims but is still relevant for the thesis as a whole.

1.3.1 Basic general-purpose components

The basic general-purpose components, described in Part III, are shown in the first green part of Figure 1.1. The *bioinfweb* web portal (chapter 10, page 165) on the top of the figure provides the platform for all developed software. It offers download pages, documentation and source code repositories, making the development and the use of the software possible in the first place. Much of the functionality has been added within this thesis.

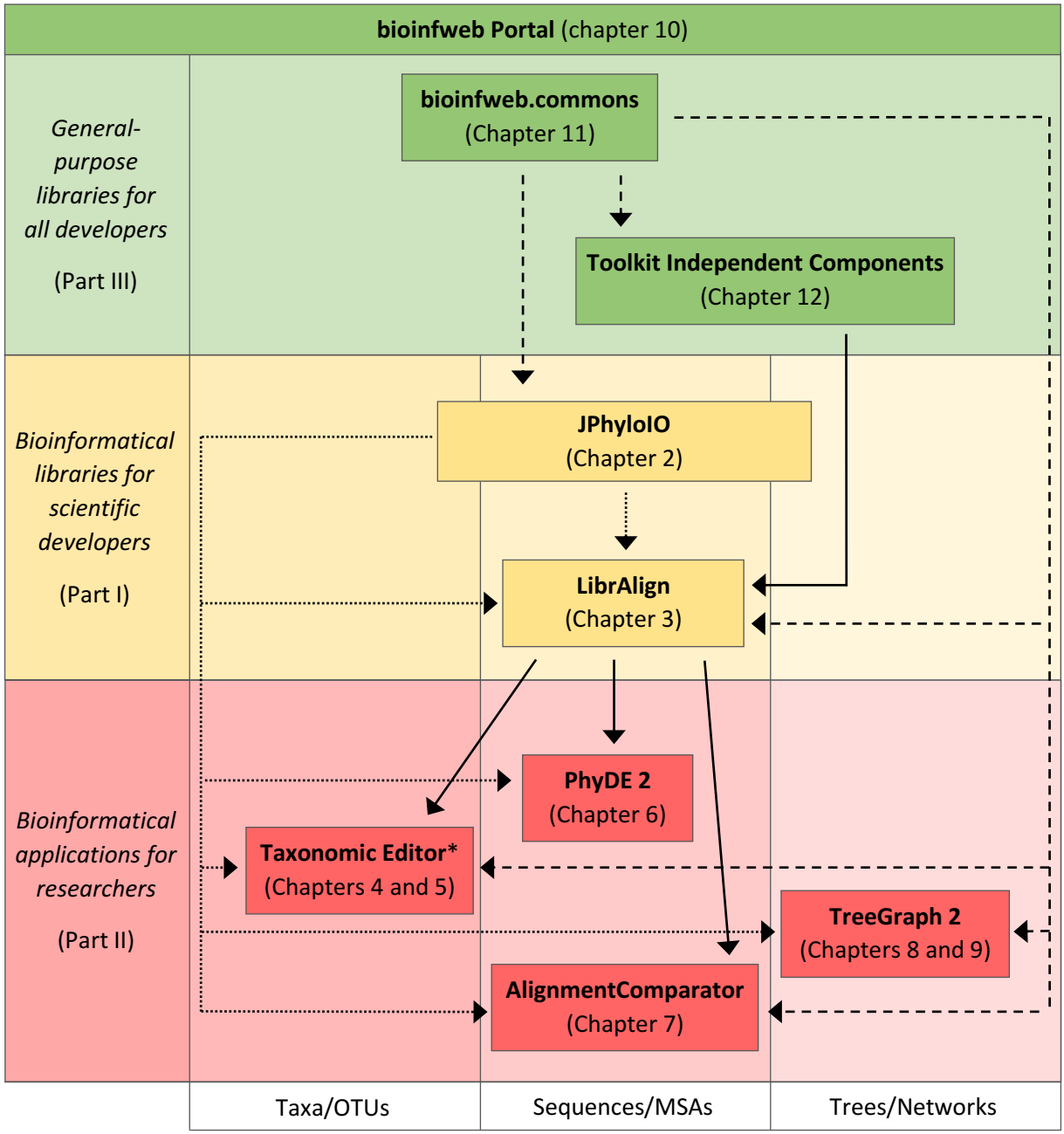


Figure 1.1 General concept of the thesis and relation between the chapters and projects

To reach the goals of this thesis different libraries and applications are developed. All of them, except for the Taxonomic Editor*, are hosted in the* bioinfweb *portal. The green components are general-purpose libraries that are not specific for bioinformatics. This includes* bioinfweb.commons *with its general tool classes for various different purposes that are shared between all projects and* Toolkit Independent Components *that provides technical GUI solutions for* LibrAlign*, which could also be used by other software. The yellow part contains software libraries that provide functionality specific for bioinformatics and includes* JPhyloIO *to read and write phylogenetic data and metadata as well as* LibrAlign *that provides GUI components to display and edit MSAs and attached raw and metadata. The applications developed in this thesis are shown in red and the arrows indicate how all components are based on each other. Due to its general functionality, all products are based on* bioinfweb.commons*, while all bioinformatical components that read and write data are based in* JPhyloIO*. On the horizontal axis, the main datatypes of phylogenetics are listed and each software is positioned depending on which of these datatypes it deals with.*

**Note that only the molecular parts of the* Taxonomic Editor *were developed in this thesis and the software as a whole is developed and hosted at the* Berlin Botanical Garden and Botanical Museum*.*

bioinfweb.commons (chapter 11, page 168) is a general-purpose programming library that has been developed together with the other software. It contains functionality that was necessary for one of the other libraries or applications and was potentially of general use beyond the current project. Performing such implementations in a separate project allows to easily reuse them and all software in this thesis depends on some of the provided features, which is why Figure 1.1 shows links from *bioinfweb.commons* to all other projects.

TIC is described in the third chapter of Part III (chapter 12, page 171) and stands for *Toolkit Independent Components*. This library provides abstractions over the two major *Java* GUI toolkits, *Swing* and *SWT*. It has been developed together with *LibrAlign* and allows to provide all of *LibrAlign*'s GUI components for both *Swing* and *SWT*, which is a requirement, since the *Taxonomic Editor* uses *SWT* and the other applications that make use of it are *Swing*-based. *TIC* is maintained separate from *LibrAlign* because its functionality is potentially useful elsewhere. Since the library is available on the *bioinfweb* portal under an open-source license, it can be used by other developers and possibly also for exposing some components of *TreeGraph 2* in a separate library in the future.

1.3.2 Bioinformatical libraries

The middle part of Figure 1.1 (yellow) shows the two bioinformatical libraries *JPhyloIO* and *LibrAlign* that are described in Part I of this thesis. *JPhyloIO* (chapter 2, page 33) provides a generalized and memory-efficient access to a variety of different phylogenetic file formats (cf. sections 1.1.3 above and 2.2.3 on page 38) with a special emphasis on supporting the full metadata model of each, while still providing one single interface that is independent of the specific formats (chapter 2.2.4, page 41). This fulfills a basic requirement to address goal 1 of this thesis (page 20) for all developed software and ensures maximal interoperability by supporting still widely-used traditional formats together with recent advanced one in one step. As indicated by the arrows in Figure 1.1, *JPhyloIO* is used to read and write data by *LibrAlign* and all applications.

LibrAlign (chapter 3, page 46) provides flexible GUI components and model implementations to display and edit single sequences and complete multiple sequence alignments. It allows to attach so-called data areas to these sequences and MSAs, which can be considered the GUI counterparts of metadata elements attached, e.g., in *NeXML* files, using predicates from externally defined ontologies. For sequence and alignment data, *LibrAlign* provides components that are as flexible with respect to attaching metadata as the *RDF*-based model of *NeXML* is. Applications dealing with such data are therefore easily enabled to model any kind of metadata directly in their GUI by simply implementing new or reusing third party data areas and combining them with *LibrAlign*'s existing components. The GUI elements are used by the *Taxonomic Editor* to handle contig alignments of marker sequences, provide the basis of the MSA editor *PhyDE 2* and are integrated into *AlignmentComparator* to display comparisons between alternative MSAs.

The yellow and the red sections of Figure 1.1 are horizontally divided into three parts and the position of the rectangles representing each software indicate which part of the *Nexus* data model a product deals with. *JPhyloIO* spans all three areas, since it can read and write data related to taxa or OTUs, character matrices or MSAs and phylogenetic trees or networks. *LibrAlign* only deals with sequences, MSAs and their metadata is therefore positioned in the center part.

1.3.3 Applications for researchers

The chapters in Part II describe different aspects of the applications that have been developed or extended in this thesis as shown at the bottom red part of Figure 1.1. These cover all three areas of the *Nexus* data model and therefore major parts of many phylogenetic, taxonomic and many related workflows.

The *Taxonomic Editor* of the *EDIT Platform for Cybertaxonomy* (also called *EDITor*) is mainly developed at the *Botanical Garden and Botanical Museum* (*BMGM*) at the *FU Berlin* and existed before this thesis started. Chapter 4 (page 59, also published as [24]) describes the general concept of the application and its platform, including the way it models alpha-taxonomic workflows. Its major advantage is linking all modeled elements back to the specimen they were initially derived from. This way, all data can still be easily used and interpreted while performing taxonomic revisions, which contributes to goal 1 (page 20). During this thesis, the *EDITor* was extended to model molecular sequence data that is generated from probes of specimens. Sanger sequencing trace files with their base call sequences and contig alignments of them to create complete marker sequences are now supported and persistently linked back to their source specimen(s). *LibrAlign* is used to display the contig alignments and its editing functionality allows to create and modify them. The pherogram raw data is displayed using special data area instances attached to the single read sequences. Exporting contig alignments to various formats is achieved using *JPhyloIO*, while the *NeXML* export contains metadata further describing the relation between the sequences to allow optimal reuse. (Details in Figure 5.3, page 82.) Chapter 5 (page 79) describes the new molecular components of the *EDITor* in detail. Since it deals with data derived from specimens that is used, e.g., for taxon diagnosis, the *EDITor* is positioned in the taxa/OTU part of the *Nexus* data model in Figure 1.1.

PhyDE [75] is an alignment editor for phylogenetic purposes, which was already in use before the start of this thesis. It was developed between 2005 and 2010 and offered, amongst other things, versatile features to manually edit MSAs. With the development of *LibrAlign* that provides components with similar functionality, the decision was made to reimplemented the application based on the library in this thesis, also because the old code base was difficult to maintain and extend. The new implementation is called *PhyDE 2* and described in chapter 5 (page 79). Since the development of *PhyDE 2* is not the primary focus of this thesis, the initial version does not yet have the full feature set of the previous version, but already offers new ones, like supporting more formats with the help of *JPhyloIO*. The new application of made as a proof-of-concept for creating a fully functional MSA editor using the components of *LibrAlign* and *JPhyloIO* and to lay the basis for the future development of *PhyDE* that will also offer a flexible and advanced metadata model necessary to fulfill the needs of goal 1, described above.

AlignmentComparator (chapter 7, page 90) provides functionality to visually compare alternative MSAs of the same dataset and allows a detailed inspection and annotation of regions with conflict or agreement between different methods. It therefore contributes to goal 2 of this thesis (allowing researchers to take the results of alternative methods into account, page 20). *AlignmentComparator* uses GUI components provided by *LibrAlign* that allow to display multiple MSAs underneath of each other to output MSA comparisons including related metadata. *JPhyloIO* contributes functionality to import MSAs from different formats and to store the comparison results in *NeXML* using the flexible metadata model of that format. Resulting *NeXML* files can be provided with a study to document alignment modifications and to increase its reproducibility and *AlignmentComparator* therefore also contributes to goal 1 of this thesis.

Both *PhyDE 2* and *AlignmentComparator* are tools that cover the sequence/MSA part of the *Nexus* data model as indicated in Figure 1.1.

Phylogenetic trees that are inferred, e.g., from MSAs processed with *PhyDE 2* and *AlignmentComparator*, can then be visualized, compared and further annotated with the phylogenetic tree editor *TreeGraph 2*. It therefore covers the third part of the *Nexus* data model, as shown in Figure 1.1. A first version of the editor is described in chapter 8 (page 128) which has been published as [76]. By the time of submission of this thesis, it is already widely used inside and outside the phylogenetic community and highly cited (see chapter 9.1 on page 137 for concrete numbers), which indicates a demand for its

functionality and its contribution to scientific progress. This is one of the reasons for its further extension described in chapter 9 (page 137). *TreeGraph 2* features various editing and selection options. The metadata model allows to attach an unlimited number of annotations to tree nodes and branches and the editor can visualize these in many ways, including automatic coloring or resizing of different tree elements and using different types of branch labels. Since the development of *TreeGraph 2* started before the development of *JPhyloIO*, its initial metadata model used string keys used in *Nexus* hot comments to attach metadata, instead of *RDF* predicates. With the completion of *JPhyloIO*, the metadata model of *TreeGraph 2* was refactored to additionally support nested annotations linked by *RDF* predicates to address the needs of goal 1 even better. (See chapter 9.3.6 on page 151 for further details.)

With respect to goal 2 of this thesis, the application additionally provides functionality to compare phylogenetic trees. The published version described in chapter 8 allows to combine support values from different analyses (e.g., maximum likelihood and Bayesian approaches) and to show them together on a single tree. If the topologies resulting from multiple approaches differ, the algorithm can identify corresponding branches and map support values accordingly. If topological conflicts exist, support values from an alternative analysis are mapped onto the branch to which they are in conflict and are highlighted. (See chapter 8.3.1.1 on page 130 for details.) The comparison feature is extended in chapter 9 to support the comparison of trees with different overlapping sets of terminal nodes and an interactive tree comparison feature is introduced. The user can open a set of alternative trees and select nodes in one of them. Both corresponding nodes and support of conflicting branches is then highlighted in all other trees. This allows an inspection of topological differences in greater detail than a single figure with one selected or consensus topology that carries all mapped support values.

More details on how the developed libraries and applications lay the foundations for increased data reuse and reproducibility can be found in the following chapters.

Part I –
Fundamental software libraries to process, display and edit phylogenetic data and metadata

2 *JPhyloIO*: A Java library for event-based reading and writing of different phylogenetic file formats through a common interface

Stöver BC[1]*, Wiechers S[1], Müller KF[1]

[1]Institute for Evolution and Biodiversity, WWU Münster, Hüfferstraße 1, 48149 Münster, Germany

*Author for correspondence: stoever@bioinfweb.info

Own contribution

See section 2.6.1 on page 45.

Abstract

Today a variety of phylogenetic file formats exist, some of which are well-established but limited in their data model, while other more recently proposed ones offer advanced features for metadata representation. Most phylogenetic software currently only supports one or few different formats, while supporting more would be desirable to achieve optimal interoperability and prevent data loss by format conversions.

We developed the Java library *JPhyloIO*, which allows event-based reading and writing of the most common alignment and tree/network formats using the strategy pattern. It generalizes between their different data and metadata concepts, while still allowing full access to all features of the nine currently supported formats.

By implementing a single *JPhyloIO*-based reader and writer, application developers can support all formats. Due to an event-based architecture, *JPhyloIO* can be combined with any application data structure, and is memory efficient for large datasets. *JPhyloIO* is distributed under *LGPL*. Detailed documentation and example applications (available on http://bioinfweb.info/JPhyloIO/) significantly lower the entry barrier for bioinformaticians who wish to benefit from *JPhyloIO*'s features in their own software.

JPhyloIO enables simplified development of new and extension of existing applications that support various standards simultaneously. This has the potential to improve interoperability between phylogenetic software tools and to motivate usage of more recent powerful formats such as *NeXML* or *phyloXML*.

2.1 Introduction

Phylogenetic file formats are an integral part of phylogenetic workflows and the basis for interoperability between the software tools and databases involved. To date a variety of different file formats exist in this domain [31,32,35,36], but many of the more widely-used tools for multiple sequence alignment [60,68,77–79], phylogenetic inference [69–71,74,80,81], tree editing or databases and web services [44,52,82,83] support only one or few of these. More often than not, this enforces conversion (with possible data loss) between such formats, e.g., when chaining several such tools. For simpler interoperability, supporting more established formats simultaneously would be desirable, but currently would force developers to invest significant additional resources. Since the same resources would have to be subtracted from working on the core functionality of the application, this is usually not done.

Here we present *JPhyloIO*, a new *Java* library that allows reading and writing a variety of phylogenetic file formats through a common event-based interface that fully models the data and metadata concepts of the all formats. This enables *Java* application developers to support all these formats by only implementing a single *JPhyloIO*-based reader and writer. Consequently, classic formats supported by

many traditional tools (e.g. *FASTA*, *Phylip* or *Nexus*) as well as new formats with advanced metadata concepts (e.g. *NeXML* or *phyloXML*) can be supported in one step, without investing additional development resources or acquiring detailed knowledge on each format. At the same time, the event-based architecture of our library gives developers maximum flexibility in the design of the application data model, instead of enforcing its own, as many I/O libraries do.

Our library is mainly intended for developers of (complex) end user applications requiring a custom data structure. Although similar libraries for other programming languages would also be beneficial for the community, we decided to develop *JPhyloIO* in *Java*. The reasons are *Java*'s modular technology that allows to easily integrate and extend *JPhyloIO,* its built-in platform-independence, and it still being one of the most common languages in software development and widely used in the bioinformatics community, as testified by the numerous *Java* applications available [24,74,76,78,84–87].

2.2 Design and implementation

Phylogenetic data files usually consist of a set of taxon- or OTU lists, character matrices or multiple sequence alignments, phylogenetic trees or networks and sets of elements (e.g. character sets), including associated metadata. Not all formats can hold all of these data types (see chapter 2.2.3 on page 38 and Table 2.1 on page 40), but a general concept for data management should model them all. The aim of *JPhyloIO* is to provide a way for reading and writing that generalizes over different file formats, without imposing constraints on the data model of applications using the library. (See Figure 2.1 and Figure 2.2.) An event-based architecture (similar to iterator-based *StAX* for *XML* parsing) was chosen over a model-based approach, because representing phylogenetic data as a sequence of event objects allows compatibility to all application data structures and memory efficient processing, with only one or very few events required to be in memory at the same time.

All *JPhyloIO* readers and writers are provided by a factory that creates instances and can guess the format from a file or input stream.

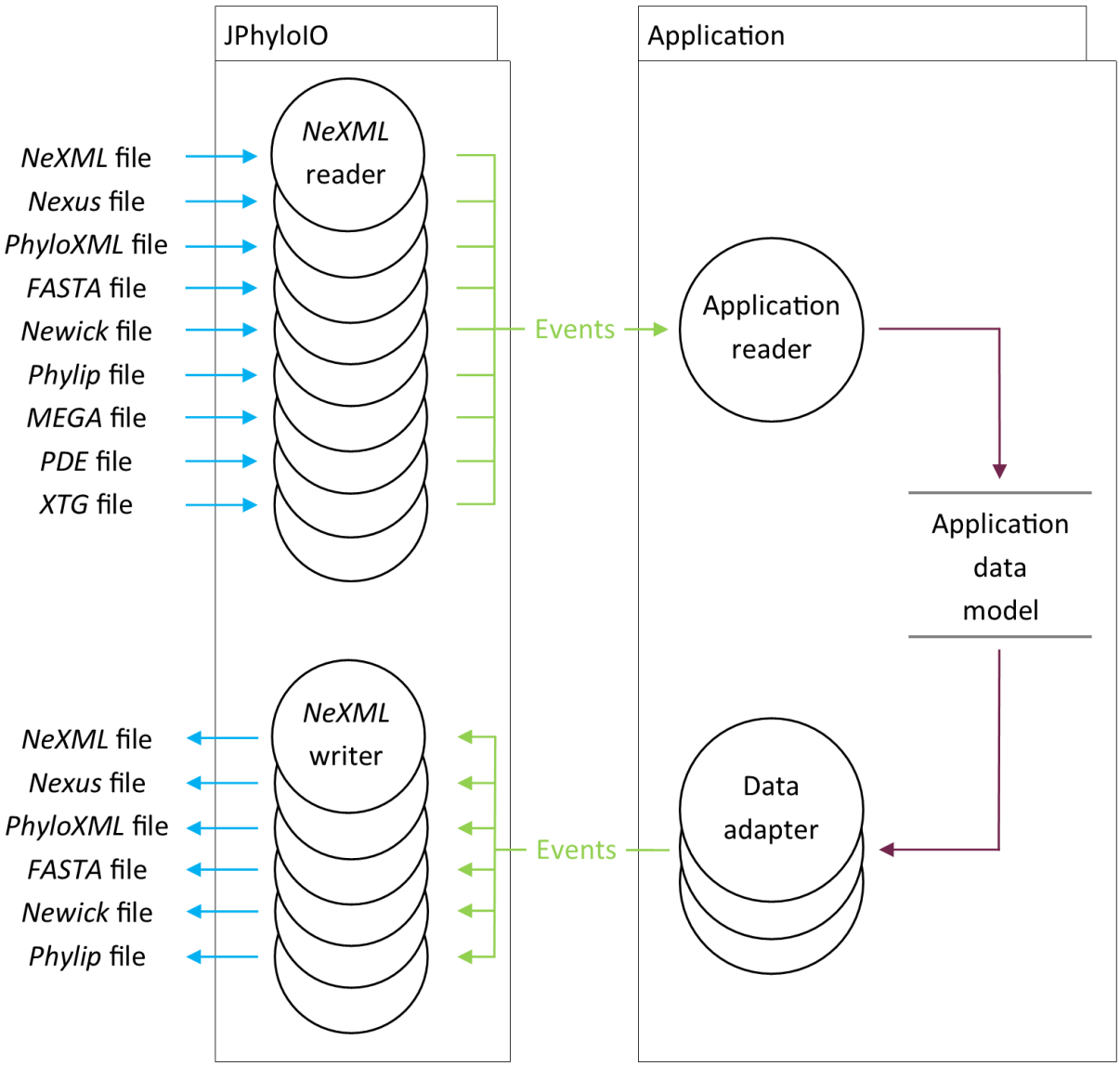


Figure 2.1 Data flow diagram showing how data is read into and written from an application data model.

JPhyloIO *contains a reader for each format that translates the contents of a file to a sequence of events that are then processed by the custom reader of an application. This reader has knowledge of the specific application data model and stores relevant information there. The writers available in* JPhyloIO *access the contents of that model using data adapters provided by the application that allow random access to the application's data model. (For supported formats specific for a single application, only readers are provided.)*

2.2.1 Event streams for reading documents

JPhyloIO's reader classes use the strategy pattern [88] to make them easily exchangeable. They translate the hierarchical data structure of a document with phylogenetic data into a linear sequence of event objects, which is formally described by the grammar in Figure 2.3. Events representing data elements that consist of smaller parts (e.g. an alignment that consists of sequences) are modelled as a pair of a start and an end event. The subsequence between these two consists of events that model the content of the data element at hand. By applying this recursively, the hierarchical structure of a document can be serialized to a linear event stream, as shown in the example in Figure 2.4. Applications using *JPhyloIO* need to implement a reader for processing the encountered events and storing relevant information in their data structure (Figure 2.1, Figure 2.2). This can be done by iterating over the event stream using (*StaX*-like) pull parsing, which allows the application to actively request events one by one and therefore keep the control flow. Classes for (*SAX*-like) push parsing are additionally available, if an inversion of control is beneficial, e.g. if multiple event listeners need to be present on

the application side. (Push parsing means that *JPhyloIO* has the control flow and calls event handlers of the application for each element of the stream. In this approach the application code cannot control when to read how far.) One such application reader implementation allows access to all supported formats, even when additional formats are added in future releases of *JPhyloIO*.

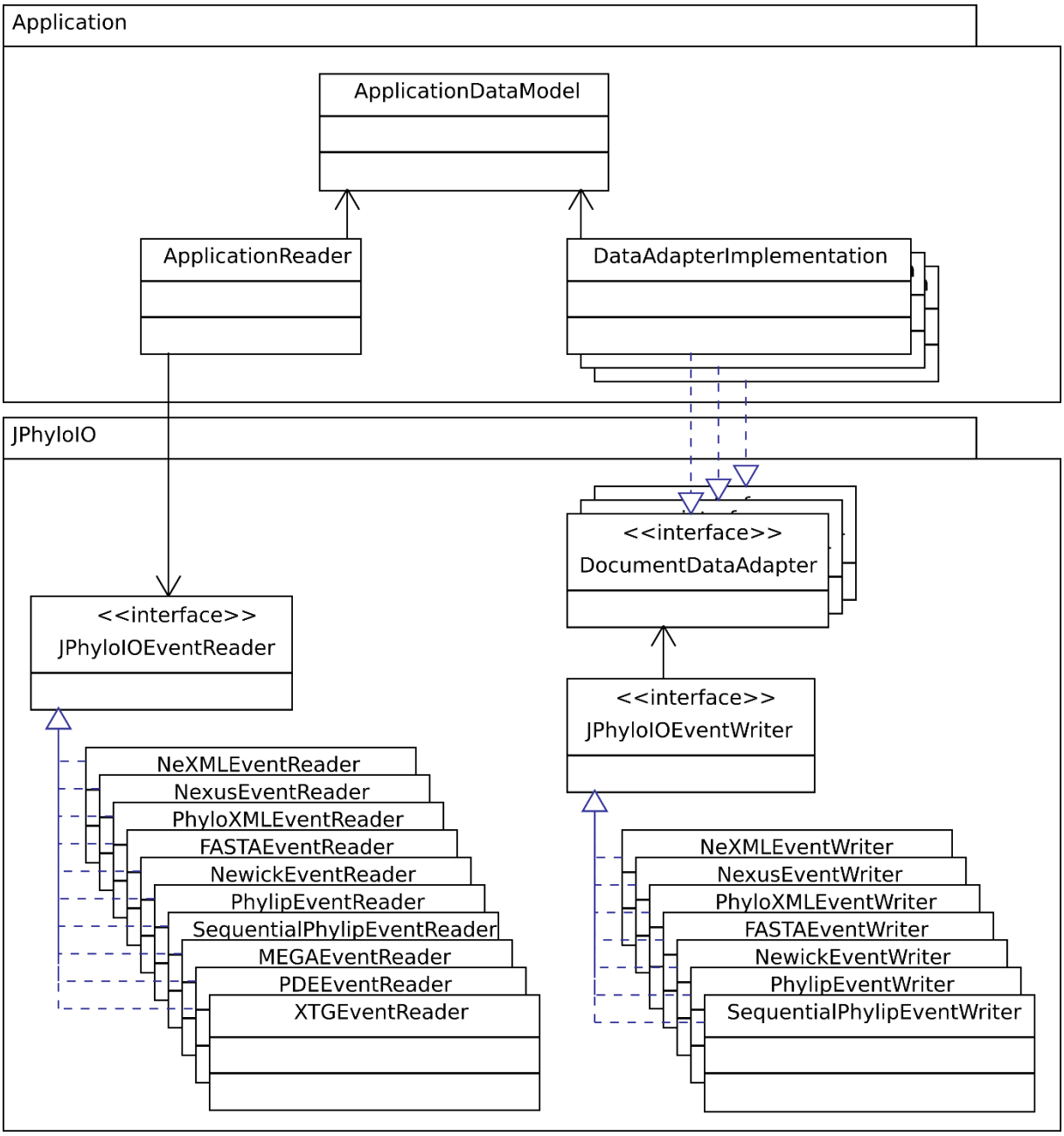


Figure 2.2 UML class diagram showing the relation between* JPhyloIO *and an application based on it.

All readers and writers implement a common interface to be easily exchangeable in the application. Event readers produce a sequence of events (see Figure 2.1) processed by an application reader class that acts as an adapter between JPhyloIO *and the application data model. Conversely, a set of data adapter implementations of the application allows the* JPhyloIO *writers to access the data. Writing needs a slightly more complex architecture than reading, because writers need to access that data in different orders depending on the target format. To achieve this, a set of data adapters (see Figure 2.5 for details) is necessary, each providing a subsequence of the whole event stream modelling a document.*

All created event objects have a string ID, which is unique inside a document's event stream and allows to link events to one another (e.g. a tree node to an OTU). References will only be made to previous events to ensure that they are already known to the application. To reduce the amount of work for application developers, the minimum information is always directly contained in an event object, so that only more complex application models will need to resolve such ID dependencies.

```
Document = "DOCUMENT.START", {DocumentContent,} "DOCUMENT.END";
DocumentContent = OTUSet | Matrix | TreeNetworkGroup | CharacterSetPart | TreeNetworkSet | MetaInfor-
    mation;

OTUList = "OTUS.START", {OTUListContent,} {OTUSet,} "OTUS.END";
OTUListContent = OTU | MetaInformation;
OTU = "OTU.START", {MetaInformation,} "OTU.END";
OTUSet = "OTU_SET.START", {SetContent,} "OTU_SET.END";

Matrix = "ALIGNMENT.START", {MatrixContent,} "ALIGNMENT.END";
MatrixContent = CharacterDefinition | TokenSetDefinition | SequencePart | CharacterSetPart | SequenceSet |
    MetaInformation;

CharacterDefinition = "CHARACTER_DEFINITION.START" {MetaInformation,} "CHARACTER_DEFINITION.END";
SequenceSet = "SEQUENCE_SET.START" {SetContent,} "SEQUENCE_SET.END";

TokenSetDefinition = "TOKEN_SET_DEFINITION.START", {TokenSetDefinitionContent,} "TOKEN_SET_DEFINI-
    TION.END";
TokenSetDefinitionContent = SingleTokenDefinition | MetaInformation;
SingleTokenDefinition = "SINGLE_TOKEN_DEFINITION.START", {MetaInformation,} "SINGLE_TOKEN_DEFINI-
    TION.END";

SequencePart = "SEQUENCE.START", {SequencePartContent,} "SEQUENCE.END";
SequencePartContent = "SEQUENCE_TOKENS.SOLE" | SingleSequenceToken | MetaInformation;
SingleSequenceToken = "SINGLE_SEQUENCE_TOKEN.START", {MetaInformation,} "SINGLE_SEQUENCE_TO-
    KEN.END";

CharacterSetPart = "CHARACTER_SET.START", {CharacterSetPartContent,} "CHARACTER_SET.END";
CharacterSetPartContent = "CHARACTER_SET_PART.SOLE" | SetContent; (* In character sets only references to
    other character sets (and not single character definitions) are using "SET_ELEMENT.SOLE". *)

TreeNetworkGroup = "TREE_NETWORK_GROUP.START", {TreeNetworkGroupContent,} "TREE_NET-
    WORK_GROUP.END";
TreeNetworkGroupContent = Tree | Network | TreeNetworkSet;
Tree = "TREE.START", {TreeOrNetworkContent,} ["ROOT_EDGE.START",] {TreeOrNetworkContent,} {NodeEdge-
    Set,} "TREE.END";
Network = "NETWORK.START", {TreeOrNetworkContent,} {NodeEdgeSet,} "NETWORK.END";
TreeOrNetworkContent = Node | Edge | MetaInformation;
Node = "NODE.START", {MetaInformation,} "NODE.END";
Edge = "EDGE.START", {MetaInformation,} "EDGE.END";

TreeNetworkSet = "TREE_NETWORK_SET.START" {SetContent,} "TREE_NETWORK_SET.END";
NodeEdgeSet = "NODE_EDGE_SET.START" {SetContent,} "NODE_EDGE_SET.END";

SetContent = "SET_ELEMENT.SOLE" | MetaInformation; (* Single elements and other sets of the same type can
    be linked using "SET_ELEMENT.SOLE". *)

MetaInformation = ResourceMeta | LiteralMeta;
ResourceMeta = "RESOURCE_META.START", {MetaInformation,} "RESOURCE_META.END";
LiteralMeta = "LITERAL_META.START", {"LITERAL_META_CONTENT.SOLE",} "LITERAL_META.END";
```

Figure 2.3 Grammar describing the event sequence generated by* JPhyloIO *readers.

These readers translate the hierarchical data structure of a phylogenetic file (e.g. a NeXML file consisting of an alignment and a tree, which again consist of sequences or nodes and edges etc.) into a sequence of events as defined by this grammar in extended Backus-Naur form. The terminal symbols (in green) represent the types of events, each of which either has a single SOLE or a START and END version, depending on whether additional data can be nested or not.

2.2.2 Data adapters for writing documents

Format-independent writing of phylogenetic data cannot be implemented as straightforward as, e.g., *StAX* writing for *XML*, since the required order of the data elements varies between the different target formats, and direct writing of an event stream (as defined by the grammar in Figure 2.3) is not possible without having to buffer large amounts of data in some cases. Therefore, we provide adapter interfaces between the application data model and *JPhyloIO* writers (Figure 2.2, Figure 2.5) that request certain subsequences of the event stream (which correspond to a grammar node in Figure 2.3) in the required order of their target format.

Implementing such data adapters may be slightly more effort for application developers than just writing a method that creates an event stream from their data model, but it has the advantage of allowing direct access in the required order for all formats and writing therefore becomes memory efficient.

2.2.3 Supported formats

As shown in Table 2.1, *JPhyloIO* supports reading and writing the majority of phylogenetic file formats, including common extensions of these. Additionally, reading of some application-specific alignment and tree formats is possible. The library imposes no restrictions on alphabets used in molecular, morphological and other character matrices, but guarantees that no invalid output for any of the target formats can be written.

Sequence data, including optional comments, can be read from and written to the *FASTA* format, with optional column indices at the beginning of each line being processed correctly. (Figure 2.6 shows an example of these special *FASTA* elements.) Writing of sequences and optional comments is supported, but generated files will never contain column indices, since these may be problematic for readers in some other software.

The *Phylip* format exists in a standard [32] and a relaxed [33] variant (the latter allows longer sequence names). Both variants can be read in interleaved and non-interleaved forms (the non-interleaved form is written for both) variants. The *Phylip* format allows sequence names only up to a certain length, resulting in the need to shorten them by *JPhyloIO* writers. In contrast to many other

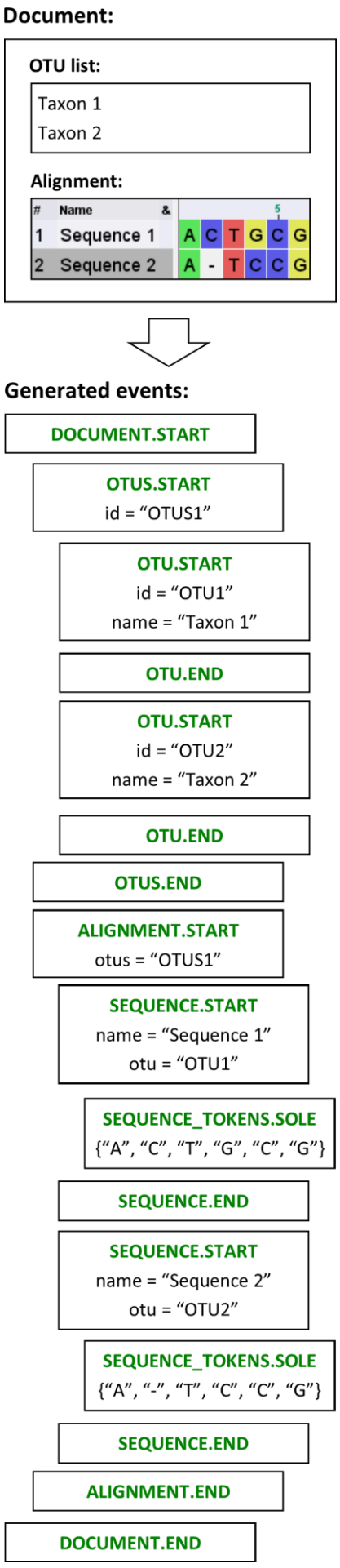


Figure 2.4 Example document with the respective event sequence. →

The document contains an OTU list and an alignment, which references this list. The event sequence is generated by a JPhyloIO *reader (see also Figure 2.2), where each box represents one event. Each has an ID in order to be referenced by subsequent events, as exemplarily shown by the OTU list and OTU start events, which are referenced by the related alignment and sequence start events.*

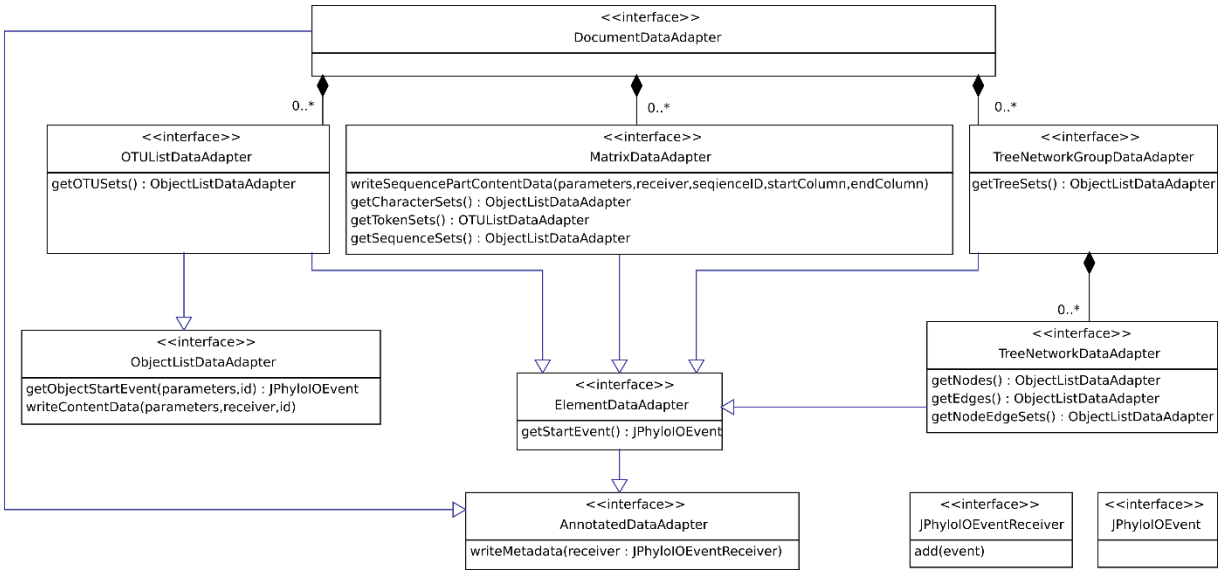


Figure 2.5 UML diagram showing the data adapter interfaces providing access to the application model for JPhyloIO writers.

From top to bottom the object relation (indicated by aggregations) is shown, while the class hierarchy can be read from bottom to top. Note that not all but only exemplary methods are shown in each interface.

The `DocumentDataAdapter` is the main adapter that provides access to other adapters modelling OTU lists, matrices and phylogenetic trees or networks. Not all application models will provide all these datatypes and therefore not need to implement all types of adapters. The format specific writer classes in JPhyloIO can access the data either by event getter methods (e.g. `MatrixDataAdapter.getSequenceStartEvent()`) with an event ID as parameter or by `writeXXX()` methods (e.g. `MatrixDataAdapter.writeSequencePart-Content-Data()`), which write a whole subsequence of the event stream to a special receiver object provided by the application. To simplify the adapter implementation for application developers only frequently used events are provided by getter methods, while the others can directly be written in a sequence by implementing an appropriate writer method. (Getter methods were introduced for cases where random access to events with known IDs is frequently necessary for writers, to avoid requesting a whole sequence, if only one event is needed. Providing some events by getter and some by writer methods in the data adapter model is a compromise between ease of implementation and runtime performance.)

Some adapters share common functionality, which is modelled by common superinterfaces, such as `AnnotatedDataAdapter` or `ElementDataAdapter`.

available software tools, this implementation ensures that all written names are unique, even if the full names only differ in characters behind the cut-off position. If sequence names were edited, the application will be informed by a translation object, mapping old to new names.

The *Nexus* format [31] is a text format consisting of blocks that contain different types of data. Each block consists of a set of *Nexus* commands. *JPhyloIO* offers readers and writers that support commands of the `TAXA` block containing taxon lists, the `DATA`, `CHARACTERS` and `UNALIGNED` blocks containing sequence and alignment data, the `TREES` block containing phylogenetic trees and the `SETS` block, containing sets of other items. One type of custom `NETWORKS` blocks containing phylogenetic networks in *eNewick* format is also supported (see below.) Sequence data can be in standard or interleaved for-

```
>Sequence 1
;Some comment
;Another comment
 0 ACGT
 5 TAGC
10 TTAGT
>Sequence 2
ACGT-ACC-TAGT
```

Figure 2.6 Example of optional FASTA elements.

The FASTA file shown here contains optional sequence comments and column numbers in "Sequence 1", while "Sequence 2" contains no optional elements. JPhyloIO allows to read and write such comments (which may appear only directly after the sequence name) and to read files containing the current column index at the beginning of each line correctly.

mat with both single character and longer tokens and ambiguous character definitions being supported. Tree nodes can be referenced by the taxon label, the taxon index, or by using a separate translation table. For the `SETS` block, character, taxon and tree sets are currently supported. The `DISTANCES`, `ASSUMPTIONS` and `NOTES` blocks are currently not supported. As *Nexus* files identify all elements by a unique label (instead of distinguishing between labels and IDs as e.g. in *NeXML*), the respective *JPhyloIO* writer edits labels to be unique if necessary, and reports such changes using the same translation object as the *Phylip* writer described above.

Table 2.1 Formats supported by JPhyloIO.

A variety of file formats used in phylogenetics are supported. These can either be text-file-based or XML formats, which is indicated in the second column. The columns in the middle show whether a format supports taxon/OTU lists, multiple sequence alignments, phylogenetic trees or networks. As shown in the two columns on the right, JPhyloIO can read and write many common formats, while application specific formats can only be read.

[1]Hot comments containing numeric or textual node and branch annotations as used e.g. by BEAST *or* MrBayes*.*
[2]Hot comments in the "New Hampshire eXtended" format, a precursor of phyloXML that allows to use a limited set of its predefined annotations.

Format	Type	OTUs	MSAs	Trees	Networks	Read	Write
FASTA	Text		X			X	X
Phylip	Text		X			X	X
Relaxed Phylip	Text		X			X	X
Nexus	Text	X	X	X		X	X
Newick	Text			X		X	X
Hot comments (Newick, Nexus)[1]	Text			X		X	X
NHX (Newick, Nexus)[2]	Text			X		X	X
eNewick (Newick, Nexus)	Text			X	X	X	
NeXML	XML	X	X	X	X	X	X
phyloXML	XML			X	X	X	X
MEGA	Text		X			X	
PDE	XML		X			X	
XTG	XML			X		X	

In addition to the initial *Nexus* standard, the `TITLE` and `LINK` commands from *Mesquite* [84] that allow linking between blocks (e.g. `TAXA` blocks can be referenced by `CHARACTERS` or `TREES` blocks) and the `MIXED` sequence datatype extension [89] from *MrBayes* [73] are recognized.

Phylogenetic trees are represented as *Newick* strings [34] in the *TREES* block of a *Nexus* document or in separate text files containing a set of *Newick* strings separated by semicolons, which are sometimes referred to as *Newick* files and are e.g. used by *MEGA* [69]. Such *Newick* files are modelled as a separate format in *JPhyloIO* that can be read and written. *Newick* tree definitions (in *Newick* and *Nexus* files) may contain metadata in hot comments, which can also be read and written. (See chapter 2.2.4 below.)

The readers for both *Nexus* and *Newick* can also read definitions of phylogenetic networks in the *Extended Newick* or *eNewick* format [90] and model the crosslink type (if specified) as metadata.

NeXML [35] is a more recent *XML* format that is inspired by *Nexus* but allows a more advanced way of linking different phylogenetic data elements (e.g. a tree node to an OTU). Additionally, it offers an *RDFa*-like way of attaching metadata to all elements (trees, alignments, nodes, sequences, …), which provides the basis for the general metadata model used in *JPhyloIO*. (See below.) Readers and writers supporting all features of the format, including its full metadata concept and automated handling of custom sequence tokens, are provided by our library.

phyloXML [36] also models complex metadata using a different concept than *NeXML*. It stores phylogenetic trees and is fully supported by *JPhyloIO*. Although *phyloXML* uses a hierarchical tree representation, it allows to specify additional clade relation tags to define phylogenetic networks that are used by *JPhyloIO*'s reader and writer.

In addition, readers for some application specific formats are available. For the *MEGA* format [69], a reader provides access to its alignment data and character sets (attached by the `LABEL`, `GENE` or `DOMAIN` commands of the *MEGA* format). Multiple sequence alignments and attached metadata from *PDE* files produced by the alignment editor *PhyDE* [75] and trees, including their metadata, from *XTG* files used by the phylogenetic tree editor *TreeGraph 2* [76] can be read as well.

2.2.4 Generalization over different metadata concepts

A major feature of *JPhyloIO* is to provide a general way for attaching metadata to any element in a phylogenetic data set, thereby abstracting away the different metadata concepts found in the individual supported formats. In our opinion, the *RDF*-based metadata tags used by *NeXML* [35] are the most powerful way of modelling metadata and therefore became the foundation of our general concept. This *RDF*-based concept allows to link external resources and to represent trees of hierarchical metadata annotations by distinguishing between resource and literal metadata, and both may be (recursively) nested inside a resource metadata element.

Following this structure, *JPhyloIO* provides a resource and a literal metadata event class. As shown in Figure 2.3, the event grammar allows nesting sequences of metadata events (represented by the grammar node `MetaInformation`) in all data elements. Metadata can either be a resource or a literal metadata event. The literal metadata objects may be simple values (e.g. numbers or strings) or complex *XML* data, modelled by a sequence of respective events. For such data, our library provides adapter classes between *JPhyloIO* and both iterator- and cursor-based *StAX* readers and writers to empower application developers to possibly reuse existing code for *StAX*-based reading and writing of respective data.

While the metadata representation in *NeXML* is by definition identical to *JPhyloIO*'s metadata model, reading and writing of other formats requires a translation to the respective format-specific model. *FASTA* and *Phylip* do not support metadata, so the respective writers ignore provided attachments and log warnings.

phyloXML does not use an *RDF*-like concept, but offers a fixed set of metadata, stored in special *XML* tags. To access such data in *JPhyloIO*, we defined *RDF* predicates for each predefined metadata element for internal use in *JPhyloIO*, to allow identifying the p*hyloXML* tags in our *RDF*-based model. In addition, *phyloXML* offers ways to freely attach metadata by (i) `property` tags to attach simple annotations (e.g. strings, numeric values or *URI*s) to trees, clades or sequences and (ii) custom *XML* structures added to a whole document, a tree, a clade or some of the predefined annotation tags. *JPhyloIO* makes use of all these features to attach metadata not linked by p*hyloXML*-specific predicates. In combination, this allows to read and write all modelled metadata. Since representing custom hierarchical *RDF* metadata (different from the predefined p*hyloXML* annotation types) is not possible in this format, parts of it will be ignored during writing and respective warnings (similar to *FASTA* and *Phylip*) will be logged. Different strategies on how to translate a full *RDFa* annotation tree into *phyloXML* are offered by *JPhyloIO* and can be selected using a writer parameter.

For attaching metadata to nodes and branches in *Newick* strings [34], two extensions that make use of hot comments (comments which contain actual data) are supported by *JPhyloIO*. One is "New Hampshire eXtended" or *NHX* [85,91], a precursor of p*hyloXML* that allows to use a limited set of its predefined annotations, identified by the respective p*hyloXML* predicates in *JPhyloIO*.

The other extension, used by e.g. *TreeAnnotator* from the *BEAST* package [74] and recent versions of *MrBayes* [73], allows to attach numeric or string values (or arrays of these) to nodes and branches using a free string identifier. These identifiers differ from the *RDF* predicates (used in *NeXML*), since they can have any form and do not need to be *URI*s. To solve this, all meta-events in *JPhyloIO* can carry a string identifier and an *RDF* predicate as alternative descriptions of their relation to their subject. If a string representation is needed for writing and was not provided, the local part of the predicate *CURIE* will be used.

By supporting these two annotation concepts, *JPhyloIO* allows to read and write metadata from and to *Newick* and *Nexus* files. As in *phyloXML*, hierarchical metadata cannot be written and warnings will be logged.

JPhyloIO also reads metadata from the application-specific *XTG* and *PDE* formats. Both formats may contain a fixed set of metadata for some of their elements and according predicates in namespaces for internal use are defined to identify these (the same way as for *phyloXML*). The *XTG* format and *TreeGraph 2* [76] additionally provide the functionality to attach numeric or string annotations to each node or branch of a tree using a string identifier, which are also supported. Basic annotations present in the *MEGA* format (e.g. a description text for a matrix) are read as well.

2.2.5 Ways to extend JPhyloIO

All readers and writers in *JPhyloIO* implement common interfaces and several abstract implementations of these are available that provide shared functionality, e.g. specific for processing text or *XML* formats. It is therefore easy for third party or application developers to add new readers and writers for additional (custom) formats that integrate seamlessly with the architecture of the library and can directly be used with all *JPhyloIO*-dependent code.

For creating complex *Java* objects from metadata event sequences or to write them back, an interface with a set of default implementations for common types is provided, which can be used for additional custom implementations.

Nexus-related classes are designed to use individual handlers for all *Nexus* blocks and commands, allowing to easily add support for new or custom *Nexus* elements in third party modules.

2.3 Discussion

2.3.1 Comparison with other libraries

Other libraries exist for the *Java* programming language that support reading or writing of alignment or tree formats. *Forester* [92] allows to read and write alignments in *FASTA*, *Phylip* and *Nexus* and phylogenetic trees in *phyloXML*, *Nexus*, and *NHX.* Phylogenies in the *Tree Of Life Response Format* [83] can be read. The *NeXML* format with its powerful metadata model is not supported and no generalization over the different metadata models exists. The tree readers implement a common interface, but there is no such interface for reading or writing trees and alignments together. As a consequence, *Nexus* files containing sequence and tree data need to be processed multiple times independently. Unlike *JPhyloIO*, *Forester* enforces its own predefined data model, which can have disadvantages for certain use cases as discussed below. The *Nexus* `TAXA` block is only supported when writing trees but not considered for reading trees or for reading and writing alignments, while *Nexus* sets are not modelled at all. The documentation is currently limited.

In its current version 4.2.7, *BioJava* [93] includes only readers and writers for sequence data from the *FASTA* and the *GenBank* format. The *BioJava* legacy version 1.9.2 [94] provides an event/call-back based API through a common interface for some sequence formats, among them the alignment formats *FASTA* and *MSF*, but none of the other formats supported by *JPhyloIO*. Independent readers and

writers for *Phylip* and *Nexus* (including support for trees but not for sets) are available, which cannot be accessed through the event-based API. There is no support for *NeXML*, *phyloXML* or complex metadata.

In other languages, multiple format-specific APIs are available (e.g. [95,96] and many unpublished ones), some of which also generalize over different formats (e.g. [97–99]).

BIO::Phylo [100] is a *Perl* library that supports a number of alignment and tree formats, among which are also 6 of the 9 formats supported by *JPhyloIO*. Reading and writing is possible through a common interface but a predefined data structure is enforced. Metadata connected using *RDF* predicates is modelled. *phyloXML*-specific predicates are used in a similar way as in *JPhyloIO*, while the set of supported elements is less complete, as e.g., `property` and `clade_relation` tags are not, and legal custom tags are only partly supported. *BIO::Phylo* is able to read (but not write) some types of hot comment tree annotations from *Nexus*, but *JPhyloIO* supports to read and write a larger set of these.

NCL for *C++* [98] supports *FASTA*, *Newick*, *Nexus* and *Phylip.* Plans to support *NeXML* and *phyloXML* were announced in 2010, but have not been implemented as of this writing, and therefore complex metadata is not modeled. Hooks for the application to directly process a whole alignment or a whole tree are provided, but these data elements are much larger than in *JPhyloIO* (where event objects only model e.g. a short sequence part or a single tree node) and processing of large alignments or trees is less efficient in *NCL*.

Compared to the existing *Java* libraries and even libraries in other languages, *JPhyloIO* is unique in supporting a large number of formats through one common interface, while allowing memory efficient event-based processing independent of the application's data structure. In particular the generalization over different metadata models, which allows full access to such data from all formats, is currently not offered by any other *Java* library. (As mentioned above, *BIO::Phylo* allows access to a comparable range of formats in *Perl* but is not event-based.) Advanced metadata modelling is a key strength, as annotating phylogenetic data is and becomes increasingly important for reproducibility and reusability (e.g. [22,40,101]).

2.3.2 Event-based processing versus predefined library data structures

With an event-based architecture as implemented in *JPhyloIO*, application developers can decide for each event whether it should be kept in memory or not. Libraries with predefined data structures load all data from a file into memory at the same time, regardless of the applications having a need for it or not. This is especially inefficient for use cases that do not need random access to all data (e.g. determining the GC-content of large sequence data sets, searching for certain repeat motives in them or counting the occurrences of a certain node in a large set of trees, e.g. taken as samples from Bayesian phylogenetic inference). Event-based processing reduces the amount of memory needed in such cases from O(n) (linear to the dataset size, e.g. the number of nucleotides) to O(1) (constant, independent of the dataset size), since only the current or a few recent events need to be in memory at once.

For applications that need random access (e.g. alignment or tree editors), the event-based architecture is still beneficial, because these are often not interested in the total content of a file (only in, e.g., sequence data, but not trees) and therefore can directly discard unused events, which they could not when using library-specific data structures. Even more relevant for complex applications may be the flexibility regarding the data structure. Providing concrete data storage classes with a library, forces applications that need a more advanced or specific model to load the data into instances of library classes first and then copy it into their own specific data structure. This way, the data of at least one file will be in memory twice, which may become a problem for large data sets. Such a problem does

not occur with *JPhyloIO*, since event data can directly be stored into any application-specific data structure.

With this in mind, we acknowledge that predefined model implementations may be beneficial for simpler scripts and tools, because developers will not have to deal with implementing their own data structure. Not needing a specific data structure is rather rare when developing more complex applications, which are the main target for *JPhyloIO*.

It is also easy to combine *JPhyloIO* with established model standards like the sequence model of *BioJava* while it still allows to access data not modelled by such libraries. As an alternative to directly implementing to fill e.g. *BioJava* model classes from *JPhyloIO* event data, our library can be combined with *LibrAlign* (see chapters 2.3.3 and 3).

2.3.3 Current usage

JPhyloIO was developed closely together with *LibrAlign* (chapter 3), a *Java* library providing powerful and reusable GUI components for displaying and editing multiple sequence alignments and attached raw- and metadata and is used in there for I/O.

The *Taxonomic Editor* of the *EDIT platform for Cybertaxonomy* ([102], chapter 5) manages taxonomic workflows and their data, while persistently linking character data to preserved individual specimens [24]. *AlignmentComparator* (chapter 9.6.2) compares alternative multiple sequence alignments of the same dataset. Both make use of *JPhyloIO* and *LibrAlign* for reading and writing alignments and attached metadata. *LibrAlign* and *JPhyloIO* also provide the basis for new versions of the alignment editor *PhyDE* ([75], chapter 6) and they are currently used by our group in the development of tools for the evaluation of automated multiple sequence alignments for phylogenetic purposes (chapter 13.2.2, page 186).

The tree-related functionality of *JPhyloIO* is the basis in extending the metadata support (chapter 9.3.6) of the phylogenetic tree editor *TreeGraph 2* [76] to import from and export to all supported tree formats making use of the generalized metadata model of the library and allowing to apply externally defined ontologies in *TreeGraph*. Versions 2.11.0 and later already use *JPhyloIO* for importing phylogenetic trees and their metadata from *NeXML* (chapter 9.3.3).

2.3.4 Future development

JPhyloIO will remain under active development in the future. According to the needs of depending software, the library will be adjusted to future changes of the supported formats and be extended to support additional formats. API stability is a key aspect and releases follow the established standard of semantic versioning [103].

In addition to the current abstraction over different formats, the abstraction over (future) metadata ontologies relevant for phylogenetics (e.g. possible in *NeXML*) can become a focus. If a critical number of established ontologies will be present, it may be interesting to extend *JPhyloIO* to model equivalent or similar predicates in different ontologies to allow translating between them and to access knowledge in a general way.

2.4 Conclusion

The field of phylogenetics as well as biological sciences as a whole would strongly benefit from a more widespread use of data annotation and respective formats. Unambiguously describing and processing morphological characters and states, documenting voucher information in collections or providing links to raw data would be some of many examples where metadata annotation (e.g. using *RDF*) and externally defined ontologies can lead to increased reproducibility of workflows and reusability of data. We developed *JPhyloIO* with the intention to simplify writing new and extending existing software that is aimed at achieving this goal by fully supporting metadata-rich formats. Maximal interoperability and

downwards compatibility is ensured by the combined support for advanced and more traditional formats by using the format-independent data model in *JPhyloIO*. Its event-based architecture makes integration with any existing application data structure easy and memory-efficient.

2.5 Data Accessibility

Source codes and binary distributions are available under the terms of the GNU Lesser General Public License 3 from http://bioinfweb.info/JPhyloIO/. This website also provides an extensive documentation, including a detailed *JavaDoc* and a set of example applications (see http://r.bioinfweb.info/JPIODemo).

2.6 Declarations

2.6.1 Author contributions

Ben Stöver conceived the concept, implemented the software and wrote the manuscript; Sarah Wiechers contributed to the implementation the software; Kai Müller and Sarah Wiechers contributed to the manuscript. All authors gave final approval for publication.

2.6.2 Acknowledgements

Funded in part by grant MU 2875/3-1 to Kai Müller by the German research foundation (DFG). We are grateful to four anonymous reviewers for their helpful comments. The developers and users of the *EDIT platform for Cybertaxonomy* at the Berlin Botanical Garden and Botanical Museum tested the integration of *JPhyloIO* with the *Taxonomic Editor*, which is highly appreciated. We thank the contributors to the open source projects used by *JPhyloIO* (*Apache commons*, *OWL API*, *JUnit*, *Hemcrest*).

3 *LibrAlign*: A flexible Java GUI library for displaying and editing multiple sequence alignments and attached raw- and metadata data

Stöver BC[1]*, Müller KF[1]

[1]Institute for Evolution and Biodiversity, WWU Münster, Hüfferstraße 1, 48149 Münster, Germany

*Author for correspondence: stoever@bioinfweb.info

Own contribution

See section 3.6.1 on page 55.

Abstract

Background: Applications from all parts of the life sciences often require the processing of DNA sequence or alignment data, which entails a need for components to visualize and edit such data, ideally in intuitive, user-friendly graphical user interfaces. In many cases, this includes handling application- or domain-specific raw- or metadata that should be displayed and edited together with the sequence(s). A programming library that provides reusable implementations of such functionality and remains easily customizable and extensible would significantly simplify the development of such applications.

Results: Here we present LibrAlign, a Java library with feature-rich and flexible GUI components to display and edit multiple sequence alignments together with related raw- and metadata. Specialized data areas can be attached to any sequence or an alignment as a whole to model raw- and metadata. While a set of such data areas is already included in the library, application developers can easily create custom implementations and use them within the provided alignment GUI components. Exchangeable token painters and a strict separation between GUI components and model classes additionally allow *LibrAlign* to be very flexible with respect to different application architectures and data structures. Access to various alignment file formats including modeled metadata is made possible using I/O components based on *JPhyloIO*. *LibrAlign* is written in *Java*, provides all its components for the two major GUI toolkits *Swing* and *SWT*, and therefore can be combined with *Eclipse RCP*.

Conclusion: *LibrAlign* enables *Java* application developers to easily integrate visualization and editing of all kinds of sequence or alignment data together with attached raw- and metadata into their applications with a minimum amount of work. The exchangeable data areas, token painters and data model implementations permit the use of *LibrAlign* in maximum number of sequence- or alignment-related use cases in all domains of the life sciences. Binaries and source codes under *LGPL 3* as well as an extensive documentation are available from http://bioinfweb.info/LibrAlign/.

3.1 Background

Sequence alignment plays a key role in numerous bioinformatical tasks in nearly all parts of the life sciences. Its applications include phylogenetic tree inference, database searching, genome assembly, identifying similar structure or predicting the function of genes, RNAs or proteins, and DNA barcoding, to name only a few. The different applications of MSA have in common that at some stage they require or benefit from user-friendly visualization or editing capabilities in different bioinformatical software.

Graphical user interface (GUI) components allowing to display and edit sequence and alignment data can therefore be of great use in a large portion of the world of bioinformatical software and such functionality should ideally be available in a general open-source programming library, to make developing of such applications easy and avoid redundant work. Due to the numerous different applications

of MSA in biology and bioinformatics mentioned above, the use cases for such a library are manifold and its components and data structures should be designed to be flexible and extensible.

Extensibility is especially important when it comes to raw- and metadata associated with alignments and sequences. The types of attached raw- or metadata used in applications from different parts of the life sciences are highly diverse (e.g. pherograms, gene function annotations, information on repeat or indel-patterns, alignment column statistics, and many more). Therefore, it is not feasible to explicitly model all possible types of raw- and metadata in a single general library. Instead, custom or third-party GUI elements for displaying sequence and alignment metadata could be enabled via plug-ins. This way, developers using such a library could design GUIs that closely model their specific tasks without having to invest resources into implementations of general sequence and alignment visualization or editing functionality.

3.2 Implementation

We developed *LibrAlign*, a *Java* library containing feature-rich and flexible GUI components providing the visualization and editing functionality envisioned above.

3.2.1 GUI component architecture

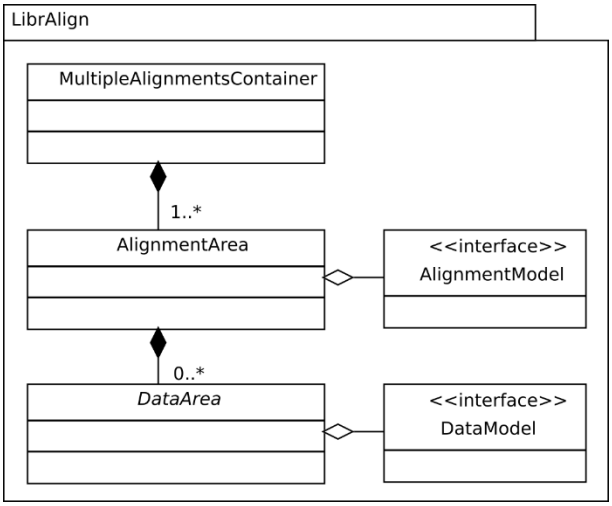


Figure 3.1 UML diagram showing the relation between the main GUI components and model interfaces

`AlignmentArea` is the main GUI component of LibrAlign*. It displays multiple sequence alignment data provided by an implementation of `AlignmentModel` and allows to edit it. An alignment area may contain multiple nested data areas (inherited from the abstract class `DataArea`) which display raw- or metadata attached to a sequence or a whole alignment. `AlignmentArea` and its nested data area implementations act as the views and implementations of `AlignmentModel` and `DataModel` act as the models in the model-view-controller paradigm. Several alignment areas can be combined within the GUI component `MultipleAlignmentsContainer` to provide a combined representation of related alignment data, but single instances of `AlignmentArea` can also be used directly. (Figure 3.2 shows a concrete example of an application window using a `MultipleAlignmentsContainer`.)*

The key part of *LibrAlign* is a GUI component called `AlignmentArea` that allows to display and edit a multiple sequence alignment, including associated raw- and metadata. The output of single tokens (e.g. nucleotides, amino acids or other sequence elements) is the responsibility of separate token painters. (Differently behaving instances can be exchanged using the strategy design pattern [88]. See “Results and Discussion” for more details.)

One or more `DataArea`s can be attached to any sequence or an alignment as a whole. These are GUI components nested within an `AlignmentArea` and represent different types of raw- and metadata. *LibrAlign* provides a set of default implementations for some types of data and developers are free to implement their own.

Several alignment areas (including their nested data areas) can be combined within a `MultipleAlignmentsContainer`. Each alignment area is independently scrollable on the vertical axis and all are scrolled together in the horizontal axis. This way *LibrAlign* allows to efficiently visually combine and compare information from different related alignments or data sets. The UML diagram in Figure 3.1 provides an overview of the relation between the different components.

3.2.2 TIC and the abstraction over Swing and SWT

LibrAlign is not limited to one specific *Java* GUI toolkit, but supports both *Swing* and the *Standard Widget Toolkit (SWT*) [104]. *Swing* is the GUI component library shipped with every *Java* virtual machine, while *SWT* is an alternative used, e.g., in applications build on the *Eclipse Rich Client Platform* (*RCP*) [105]. To achieve this, we developed *TIC* (*Toolkit independent components*, chapter 12), a separate *Java* library that allows to create GUI components independent of the target toolkit. *TIC* provides a generalization over painting and user interaction of both toolkits, so that a respective *Swing* or *SWT* component can automatically be created from a single implementation of the interface `TICComponent`. All *LibrAlign* GUI classes make use of this functionality.

Together with *TIC*, *LibrAlign* also provides functionality to overcome the maximum component size limitation *SWT* has under some operating systems and therefore allows to display and edit large alignments with native components for all environments.

Beyond the use in *LibrAlign*, *TIC* can also be used independently of *LibrAlign*, to develop GUI components that should be usable in *Swing* and *SWT* applications alike. We released *TIC* including documentation and an example application as a separate project as a service to the (scientific) community at http://bioinfweb.info/TIC/. Further details can be found in chapter 12.

3.2.3 Data model

LibrAlign realizes the model-view-controller paradigm [106], which means that classes implementing the model (storage of alignment- and attached raw- and metadata) are kept separate from the GUI components' view and controller functionality. As shown in Figure 3.1, an instance of `AlignmentArea` uses the interface `AlignmentModel` to access the alignment data (sequence names and sequence content) and the respective set of allowed tokens (e.g. nucleotides). Data areas use implementations of `DataModel` to access their data respectively. By defining these model interfaces instead of a concrete data structure, *LibrAlign* can be combined with any (custom) model implementation. Tokens (elements of a sequence) can be any set of *Java* objects, which may model discrete (e.g. amino acids) or continuous (e.g. a continuous morphological character states) data types. There is no need for token classes to be inherited from a certain class or to implement a certain interface, making it possible to use any preexisting class to model tokens. Registering event listeners is possible in order to track content changes within a model.

Special interfaces are provided that define modified views of alignment models using the decorator design pattern [88] and adapters from and to other (sequence-like) data structures defined outside of *LibrAlign*. (See "Results and discussion" for further details.) Factories (see "Factory method pattern" in [88]) automatically selecting model instances appropriate for certain sequence types (e.g. DNA, RNA, continuous characters, ...) can be used to easily instantiate the available model implementations.

3.2.4 I/O and interaction with JPhyloIO

To read and write data, *LibrAlign* makes use of *JPhyloIO* (chapter 2), a *Java* library that allows access to different alignment formats through a common event-based interface, including the full support of the metadata models of all formats (e.g. for *NeXML* [35]). The readers of *JPhyloIO* convert the contents of a file into a sequence of events, which can then be processed by the I/O classes of *LibrAlign*. For writing documents, the process is executed backwards. This way *LibrAlign*'s I/O classes act as an adapter between any *LibrAlign* model implementation and *JPhyloIO*'s readers and writers and *LibrAlign* has automatically access to all formats supported by *JPhyloIO*.

3.3 Results and discussion

Binaries and source codes of *LibrAlign*, as well as the documentation, including a detailed *JavaDoc*, code examples and a fully functional alignment editor example application, are available from http://bioinfweb.info/LibrAlign/.

3.3.1 Alignment GUI components and editing capabilities

As described above, *LibrAlign* provides GUI components displaying one or more multiple sequence alignments, which use exchangeable token painters. A set of standard token painters (e.g. displaying token names as text) is included and custom implementations can alternatively be used. DNA, RNA, protein or even protein domain alignments, or matrices with any type of (e.g., morphological) character states, can be displayed this way.

Alignment areas displaying an MSA can be set to be either fully editable (sequences and gaps can be edited), alignable (only gaps may be added or removed) or immutable. Manual editing is possible with an alignment cursor that can span any number of rows to edit these simultaneously. Different selection modes allow to select rows, columns, or any rectangular set of cells, depending on the needs of the application. Such editing operations are especially relevant for applications that allow manual alignment corrections, a common practice in creating alignments for phylogenetic inference (see [107] and references therein), or in tasks like tabulating morphological character states. (Figure 3.2 shows a minimal code example on how to use an alignment area.)

Since several alignment areas can be combined within a container component (see chapter 3.2.1), *LibrAlign* enables applications that deal with more than one MSA at a time to display related alignment data together or to compare alternative alignments, like *AlignmentComparator* (chapter 9.6.2) does (Figure 3.3). In simpler cases, multiple combined alignment areas may just be used, because each defines an independently vertically scrollable region. This way, some data areas (e.g. the sequence index or the consensus sequence area) can always stay visible while scrolling vertically through the actual alignment that is displayed by another alignment area.

Data areas are a key feature of *LibrAlign* and can be attached to any sequence or an alignment as a whole to display related raw- and metadata. The library already provides a set of data area implementations displaying the alignment column index, column sets, token frequencies, or pherograms from Sanger sequencing that were the source of aligned sequences. Beyond that, a common use case will be that application developers implement their own specialized data area(s) to model the specific type(s) of data their application deals with. Displaying gene function annotations, structural information on protein or RNA folding, or repeat patterns are some of many examples where custom data areas would be useful.

By supporting *Swing* and *SWT* (using *TIC* as described above and in chapter 12), our library can also be combined with the *Eclipse Rich Client Platform* and *Bioclipse* [108], ensuring that the maximum number of *Java* developers can benefit from it.

```
// Basic Swing example

frame = new JFrame();
frame.setLayout(new BorderLayout());

// Read an alignment file:
List<AlignmentModel<?>> models =
    IOTools.readAlignments(new File("Example.nexml"));

// Create an alignment area displaying the first alignment from the file:
AlignmentArea area = new AlignmentArea();
area.setAlignmentModel(models.get(0), false);

// Add the Swing version of alignment area to the GUI:
frame.getContentPane().add(SwingComponentFactory.getInstance().getSwingCompo-
nent(area), BorderLayout.CENTER);
```

```
// Basic SWT example

shell = new Shell();
shell.setLayout(new FillLayout(SWT.HORIZONTAL));

// Read an alignment file:
List<AlignmentModel<?>> models =
    IOTools.readAlignments(new File("Example.nexml"));

// Create an alignment area displaying the first alignment from the file:
AlignmentArea area = new AlignmentArea();
area.setAlignmentModel(models.get(0), false);

// Add the SWT version of alignment area to the GUI:
SWTComponentFactory.getInstance().getSWTComponent(area, shell, SWT.NONE);
```

***Figure 3.2 Basic usage examples of* LibrAlign *for* Swing *and* SWT.**

The first code block shows a minimal example to create a Swing *window containing an alignment area component and the resulting screenshot next to it. The displayed alignment is loaded from a file using the I/O functionality of* LibrAlign*. The example shows that* LibrAlign *allows application developers to include alignment editor components into their GUIs as easy as, e.g., editable text fields.*

The second code block is the same minimal code example for an SWT *application. As one can see, the only difference between the code for both toolkits is that* `SWTComponentFactory` *is used instead of* `SwingComponentFactory` *to create the toolkit specific component from an instance of* `AlignmentArea`*.* LibrAlign *provides true toolkit independence, as the rest of the code is (or any other code would be) identical for both toolkits, resulting in similar looking components in both types of applications.*

At the bottom a screenshot of the resulting Swing *window is shown. The respective* SWT *output would look identical. The complete example class files can be found at http://r.bioinfweb.info/LibrAlignMinCode. A more comprehensive demo application, showing how to build a simple alignment editor using* LibrAlign *that allows editing and reading and writing multiple formats is available at http://r.bioinfweb.info/LibrAlignSwingDemo.*

Figure 3.3 Screenshots demonstrating the possibilities and usage of* LibrAlign *in different applications.

(Caption on the next page.)

Figure 3.3 (continued) Screenshots demonstrating the possibilities and usage of* LibrAlign *in different applications.

The Taxonomic Editor *of the* EDIT platform for Cybertaxonomy *in the top screenshot is based on* Eclipse RCP/SWT *and models marker contig alignments including attached raw data.* LibrAlign *is used to display and edit the contig alignment and to show the pherograms as sequence-attached data areas and in a stand-alone component (bottom).* PhyDE *is an alignment editor for phylogenetic purposes now based on* LibrAlign*. The screenshot in the middle shows an opened protein alignment.* AlignmentComparator *is a* Swing *application that allows to visually compare alternative MSAs of the same data set. The screenshot at the bottom shows how such a comparison can be visualized and edited using the multiple alignments container component of* LibrAlign*. (The latest development versions of all applications are shown.)*

3.3.2 Data model

As mentioned in chapter 3.2.3, the alignment data displayed by *LibrAlign*'s GUI components is provided by model classes implementing specific model interfaces. The library contains a set of default implementations of that interface that allow (i) storing big sequence data sets in compressed token lists, (ii) modified views of other alignment models using the decorator design pattern [88] (e.g. converting DNA to RNA), or (iii) adapters to other established *Java* data structures like simple character sequences and strings, the *Java Collections Framework* [109] or the *BioJava* sequence model [93]. The compressed alignment models and decorators to provide modified views without copying data allow to efficiently handle big (genomic) sequence and alignment data sets.

Of course, application or third-party developers can also use their own custom data model implementations, e.g. to allow efficient direct access to databases or application-specific data structures. To simplify this, a set of abstract classes offering basic functionality for both model and decorator implementations is available.

Since tokens may be modeled by any *Java* object, and there is no need to implement a specific interface or to inherit from a specific class, *LibrAlign* can directly be combined with all custom or third-party classes modeling sequence tokens.

3.3.3 I/O and metadata access

Since *LibrAlign* provides I/O classes that rely on *JPhyloIO*, all of *JPhyloIO*'s alignment formats are supported, which currently include *NeXML* [35], *Nexus* [31], *FASTA*, *Phylip* [32] and relaxed *Phylip* [33] for reading and writing, as well as the application-specific formats *MEGA* [69] and *PDE* (used by the alignment editor *PhyDE* [75]) for reading.

Support for additional (custom) file formats can easily be added either by creating new readers and writers for *JPhyloIO* (which allows to plug in custom implementations at runtime) or by implementing I/O classes that directly work together with the alignment model interface of *LibrAlign*. The architecture of both libraries is designed to be easily extensible.

Although an alignment can be read or written using a single method call (see Figure 3.3), *LibrAlign* alternatively allows application developers full access to the *JPhyloIO* event stream and therefore also to any type of (custom) metadata to be used with (custom) data areas.

This architecture provides the connection between metadata attached in files using e.g. externally defined ontologies and (externally implemented) data areas of *LibrAlign*, which can be considered as counterparts. The combination of *LibrAlign* and *JPhyloIO* makes handling of alignment raw- and metadata easy and flexible to be used in different (domain-specific) applications and fully compatible with common metadata models, as used e.g. by *NeXML* [35].

Beyond alignments and their attached metadata, even data not handled by any *LibrAlign* component but by other parts of a dependent application (e.g. phylogenetic trees), can directly be accessed via the *JPhyloIO* event stream in the same read or write operation.

3.3.4 Comparison to other software

Although open-source alignment editors exist in *Java* [78,110–113] and other languages [114–116], and their code could potentially be reused in other software, their editor components are custom-tailored towards the individual applications and not designed to be flexible and compatible with the architecture of other applications or to ensure API stability in future releases. Alignment editors usually neither provide components specifically designed to be customized by other developers, such as the data areas, token painters or model interfaces of *LibrAlign*, nor is the reuse of their code well documented, since that is usually not the intention of their developers.

BioJava legacy 1.9 [94] is a library also providing a set of sequence and alignment renderer classes that can display sequences and render pherograms. Additional custom renderers can be added, which would allow to display custom metadata and this way offer some of the functionality provided by the data areas of *LibrAlign*. Although, alignments and attached data can only be displayed and not edited. I/O functionality from *BioJava* 1.9 for *FASTA*, *Phylip* or *Nexus* can be used to read and write alignment data. In contrast to that, *LibrAlign* offers editing functionality for sequences, alignments and its data areas (such as the pherogram area) adjust the representation of their data to edits of the sequence they are attached to. *LibrAlign* supports I/O of more alignment formats via *JPhyloIO* and access to the nested metadata model of *NeXML* and to annotations of other formats, which is necessary to process metadata handled by (custom) data areas. The alignment renderers in *BioJava* 1.9 make use of their own specific sequence model, while *LibrAlign* has a flexible data model architecture that allows any set of *Java* objects (including those from any version of *BioJava*) to be used as sequence tokens. This makes combining *LibrAlign*'s GUI components with any (custom) data structure (that fulfills applications-specific needs) much easier than it would be in *BioJava*. Furthermore, *BioJava* provides no direct *SWT* support, since only *Swing*-based GUI components are available. (Although using *Swing* components in *SWT* applications is theoretically possible, in practice in can become complex to handle the interaction between the different GUI threads of both toolkits and avoid unexpected behavior or exceptions. Producing stable code using this workaround is much more complex and time consuming than the alternative use of native *SWT* components as provided by *LibrAlign* using *TIC*.)

Alignment renderers have not been ported to the later versions 3 or 4 of *BioJava* [93], while *LibrAlign*'s flexible data model is interoperable with the sequence model of *BioJava* 3 and 4. The sequence token model classes of *BioJava* 3 or 4 (implementations of its interface `Compound`) can be combined with an implementation of *LibrAlign*'s `AlignmentModel` and whole sequences and alignments can be imported from the respective *BioJava* data structures. (*BioJava* sequence instances can though not be efficiently used as the direct storage for editable alignments, since they are immutable and being atomic is a major characteristic of the sequence model of *BioJava* 3 and 4. This is a principle problem that cannot be addressed within *LibrAlign* or any other library providing editable alignment functionality.)

Different web-based applications for viewing MSA data in a web browser [117–120] exist and some provide the option to extend them with custom *JavaScript* components, possibly allowing to display custom metadata. None of these support alignment editing or provide flexible I/O components with full metadata access, the aim of such projects is different from that of *LibrAlign*. These products are used to display MSAs on websites or in web apps, while *LibrAlign* is used by desktop (or *Java Webstart* [121]) applications to display and also to edit MSA data and its attachments.

The *ETE toolkit* [122] is a *Python* library for analysis and visualization of phylogenetic trees, which also allows to display aligned sequences associated with terminal tree nodes. This is a very different focus than in *LibrAlign* and the overlap between both libraries is only very small, since components for alignment editing are not present and displaying alignment metadata is not the focus of *ETE*.

Although *BioJava* 1.9 and few libraries in other languages provide functionality to display MSAs and sometimes also custom metadata, *LibrAlign* is the only available library that combines this with many editing functionalities, a flexible data model architecture and I/O components with full metadata access. Furthermore, the documentation of other libraries is sometimes limited, while *LibrAlign* offers a detailed documentation including a complete *JavaDoc* and code examples.

3.3.5 Current usage

The development of *LibrAlign* was initiated due to a demand for its functionality in several *Java* applications. The *Taxonomic Editor of the European Distributed Institute of Taxonomy (EDIT) Platform* manages taxonomic workflows, while persistently linking character data to preserved individual specimens [24]. It contains an editor to combine single reads from Sanger sequencing to a contig alignment linked to a specimen (chapter 5). It uses the alignment editing functionality of *LibrAlign* and its components to display pherograms.

AlignmentComparator (chapter 9.6.2) visually compares alternative MSAs of the same dataset in a superalignment and uses *LibrAlign* to display and manually edit such a comparison. *LibrAlign* also provides the basis for recent versions of the alignment editor *PhyDE 2* (chapter 6) and is currently used by our group in the development of tools for the evaluation of automated multiple sequence alignments for phylogenetic purposes. (See chapter 13.2.2, page 186.)

Figure 3.3 shows screenshots of the latest development versions of all applications demonstrating different ways of using *LibrAlign*.

3.3.6 Future perspectives

In the future, *LibrAlign* will be maintained and extended according to the needs of the applications that rely on it. This may especially include new data areas of general use, more token painters or additional data model or adapter implementations. Contributions by third-party developers are always welcome. Therefore, we e.g. provide a repository mirror on GitHub (https://github.com/bioinfweb/LibrAlign).

Externally implemented data areas combined with respective metadata attached using externally defined ontologies (e.g. in *NeXML* [35]) could enable *LibrAlign*-based applications to handle any type of raw- or metadata without explicit knowledge on the respective data types. Future versions of the alignment editor *PhyDE* (chapter 6, page 84), *AlignmentComparator* (chapter 7, page 90) or other software could display any unknown type of raw- or metadata by downloading data area implementations that fit the metadata type from an online-database or directly from an URL specified by an annotation in the input alignment file (e.g. in *NeXML*).

3.4 Conclusion

The GUI components provided by *LibrAlign* enable *Java* application developers to easily integrate visualization and editing of all kinds of sequence or alignment data together with attached raw- and metadata into their applications with a minimum amount of work using any of the major *Java* GUI toolkits. The extensibility of the library achieved by allowing custom data areas or token painters and specific model or adapter implementations makes it useful for software developers working in all fields of the life sciences that deal with any type of sequence or alignment data and related metadata.

We hope that *LibrAlign* will help to improve usability of bioinformatical software by easily allowing developers to provide intuitive and user-friendly interfaces. In combination with *JPhyloIO* it should also foster the use machine-readable metadata annotations and externally defined ontologies to model and stored metadata, resulting in an increase of interoperability and reproducibility in respective workflows.

3.5 Availability and requirements

Project name: *LibrAlign*

Project web page: http://bioinfweb.info/LibrAlign/

GitHub Repository: https://github.com/bioinfweb/LibrAlign

ResearchGate project page: http://r.bioinfweb.info/RGLibrAlign

Operating system: Platform independent

Programming language: *Java*

Other requirements: *Java* Runtime Environment 8 (or higher)

License: *GNU Lesser General Public License* Version 3 (*LGPL*)

Any restrictions on use by non-academics: The restrictions specified in the *LGPL* apply. (See http://bioinfweb.info/LibrAlign/License/LGPL.)

3.6 Declarations

3.6.1 Authors' contributions

Ben Stöver developed the idea and conceived the architecture of *LibrAlign*, wrote the manuscript and implemented and tested the software. Kai Müller contributed conceptionally and the manuscript.

3.6.2 Acknowledgements

We would like to thank the developers of the *EDIT platform for Cybertaxonomy* at the *Berlin Botanical Garden and Botanical Museum* for testing the integration of *LibrAlign* into the *Taxonomic Editor*. The work of Phoebe Brech and Jonas Bohn, who contributed to an example application in the documentation of *LibrAlign* in a bachelor and master modules under the supervision of Ben Stöver, is highly appreciated. Thanks to Dietmar Quandt for his input to the concept of the first version of *PhyDE*, which was the basis for some functionality of the components of *LibrAlign*. We thank the contributors to the open source projects used by *LibrAlign* (*Apache commons*, *BioJava*, *OWL API*, *SWT*, *JUnit*, *Hemcrest*). Funded in part by grant MU 2875/3-1 to Kai Müller by the German research foundation (DFG).

Part II –
Applications to
model, visualize, edit and compare
phylogenetic data and metadata

4 Sample data processing in an additive and reproducible taxonomic workflow by using character data persistently linked to preserved individual specimens

Kilian N[1]*, Henning T[1], Plitzner P[1], Müller A[1], Güntsch A[1], Stöver BC[2], Müller KF[2], Berendsohn WG[1], Borsch T[1]

[1]Botanical Garden and Botanical Museum Berlin-Dahlem, Dahlem Center for Plant Sciences, FU Berlin, Königin-Luise-Straße 6-8, 14195 Berlin, Germany

[2]Institute for Evolution and Biodiversity, WWU Münster, Hüfferstraße 1, 48149 Münster, Germany

*Author for correspondence: n.kilian@bgbm.org

This chapter has been published in *Database* 2015, **2015**:bav094
http://dx.doi.org/10.1093/database/bav094

Own contribution

My responsibility in this project was the development of the molecular components of the *Taxonomic Editor* (module `eu.etaxonomy.taxeditor.molecular`), while Patrick Plitzner helped with their integration into the application and coordinated the work between Münster and Berlin. Andreas Müller added required implementations for the *CDM API* to provide interoperability with the new molecular functionality. I implemented the new molecular module, developed the concept for the required libraries *LibrAlign* (chapter 3, page 46) and *JPhyloIO* (chapter 2, page 33) and implemented them, designed the export feature for molecular data of the *Taxonomic Editor* with a draft for a required ontology (see also chapter 5, page 79) and wrote section 4.4.4.3 (page 74) of the manuscript. (The development of the concept for the general project and workflow modeling, the development and implementation of all other modules of the *Taxonomic Editor* and writing all other sections of the manuscript was done by other authors.)

Abstract

We present the model and implementation of a workflow that blazes a trail in systematic biology for the re-usability of character data (data on any kind of characters of pheno- and genotypes of organisms) and their additivity from specimen to taxon level. We take into account that any taxon characterization is based on a limited set of sampled individuals and characters, and that consequently any new individual and any new character may affect the recognition of biological entities and/or the subsequent delimitation and characterization of a taxon. Taxon concepts thus frequently change during the knowledge generation process in systematic biology. Structured character data are therefore not only needed for the knowledge generation process but also for easily adapting characterizations of taxa. We aim to facilitate the construction and reproducibility of taxon characterizations from structured character data of changing sample sets by establishing a stable and unambiguous association between each sampled individual and the data processed from it. Our workflow implementation uses the European Distributed Institute of Taxonomy Platform, a comprehensive taxonomic data management and publication environment to: (i) establish a reproducible connection between sampled individuals and all samples derived from them; (ii) stably link sample-based character data with the metadata of the respective samples; (iii) record and store structured specimen-based character data in formats allowing data exchange; (iv) reversibly assign sample metadata and character datasets to taxa in an editable classification and display them and (v) organize data exchange via standard exchange formats and enable the link between the character datasets and samples in research collec-

tions, ensuring high visibility and instant re-usability of the data. The workflow implemented will contribute to organizing the interface between phylogenetic analysis and revisionary taxonomic or monographic work. **Database URL:** http://campanula.e-taxonomy.net/

4.1 Introduction

Biological systematics, referred to as systematics in this study, aims to assess organismic diversity by attempting to identify natural biological entities above the individual level (taxa), to uncover their relationships and to characterize, classify and name them [123]. All analyses in systematics (Figure 4.1) are based on 'samples', a term used in this study in the unspecified sense of a probe or examination object taken from an individual organism. Examination of these samples produces 'character data' – often named 'descriptive data' [124,125] and sometimes 'comparative data' [123] – a class of data referring to 'taxonomic characters' [126], which each have two or more states and can cover all data suitable to characterize a taxon in comparison with related or similar taxa. Character data that are suitable for use in evolutionary analyses are processed in order to group sampled individuals into natural biological entities. Evolutionary analyses may include to study tokogenetic relationships within a species, or to study sampled individuals as representatives of species in a phylogenetic context. The character data may be analyzed also using a phenetic or other approach. The results in each case are initially unclassified entities, which in subsequent steps can be assigned to taxa and then be named (Figure 4.1) [127,128]. The taxon assignment of unclassified entities revealed from evolutionary analyses translates evolutionary relationships into a classification. This translation essentially employs decisions on appropriate circumscriptions and ranks of taxa, guided by certain sets of criteria, which may be subject to debate. Additional individuals that match these taxa can also be assigned to them. Taxon assignment of individuals, i.e. the process of matching sampled individuals with taxa, thus of their identification, uses a subset of character data as indicators that are considered diagnostic for a taxon and for its distinction from similar or related taxa. The available character data obtained from all sampled individuals of a taxon are finally 'aggregated', thus summed up, into a comprehensive 'taxon characterization' [129] (frequently but less appropriately referred to as 'description', see section two of this study). Taxon characterizations are thus the product of the taxon delimitation [127] and may vary in so far as different taxon delimitations are applied ('taxon concepts') [127,130–132] or different geographic scopes may be considered. The characterizations of higher taxa (taxa that include subordinate taxa) are in the same way the product of taxon delimitation and are the sum of their included subordinate taxa. The taxon characterizations of all subordinate taxa making up a higher taxon are thus to be aggregated into the characterization of the corresponding higher taxon.

The generalized scheme in Figure 4.1 of steps from the investigation of organism individuals to the characterization of taxa also illustrates the interface between evolutionary analysis and taxonomy: both share step 1 (sampling and examination of samples), while step 2 (analysis of relationships) is the core domain of evolutionary analysis, and steps 3 and 4 are the core domains of taxonomy. If the evolutionary analysis in step 2 is replaced by an evaluation of morphological similarities and discontinuities, the result is a so-called 'alpha-taxonomic' classification. This article addresses the taxonomic part of that work process, thus step 1, and, taking up the results of evolutionary or other analyses from step 2, it also addresses steps 3 and 4. We are conscious of the fact that taxon characterizations of microorganisms and fungi may set different accents for the taxonomic work process [133].

Usually, a taxon characterization is based on the examination of a very limited set of individual representatives of the taxon and on a set of character data limited by the selection of examination methods applied. Consequently, any new, sampled individual as well as any new character data may affect the taxon characterization and/or the taxon delimitation. Moreover, in the vast majority of cases the evaluation of the sampled and examined individuals is still based just on morphological similarities and

discontinuities (alpha-taxonomy) and remains to be confirmed by phylogenetic reconstructions. Actually, our understanding of the evolutionary history as well as the classification and naming of taxa necessarily is an iterative process, with an approximation to reality, often triggered by methodological innovations. The need for minor or major revisions or adjustments of established classifications and taxon characterizations, also affecting their names, is thus both pervasive and continuous [134–136].

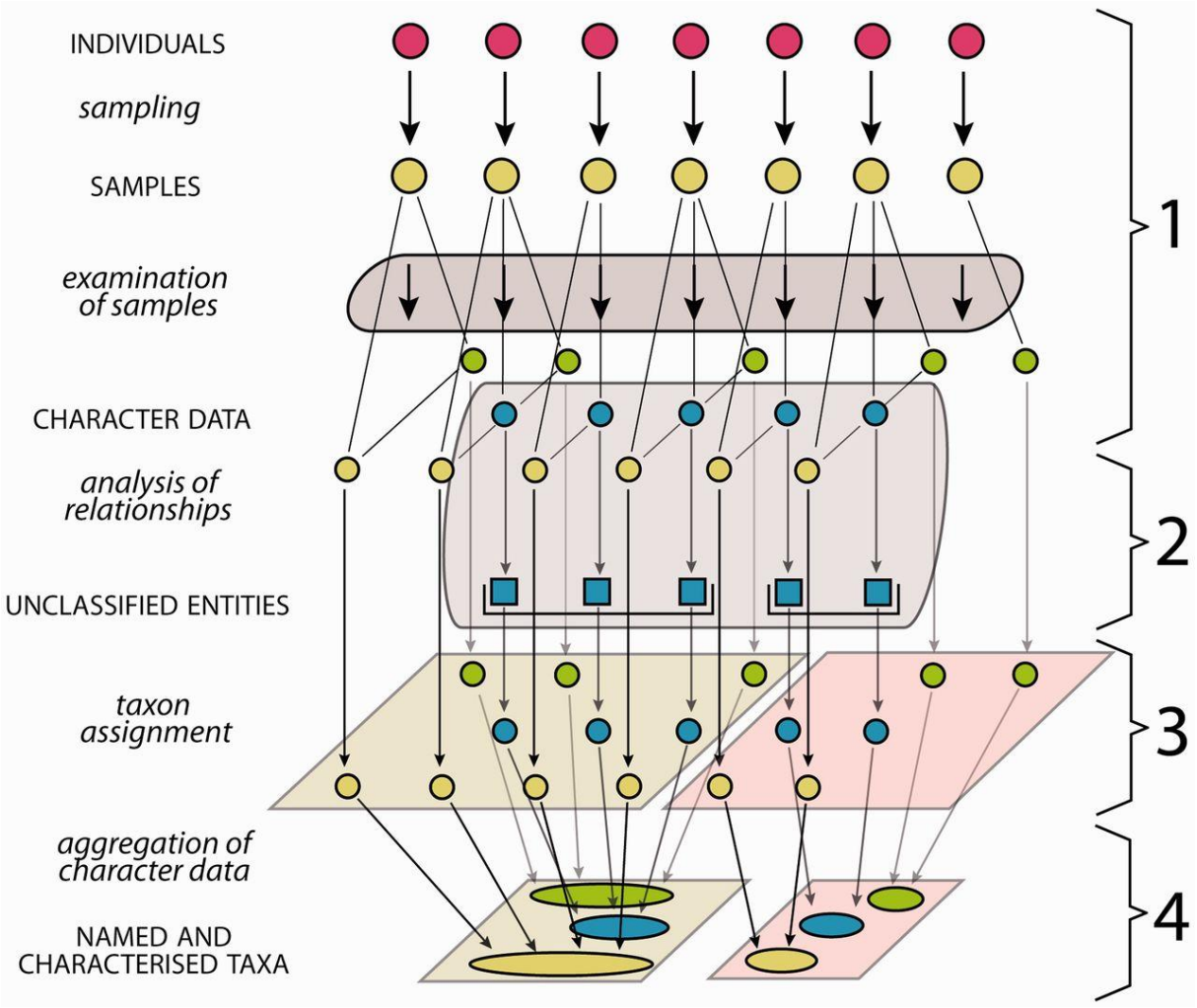


Figure 4.1 Generalized scheme of the steps in systematics from the investigation of organism individuals to the characterization of taxa

The first column lists the processes (lower case letters + italics) and products (upper case letters + normal style), the diagram illustrates the data flow and the last column numbers the steps as explained in the following: (1) samples of individuals are examined, providing different types of character data (green, blue, yellow), not all of them necessarily available for all samples. (2) Analysis of relationships (e.g. phylogenetic or tokogenetic), using e.g. available molecular character data (blue), reveals evolutionary relationships among the sampled individuals, grouping them into unclassified entities such as clades. In a phenetic approach, the evolutionary analysis in this step is replaced by an evaluation of morphological similarities and discontinuities. (3) In order to translate inferred (from whatever analysis) relationships into classification, the unclassified entities with the included samples and character data are assigned to taxa, also employing further character data types (yellow, green). (4) Naming of taxa and aggregating (summing up) of the character data from the individuals included results in named and characterized taxa. Further sampled individuals not included in the evolutionary analysis but matching the taxon characterization can be included, their data adding to the characterization.

The process of synthesizing our growing knowledge of biodiversity is challenging. Integrative taxonomic treatments in general, and monographs as a final product of systematics [137,138] in particular, consequently represent the approximate knowledge at a given point of time. Societal demands for

reliable, up-to-date, and authoritative products, such as biodiversity inventories, identification aids and encyclopaedic works on groups of organisms [139], call for name stability, while progress in systematics may affect established classifications and names.

One of the major problems involved is that print publications are too static to function as knowledge bases of organismic diversity. For this, biodiversity informatics has developed solutions to design synthesizing works in biodiversity research as dynamic ventures [102,140–143] and to facilitate data exchange by providing unified and convenient query mechanisms for distributed and often highly heterogeneous data repositories.

However, in order to organize dynamic approaches to such syntheses, they need to be generated from data that are structured in a standardized form and stored in an underlying database. Character data structured in character and state matrices [125,126,144] and data aggregation procedures for taxon-based character data are well established, although still applied by a limited number of workers. Several applications are available for storing structured taxon-based character data in order to generate identification keys and natural language descriptions, and to aggregate them from lower to higher taxa. Starting with the *DELTA* (*DEscription Language for Taxonomy*) [145] system [125] as the pioneer, others followed such as *Lucid* [146], *Delta Access* [147] and *Xper*^{2} [148]. With the development of the *XML*-based *SDD* (*Structure of Descriptive Data*) standard [149], data in the *DELTA* and *Nexus* [31] standards [150] are becoming fully exchangeable, *SDD* compliance provided. With the *NeXML* exchange standard [35], recently an *XML*-based *Nexus* successor for representing taxa, phylogenetic trees, character matrices and associated metadata has been developed.

The implicit conclusions for the association between character data from sampled individuals and taxon characterization have, however, hitherto hardly been drawn with the necessary rigor. The Prometheus Model [125,127,151], an approach based on taxonomic working practices rather than on taxonomic outputs, is a remarkable exception. In order to make an investigation both transparent and reproducible, vouchers (commonly termed specimens) allowing an assured identification of the sampled individuals are permanently preserved. Consequently, the Prometheus Model emphasizes that the research process in systematic biology at the species level and below is specimen based, and the taxon characterization is the product of the included specimens, and the taxa above the species level are circumscribed by the subordinate taxa. The taxon characterization can thus only be determined and reproduced in an objective way by the included specimens. The Prometheus Model takes a specimen-oriented rather than a taxon-oriented approach. With the Prometheus Description Model, Pullan et al. [125] moreover perspicaciously addressed the need for the re-use and exchange of character data between different research projects, and modelled pioneering solutions for the main problems involved. This includes a solution for compatibility issues of character datasets from different sources and also the possibility of recording character data at various levels of concreteness, ranging from a single instance of a structure on a specimen to the individual specimen as such. Yet, the Prometheus Model was never developed to a tool available for taxonomic work.

Therefore, until today common taxonomic working practice is that the characterization of a taxon refers only collectively to a set of included specimens so that the character data are not associated with the individual specimens they were taken from. In this way, the only accurate way of achieving adjustments with respect to taxon delimitation and consequently to taxon characterization is the most laborious: re-examining the characters and specimens.

For a sound foundation of the character data aggregation procedure and in order to streamline taxon characterization, a reversible generation of a taxon characterization from the character data of the sampled individuals is necessary. The prerequisite for this foundation is to establish a persistent and unambiguous connection between each sampled individual and the data processed from it. Specimens

remain the representatives of the sampled individuals after the conclusion of the systematic research process and are preserved and curated in corresponding research collections. The obvious conclusion should therefore be the establishment of an unambiguous association between the character states and ranges recorded for each specimen, or for each sample substantiated by a specimen, and their persistent connection with the specimen metadata. Any newly examined individual assigned to a certain taxon may then confirm or modify the taxon characterization upon re-aggregation of the character data. Once evolutionary analysis of character data reveals changes in taxon delimitations, its characterization can then be regenerated upon aggregation of the character data from the altered sample sets. The necessity to document the character data for the individual specimens rather than for taxa similarly applies to phylogenetic analyses, in particular for such based on morphological characters, where the corresponding problems have been clearly addressed [152].

This article presents the concept of a workflow and dataflow that blazes a trail in systematic biology for the re-usability of character data and their additivity from specimen to taxon level, and its implementation, using the *EDIT* (*European Distributed Institute of Taxonomy*) *Platform* [102]. We first (part 2) explain our concept for the implementation of a persistent and unambiguous connection between character data and samples in the systematic research process. Subsequently (parts 3 and 4), we describe the implementation of the single steps of the workflow using the *EDIT Platform*.

Our solution aims to (i) establish a reproducible connection between sampled individuals and all types of samples derived from them during the research process; (ii) persistently link the metadata of all types of samples with the respective character data; (iii) record and store specimen-based phenotypic, geographic and environmental as well as molecular character data in formats suitable for data exchange; (iv) reversibly assign sample metadata and character datasets to taxa in an editable classification and display them and (v) organize the exchange of sample data sets via standard exchange formats. Finally, we discuss the opportunities that our solution opens up for the preservation of raw data and for the deposition of character datasets along with samples in research collections, and we identify fields where further developmental work is needed.

4.2 Conceptual foundations of integrated sample data processing

4.2.1 Organismic samples, their associations and data

In systematics, the analyzed samples each directly or indirectly originate from a population of organisms in the field. Collecting samples of such a population creates a ‘gathering’ [the term is here used in the sense of the ‘International Code of Nomenclature for algae, fungi and plants’ (ICN)] ([153], p. 156) for ‘a collection of one or more specimens made by the same collector(s) at one place and time’. The ‘gathering event’ [154] is thus connected to a specific time and location. The single gathering, to which usually a unique ‘field number’ or ‘collecting number’ is assigned, is a data object termed ‘field unit’ [154]. We here use this term to refer to a single (named or unnamed) taxon, and either to a single individual, of which it may include one or more samples (depending on the size of the individual), or to a population, of which it consequently includes a number of individuals or parts of them. Therefore, the field unit can consist of one ‘specimen’ or a number of ‘specimens’ and in the latter case they are commonly considered duplicates of one another and are thus principally exchangeable with respect to their essential information content. This depends, of course, on the research context: population genetic analyses, e.g. require that duplicates must stem from the same individual. However, the concept of the ‘field unit’ also allows the handling of multitaxa gatherings; as taxon-ambiguous field units are permitted, it is, therefore, also applicable to the study of microorganisms.

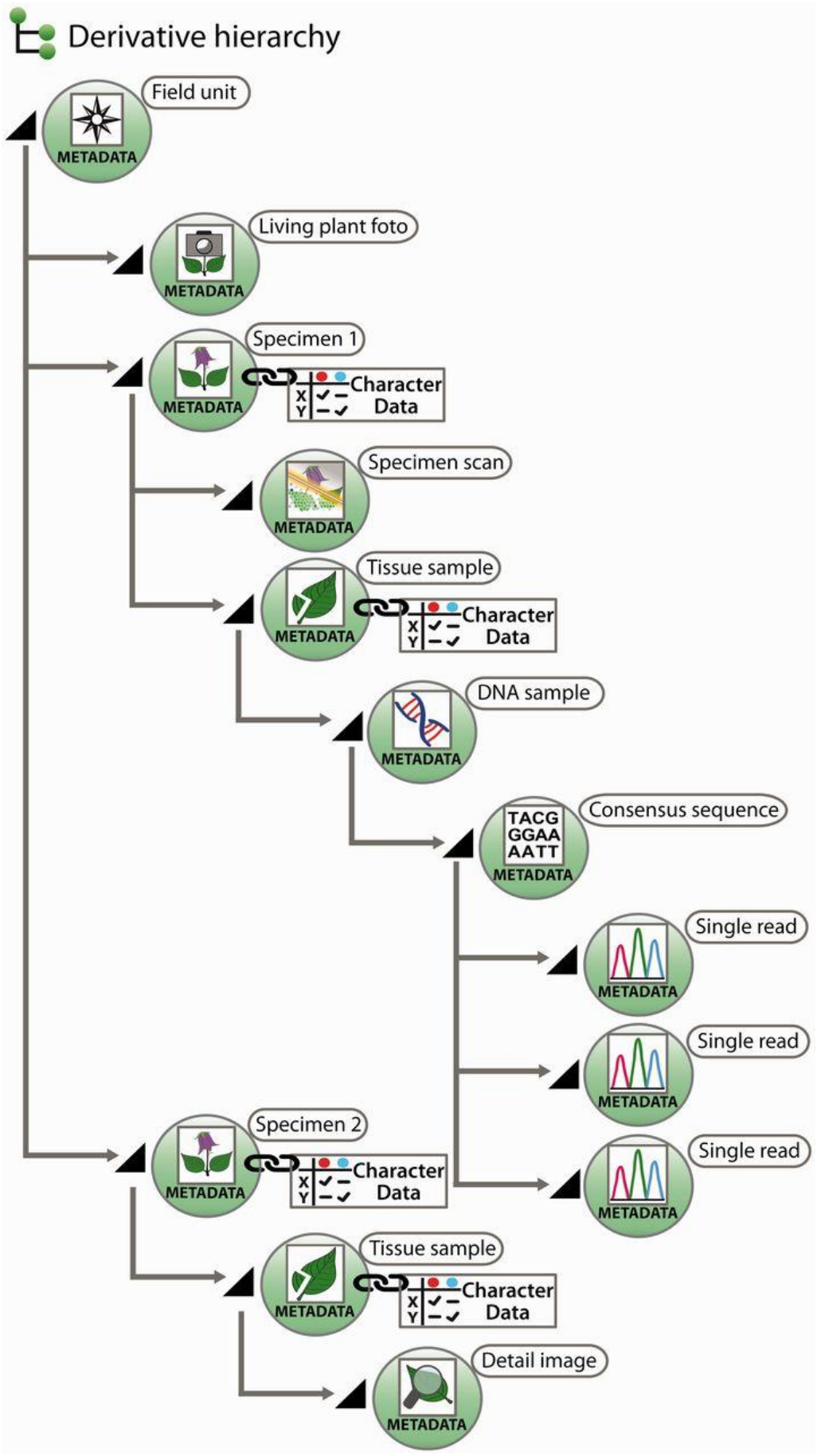


Figure 4.2 Exemplar scheme of samples with metadata and character data in a derivative hierarchy

All further samples taken from a specimen of a field unit are termed 'derivatives', more precisely 'specimen derivatives' ('derived units') [154]. Based on the field unit, derivation events can create a series of derivatives. Being products of derivation events, derivatives are usually hierarchically structured (e.g. specimen → pollen sample → scanning electron microscope (SEM) micrographs). Both the derivative hierarchy and any single derivative are rooted to the field unit, ensuring that each derivative is rooted even if an intermediate derivative is lost, of ephemeral nature or has never been recorded. A first derivation step from a taxon-specific field unit is the individualization of specimens, the specimen thus constitutes a first derivative of the field unit (Figure 4.2).

The taxon assignment of a taxon-specific field unit is normally inherited (in terms of data processing) to and valid for all the field unit's derivatives. Similarly, a taxon assignment to a derivative is inherited to all its other elements and the field unit. Erroneous assignments of samples to a taxon-specific field unit may result from misidentification in the field or in light of novel insights following later analyses leading to the re-circumscription of a taxon. Another possibility is the consideration of a taxon which was outside the scope of the original gathering (e.g. epiphytic lichens, parasites) [155]. Consequently, such samples need to be separated (at least in their virtual representation) and assigned to the correct newly developed taxon-specific field units.

Once a derivative becomes part of a collection (e.g. a herbarium), and thus a collection object, a metadata type termed 'collection unit' can be assigned to the derivative.

Any sample that is examined, regardless of whether it is newly collected in the course of the research process or taken from a research collection, which, in the latter case, may be from a living collection (e.g. botanical or zoological garden) or museum collection, is assigned to a specimen derivative hierarchical level. Two types of data are principally associated with each sample:

i. 'Sample metadata' predominantly include the event-related information, including sample origin, collecting locality, observations in the field, gathering method, preparation process, derivation events in the examination, position in the derivative hierarchy, accession and storage place in a collection and more. The main functions of the sample metadata are to give the sample a unit identity and to make it reproducible or at least traceable. The core of the sample metadata is found on labels attached to a collection object, which may be supplemented, in the case of poorly labelled 'historical' specimens, by data from related sources, such as published reports on expeditions and laboratory protocols. The 'taxon assignment data' are a particular type of sample metadata, which indicate the taxonomic identification of a (taxon-specific) field unit and all its derivatives, including the taxon name, typification, name of identifying scientist, date of determination, synonymizations and determination history. The taxon name connects the sample and its data to a certain taxon in the classification. One type of sample metadata has a double nature: data related to the gathering event in the field, such as locality data, gathering date and observations on the gathered organism, will also contribute to the characterization of that taxon (described in detail below), by information such as distribution, ecology or phenology.
ii. 'Character data' include all primary (raw) and secondary (edited or derived) data gained through the examination of a sample. They can theoretically comprise the entire phenome (the entirety of a taxon's 'traits' or 'features'), genome information plus all related geographical and environmental data. If character data have an unambiguous connection to a single documented sample, they are referred to as specimen-based as opposed to merely taxon-based character data. 'Structured character data' are organized in a matrix distinguishing characters and two or more states, in contrast to 'textual character data' (e.g. in a natural language description).

The term ‘trait’ is conceptually narrower than ‘character data’. Trait refers to phenotypic variation relative to genetic and environmental factors for particular phenotypes. However, it has been used ambiguously either corresponding with a character or, more commonly, with a state. The definition of the term trait has been widened in ecology to functional and physiological traits. The term character data is inclusive of these as well. The terms ‘descriptive data’ [125,148] and ‘comparative data’ [123] are largely synonymous to character data. However, the former in particular has often been used in the narrow sense, referring to the data of the ‘taxon description’, which historically ranges from a brief morphological differential diagnosis to a more or less comprehensive morphological description of a taxon. The term ‘factual data’ [156], coined in the context of modelling data relations of taxon concepts and names, is wider than the above mentioned terms. It refers to any factual information that is connected to a taxon and thus also includes information about human uses or the conservation status of a taxon, which is too extensive to be included with the character data.

Derivation events frequently lead to samples that are either not preserved as physical objects, or they lose their physical concreteness and then are merely present as digital objects. Examples include the SEM analysis of pollen samples, where only the digital SEM micrographs remain, or the amplification and sequencing of markers from a DNA isolate, where after an isolate has been used only the trace files remain. Where derivation events transform physical into digital objects, the digital objects can, with similar justification, be treated as sample derivatives or as data gained from samples. Generalizing this, one could consider the generation of character data from a sample as a derivation event, and the obtained character dataset as a further derivative instead of a sample-based characterization item. We have decided, however, to treat in the data model only derivatives in the narrower sense, i.e. not character data as derivatives, but in the interest of user convenience, a joint visualization of derivatives and character datasets in the user interface independent of the model decision is possible (Figure 4.2, and see below).

4.2.2 Processing sample metadata

Usually, samples in a research project in systematic biology are to some part newly collected, while to some other part obtained from research collections, either as physical objects or as digital representations. A required functionality is therefore the communication with research collection databases or corresponding aggregators to search for and to import digital sample representations and sample metadatasets. The standard exchange formats *ABCD* (*Access to Biological Collection Data*) [157] and *Darwin Core* [158] should be supported.

Imported metadatasets may need, at some stage, to be edited. Editing may include the following: (i) completion of label data; (ii) addition of relevant metadata from other sources, such as duplicate samples and itineraries, for insufficiently labelled historical collection items; (iii) standardizations, such as making collector names unambiguous and conversion of data into standard units; (iv) completion or correction of the parsing of the metadata into the relevant data fields; (v) clarification of toponyms and georeferencing localities and (vi) fixed associations of taxon names with specimens following nomenclatural typification. Editing with respect to (iii), (iv) and (v) is essential for the processing of metadata elements in the context of taxon characterization, such as georeferenced localities for distribution mapping or collecting dates for phenology. Type information (vi) is to be processed in order to fix the application of a name to the taxon containing this specimen.

In the case of collection items or their derivatives for which no digital metadatasets are available, these need to be newly entered into the data store. In the case of newly collected material for an investigation, it depends on institutional workflows; the material may be first accessioned by the research collection and its metadata can then be imported from the institutional collection database, or vice versa. Exporting the newly entered and the edited sample metadatasets to institutional research collections

is possible using standard exchange formats ABCD [157] and Darwin Core [158]. Furthermore, this can be done in a way that clearly distinguishes original and edited data.

4.2.3 Linking specimen-based character data to sample metadatasets

Sample examination produces specimen-based character datasets of various types and formats. These datasets are characterization items to be persistently linked to the analysed samples (represented by their metadataset) and via the derivative hierarchy also to the individual specimens documenting the individual organismic source of these data. For all stages of the research process the corresponding character datasets should be available, visible and easily accessible. An export of the sample metadatasets to the respective research collection should contain a stable link to the existing character datasets, or even be directly associated with the available character datasets.

4.2.4 Taxon assignment of samples and their data

Through assignment to a taxon, the field unit as the root of the specimen derivative hierarchy becomes connected to the taxonomic classification of a group of organisms. As a consequence, all connected derivatives, the sample metadata corresponding to the gathering and the character data resulting from the examination of a sample also become assigned to that taxon. The taxon assignment is thus effective for all levels in the derivative hierarchy and is reversible. Samples and character datasets assigned to taxa should be easily visible and accessible.

Simple moving of a taxon within a classification or renaming does not affect the connection between samples and taxa. In contrast, re-delimitation of a taxon, which involves a re-evaluation of the included samples and/or character data, will also demand to adjust the taxon assignment of the samples.

4.2.5 Aggregating specimen-based character data at the taxon level

The essential procedure for any taxon characterization is the aggregation of the specimen-based character data to taxon character data according to the delimitation of the taxon. The extent and type of the aggregation depends on the data type and structure, and the means and purposes of their use at the taxon level. This may include an ‘appending aggregation’ (leaving the appended data unchanged), such as DNA sequence data, or a ‘merging aggregation’ (statistical values), such as the measurement of floral features or altitudinal distribution ranges.

It is necessary for data aggregation to be designed as an iterative and automated procedure, permitting changes in the sample basis of the data, due to changing taxon delimitation or data availability. This would trigger a new round of aggregation, which replaces the results of the preceding one. The prerequisites are that the data are structured and compatible. Taking the domain of morphological data as an example, it becomes evident that the main obstacle is to ensure that sets of characters and states are compatible during specimen investigation across a larger group of organisms. Aggregation for distant taxa of the same larger group of specimen-based data at the lowest taxon rank applied must not use incompatible matrices in order for subsequent aggregations at higher ranks to be successful.

A number of applications exist to create taxon-based character and state matrices and to further process them for the generation of identification keys and natural language descriptions, and to aggregate them from lower to higher taxa [125,146–148]. As long as compliance with the XML-based SDD standard [149] is provided, the data are exchangeable between the applications. Problems regarding exchangeability of structured data matrices, term ontologies including addressing homology issues and the character data model [125] remain to be addressed in future work.

Fortunately, the aggregation of character data from lower to higher taxa is principally the same as the primary aggregation of specimen-based character data at the taxon level, with respect to data structure and aggregation algorithms. The same applications can thus be employed in order to record and aggregate specimen-based character data.

4.3 Workflow implementation using the EDIT Platform

4.3.1 Extending the EDIT Platform to handle the variety of sample data

Our concept for an integrated workflow for sample data spans from the selection of sampled individuals to the aggregation of character data for named taxa (Figure 4.1), but intentionally it excludes the capacity to conduct evolutionary analysis of sampled individuals. However, it aims to include the entire data recording for the examined samples (metadata and character data) and to hold and provide the specimen-based structured character data (morphological and molecular) of the sampled individuals for any evolutionary analysis, such as phylogenetic reconstruction. The datasets for the sampled individuals can be assigned to taxa according to the results of the analysis and the character data can be aggregated to add to the taxon characterization.

The implementation of this workflow requires a web-enabled working platform, readily allowing networking of distributed team workers, capable of the pertinent data exchange standards for collection data, with suitable interfaces to handle character data, and capable to handle taxonomic classifications. Therefore, the *EDIT Platform for Cybertaxonomy* [102,159,160], or shorter, *EDIT Platform* has been selected for development of our workflow model. The *EDIT Platform* provides the necessary basic functionalities which require minimal extensions, especially in the specimen module. The EDIT Platform is based on the *Common Data Model* (*CDM*) [161], which is a comprehensive object-oriented taxonomic information model covering the flow of taxonomic information from fieldwork to data publication. The pivot of this model is the 'taxonomic concept' (or 'potential taxon') being strictly separated from scientific names. This approach was originally developed by Berendsohn [162] and later refined and implemented in the *Berlin Model* e-Platform [163,164]. Added to this was a rule-based 'transmission engine' for the transfer of character and other taxon-related 'factual data' between concepts in a network of taxonomic concepts [165,166]. The *CDM* complies to the relevant data standards of biodiversity informatics (*Biodiversity Information Standards* [*TDWG*], also known as *Taxonomic Databases Working Group*) [167], including *ABCD* [157], *Taxon Concept Schema* [168], *SDD* [149] and *Darwin Core* [158]. Besides the *EDIT Platform* it is also the basis for *Creating a Taxonomic E-Science* [141].

An outstanding feature of the *EDIT Platform* is its connectivity and interoperability among the emerging international biodiversity informatics infrastructures through standardized web service layers. Data exchange interfaces to various biodiversity e-infrastructures have been implemented including the *GBIF* (*Global Biodiversity Information Facility*) *Checklist Bank* [169], *Biowikifarm* [170], *Scratchpads* [171], *Plazi* [172], *BioVeL* [173] and *Biodiversity Heritage Library* [174].

The *EDIT Platform* is open source and applicable to all groups of organism, in particular those covered by the *ICN* [153]and the *International Code of Zoological Nomenclature* [175]. Current applications are monographic in approach (*Cichorieae Portal*; *CLD-CoW Portal*; *Palmweb*) [176–178], regional checklists [179] or floras [180].

4.3.2 Basic functionalities of the EDIT Platform, scalability and use cases

The *EDIT Platform* can be employed to handle and connect the different data types associated with the samples right from the start of a research process in systematic biology. It provides three main components:

i. Data repository and server: The *CDM store* hosts the taxonomic classification, the metadata and character data for samples, and also links to external web resources. All data objects are accessible through Java and web service interfaces.
ii. *Taxonomic Editor*: The core application of the working platform functionality is the *Taxonomic Editor*. Among others, it allows the searching for, importing, entering and editing of all taxon- and specimen-related information stored in the *CDM*.
iii. Data Portal: The portal provides a dynamic visual user interface for online publication. It gives access to all publication-relevant data objects stored in the *CDM*. Classifications are represented by a taxon tree, which allows users to navigate through multiple hierarchies. The portal links out to biodiversity e-infrastructures such as *BHL* (*Biodiversity Heritage Library*) [174] and *GBIF* [169] and has advanced functions for visualizing species distributions and multimedia files.

Although the *EDIT Platform* is being designed to support the distributed research process in systematic biology from sample acquirement to the publication of a monograph, more frequent use cases are taxonomic revisions or phylogenetic analysis of smaller groups of organisms, and in some cases a combination of both. Such work is frequently conducted by an individual scientist or a working group and usually without a long-term dedication to a particular group of organisms. Cases like these often lack the active institutional support, in particular IT infrastructure. Instead of the fully operational 'community installation', they may use the easy to install 'individual installation', which allows a single worker to edit and maintain an individual dataset on a personal desktop, and a 'group installation' for a working group with a shared data repository within an institutional intranet. In contrast to the individual installation scheme, the group installation comes with a data portal to publish the data electronically (http://cybertaxonomy.eu/cdm-setups/). An installation with the full implementation of the workflow described in this article is expected to be available for download by the end of the project in December 2015.

4.4 Steps of the integrated sample data workflow

4.4.1 Scope of the workflow

Here we outline the steps of the integrative processing of sample metadata, sample character data and their taxon assignment, as it has been developed and is being implemented in the *EDIT Platform*. Its aim is to create the prerequisites for a consistently specimen-based research process in systematic biology. This includes the following: (i) establishing a reproducible connection between sampled individuals and all types of samples derived from them, which allows instant sample metadata processing, including de novo input, retrieval, import, documented (for potential synchronization with external sources) editing, display and export, within the research process; (ii) stably linking the metadata of all sample types with the respective character data gathered from them, by providing means for handling specimen-based character data (morphological and molecular) and for firmly linking them to the sampled individual; (iii) recording structured specimen-based character data in formats allowing data exchange and easy retrieval; (iv) reversibly assign sample metadata and character datasets to taxa in an editable classification, allowing optional publication of the investigated samples in the context of taxon-based information portals and (v) organizing data exchange via standard exchange formats and enabling persistent, specimen-linked storage effectively accessible for humans and machines in research collections, ensuring high visibility and instant re-usability of the data.

The workflow described is still a work-in-progress. Although the foundations were laid for the implementation of the entire workflow in the *EDIT Platform*, its single steps have been elaborated so far to different depths. It will be workable throughout by the end of 2015 but in particular the handling of structured (morphological) character data will have to be considerably improved to meet all essential needs by a corresponding follow-up project proposal submitted to the *German Research Foundation*.

4.4.2 Establishing a reproducible connection between sampled individuals and all types of samples derived from them

4.4.2.1 Searching, retrieving and importing of sample metadata

The 'specimen search' in the *Taxonomic Editor* is defined by the search parameters and the query interface supported by a specimen data provider. The implemented system supports a list of specimen data providers and allows users to decide which provider to query. It converts the query to the required format for the provider's interface. One option currently implemented is *GBIF* [169], which is queried via web services; the other option is *BioCASe* [181], the providers of which are queried with a specific *XML*-based query protocol [155] (Figure 4.3). Common search parameters are taxon name, collector, collector's number and country. Specifying two or three of these is usually sufficient to reduce the search results.

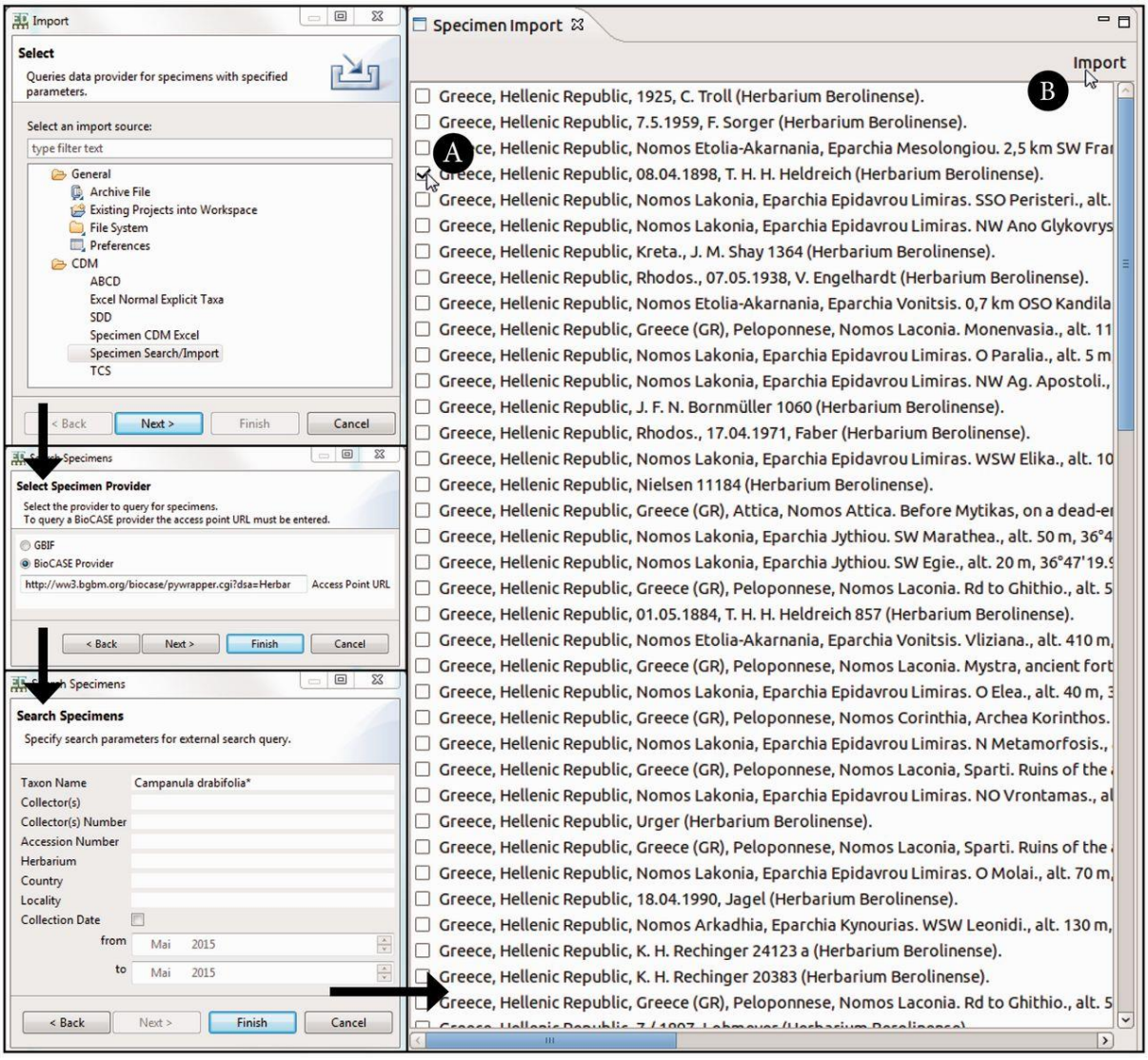


Figure 4.3* Taxonomic Editor *of the* EDIT Platform*, derivatives perspective: screenshot of the specimen query and import interface

The black arrows indicate the single menu steps that specify the import. After the import form has been sent out, the search results are listed in a separate tab. The specimen can then be chosen (A) and the import of the datasets can be completed (B).

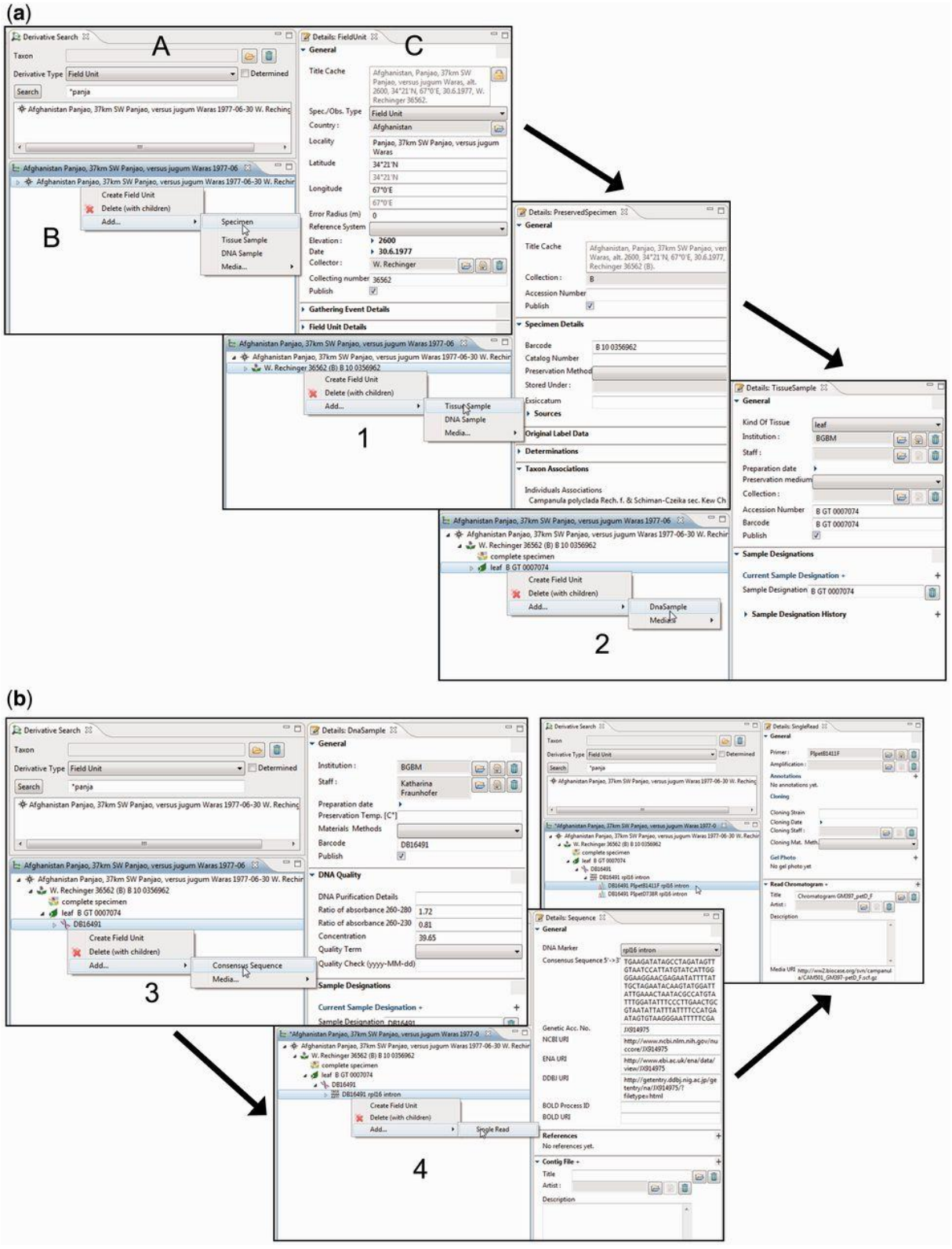


Figure 4.4* Taxonomic Editor *of the* EDIT Platform*, derivatives perspective

Screenshot of the derivative view displaying the derivative search (A), the derivative hierarchy (B) and a details view for the corresponding metadata (C). Screenshots illustrate the stepwise establishment of a derivative hierarchy by successive creation of derivatives and insertion of their data: (a) addition (1) of a tissue sample and (2) of a DNA sample; (b) addition (3) of a consensus sequence with links to one of the INSDC *(*International Nucleotide Sequence Database Collaboration*) databases, (4) of single reads (Sanger sequencing trace files) and/or a contig file.*

The *Taxonomic Editor* provides an import routine that can both convert the different formats returned (*ABCD* and *Darwin Core*) to display the results in a *CDM*-unique, standardized format and provide the functionality to store the specimen data in the *CDM*, merging it with existing data. The imported data are stored with the provider's original unique identifiers to enable data synchronization.

4.4.2.2 Editing metadatasets

The specimen module of the *Taxonomic Editor* has been extended to provide full user interface functionality for displaying and editing all levels of the derivative hierarchy. The tissue and molecular sample modules of the *CDM* have been extended to enable full data coverage. Fields with pre-defined or user-defined elements have been selected to avoid redundancy and ensure coherent use of terms and names, e.g. for primers and DNA markers.

4.4.2.3 Building and editing specimen derivative hierarchies

The derivative hierarchy is displayed as a tree in a separate interface, the 'derivative search view' (Figure 4.4 A). 'Derivative view' (Figure 4.4 B) and 'details view' (Figure 4.4 C) form a functional unit that allows the convenient access to, and the creation and processing of, derivatives and their data. The field unit element is obligatory because it is the root of the derivative hierarchy and appears, if not manually created, automatically once a specimen or any other sample is entered or imported. All subsequent derivation steps and derivative types are prearranged in a hierarchical order according to the typical research workflow.

According to our concept of the derivative hierarchy, the derivative view holds a central position in the specimen module of the *Taxonomic Editor*. It is used to build the derivative hierarchy, thus to select derivatives, to visualize associated character datasets of the respective samples, to add and edit sample metadata and to display the hierarchy with all its data types in the *Taxonomic Editor* as they may also appear in the Data Portal. An example of such a prearranged derivative hierarchy (Figure 4.2) in the *Taxonomic Editor* is as follows: field unit → specimen collected → tissue sample taken → DNA isolated → DNA trace file created by the sequencer → consensus sequence generated from the contigs (Figure 4.4). In all such cases, the full sequence of the derivatives is not mandatory and can be applied as appropriate. For example, if no tissue sample and DNA isolate are stored, the trace file or consensus sequence can directly be attached to the specimen.

By selecting the details view, input options are provided for the essential metadata of each derivative. In the case of molecular data, the necessary terms and input options are matched with those compiled for the *GGBN* (*Global Genome Biodiversity Network*) network [182]. The full extent of data covered by other repositories can be accessed via links in the details views (Figure 4.4 B).

4.4.2.4 Versioning, synchronizing and exchanging metadatasets

The editing process (adding, deleting or changing data) will enrich and refine the original metadataset. These changes are separated from the original dataset, resulting in two semantic parts of sample metadata: (i) the 'core copy', the original dataset from an external provider and (ii) 'enrichment and refinement', the edited data which can be subject to a manual versioning by employing the auditing functionality of the *EDIT Platform* (Figure 4.5). On this basis, a special 'Diff-Viewer' can be implemented in the future to visualize the differences between versions and additionally allow the user to revert changes to an older version (Figure 4.5). Edited and newly entered specimen metadatasets can be provided to the corresponding research collections using *AnnoSys* [183] as a back-end service for storing and communicating the annotations. *AnnoSys* provides the functionality to annotate publicly displayed specimen records by users, to keep track of, and to inform data providers about annotations. *AnnoSys* exposes the annotations in the *ABCD standard exchange format*. The exposed dataset will include the documentation of the editing of the core copy to give the providers the opportunity to

update their data. Conversely, the researchers can ask providers for a possible update of an earlier imported core copy and manually update their local copy.

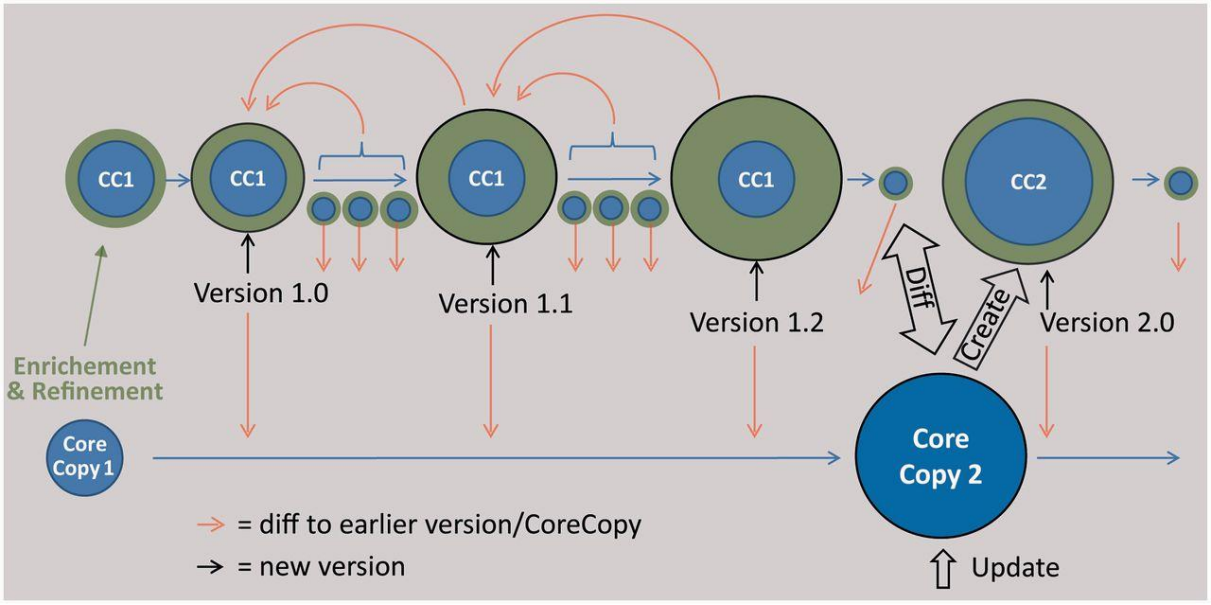


Figure 4.5 Scheme of the envisaged versioning functionality for sample metadata

The core copy is a copy of an imported dataset of an external provider, which is edited (green data). The versioning support of the CDM *database, reporting every single change in the data, is used at certain intervals to create versions of the data, which can be compared using a diff viewer. The result of a subsequent query at the provider is stored as a new core copy, which can be compared with the latest version based on the first core copy and subsequently be edited.*

4.4.3 Stably linking character datasets to the sample derivative hierarchy

The central interface for linking specimen-based character datasets to the sample derivative hierarchy is the 'factual data view' of the *Taxonomic Editor*. The addition of such character data is displayed in the derivative hierarchy, where the derivative symbol is then replaced by a 'derivative + character data' symbol (Figure 4.2). Storage of the sample derivative hierarchy data in the *CDM* is configured to include the information about and, optionally, a stable link to external character datasets, or the stored character datasets themselves.

4.4.4 Recording and storing specimen-based morphological and molecular character data

4.4.4.1 Storage

Specimen-based character data can be stored and curated in the *EDIT Platform* using the *Taxonomic Editor*, independent of their format. Data available in files from external applications can be stored and linked via the Web. For storing data files of various types in a working environment, we are using a server with *Apache Subversion* (*svn*, https://subversion.apache.org/), which combines convenient accessing of the file repository (e.g. using *TortoiseSVN*) with the advantages of a versioning and revision control system. The files are publicly available via a URI (uniform resource identifier). Mere textual data can be stored in free text fields of the *CDM data store*, some types of structured data can be directly mapped to the corresponding *CDM* classes for structured factual data. A fully functional data management, however, requires structured data in the supported exchange formats (see below).

For recording and editing character datasets, the *Taxonomic Editor* provides the 'Factual Data View' and specialized views for different data types, in which seamless integrations of otherwise independent applications are operational.

4.4.4.2 Structured morphological character data

For the recording and processing of structured morphological and related types of character data, the *Xper*^{2} software [148] is used. This software enables free creation of matrices of characters and character states and the recording of qualitative and quantitative character data of specimens and derivatives. In a recent paper ([130], p. 295–296) we have outlined an approach, employing a terminology server and semantic web technology to ensure the compatibility of characters and states taken across a larger group of organisms, which we identified as a main challenge in part 3, above. There, we have also proposed a strategy as to how the wealth of unstructured textual descriptions in the literature can, in a controlled way, be employed in the frame of an otherwise specimen-based approach relying on structured data for taxon characterizations at lower taxonomic rank. Implementation of these approaches is subject to a corresponding follow-up project proposed.

4.4.4.3 Molecular character data

For recording and processing molecular (DNA) data, the *Taxonomic Editor* has been extended using several GUI (graphical user interface) components that display pherograms (trace files from Sanger sequencing) imported from *AB1* or *SCF* files with their base call sequences and allows the combination of these in contig alignments and the creation of consensus sequences. The user can easily manually correct the base calls or edit the contig alignment and the consensus sequences. To achieve this, a new open source *Java* library called *LibrAlign* (chapter 3, page46) has been developed. It provides powerful and flexible GUI components for displaying and editing raw data and metadata for sequences and alignments. Although *LibrAlign* was mainly developed for use in the *Taxonomic Editor*, its components have been designed to be of general use for other developers in the scientific community and it may be integrated into any *Java* GUI application, based on *Swing*, *SWT*, *Eclipse RCP* and *Bioclipse* [108], and it is interoperable with the *CDM Library* [161] and *BioJava API* (application programming interface) [93]. Furthermore, support for importing and exporting whole contig alignments in various formats, such as *FASTA*, *Nexus* [31], *MEGA* [184] or *NeXML* [35], is currently implemented using *JPhyloIO* (chapter 2, page 33) in combination with *LibrAlign*.[a] *JPhyloIO* is another general purpose *Java* Library developed for the *Taxonomic Editor* that provides event-based format-independent access to different sequence and alignment file formats. It is closely integrated with *LibrAlign*, but can also be used in the development of any application that does not use *LibrAlign*.

4.4.5 Taxon assignment of sample metadata and character datasets

4.4.5.1 Adding sample data to a classification

The classification used for an investigated group of organisms can be displayed and edited in the 'taxonomic perspective' of the *Taxonomic Editor*, a pre-defined and pre-ordered set of graphical interfaces. This enables taxonomic hierarchies with synonymies to be imported, created and edited, including complex re-classification operations. The taxon assignment of a specimen or derivative hierarchy is effected in the details views of the derivative view or in the factual data view, where a taxon of the stored classification can be selected (and deselected). In this way, the derivative hierarchy with all linked character data becomes assigned to a certain taxon. If the status or position of a taxon is changed during the revision of a taxonomic classification in the taxonomic perspective of the *Editor*, all appended sample metadata and character data remain with the taxon. If one taxon is united with another one, the appended sample metadata and character data are synchronously moved with the taxon and their former placement is recorded. If a changed circumscription of a taxon requires the moving of specimens to another taxon, their former placement also is recorded.

[a] The full functionality reached in this thesis after the publication of this chapter in 2015 is described in chapter 5 (page 55).

4.4.5.2 Aggregating specimen-based character data at the taxon level

Iterative character data aggregation procedures are being implemented in the *EDIT Platform* for two different data types.

i. Occurrence data: primary aggregation of geographical coordinates will result in dot distribution maps in the Data Portal. Aggregation of combined area unit distribution and occurrence status data at the same or from lower to higher taxon ranks is currently operated using a corresponding transmission engine. This rule-based engine aggregates distribution information (including occurrence status data) for a given taxon and region, recursively using its subtaxa and subregions. In the case of conflicting status values, decisions are made on the basis of defined priority rules.
ii. Character data stored in *SDD*-compliant character-state-matrices: the *Xper*^{2} software for character data management and interactive identification [148], which is integrated into the *EDIT Platform*, provides algorithms for data aggregation, merging numerical data while appending categorical data. The primary aggregation of the specimen-based character data at taxon rank currently only tentatively allows the automated generation of a natural language taxon description from the matrix. However, a workaround is the manual editing of the data using a description template. The storage of structured character data also enables the use of the data matrix for interactive taxon identification with the aid of multi access keys accessible through the data portal's Keys Tab which we describe below.

4.4.5.3 Publishing sample metadata and character data with the CDM Data Portal

The *EDIT Platform*, unless in the individual installation of the software, allows the visualization of the data through its online *Data Portal*, which is customizable in its basic structure according to one of the principal aims: (i) a systematic revision or monograph providing maximum data, (ii) a flora or (iii) a checklist with the most restricted array provided. Classifications and taxon-related data are visualized in the portal and are accessible through a navigable taxon tree or via taxon name, area and subject searches. A data portal with the function of a systematic revision or monograph presents the information for each taxon, independent of its rank, in five basic tabs: (i) the 'general tab' displays the summarized taxon-based character data organized in feature chapters; (ii) the 'synonymy tab' displays the detailed synonymy and typification data organized in blocks of homotypic synonyms; (iii) the 'image tab' displays stored images; (iv) the 'key tab' offers identification keys (interactive or single access) optionally for taxa including subordinate taxa and (v) the 'specimen tab' finally displays the investigated or determined specimens with their derivative hierarchies and available character datasets, as well as a dot distribution map for the taxon based on the georeferenced specimens (Figure 4.6). Setting the 'publis' flag in the *Taxonomic Editor* for a specimen derivative hierarchy and the appended character datasets displays these data in the *Data Portal*. A search function, still in preparation, will allow users to filter certain derivative types and their data in the specimen tab of the *Data Portal*. For each specimen and its derivatives besides the expanded table view, a separate page with metadata, character datasets and links to other available character datasets can be opened (Figure 4.6). The *Campanula Portal* [185] (see, under 'Preview' on the 'Welcome' page, the exemplar taxa listed) is being used to visualize exemplars of taxa with various types of specimen-based datasets.

Using the publication services of the *EDIT Platform*, more specific outputs can be designed for publication of subsets of data in print or electronic publication media.

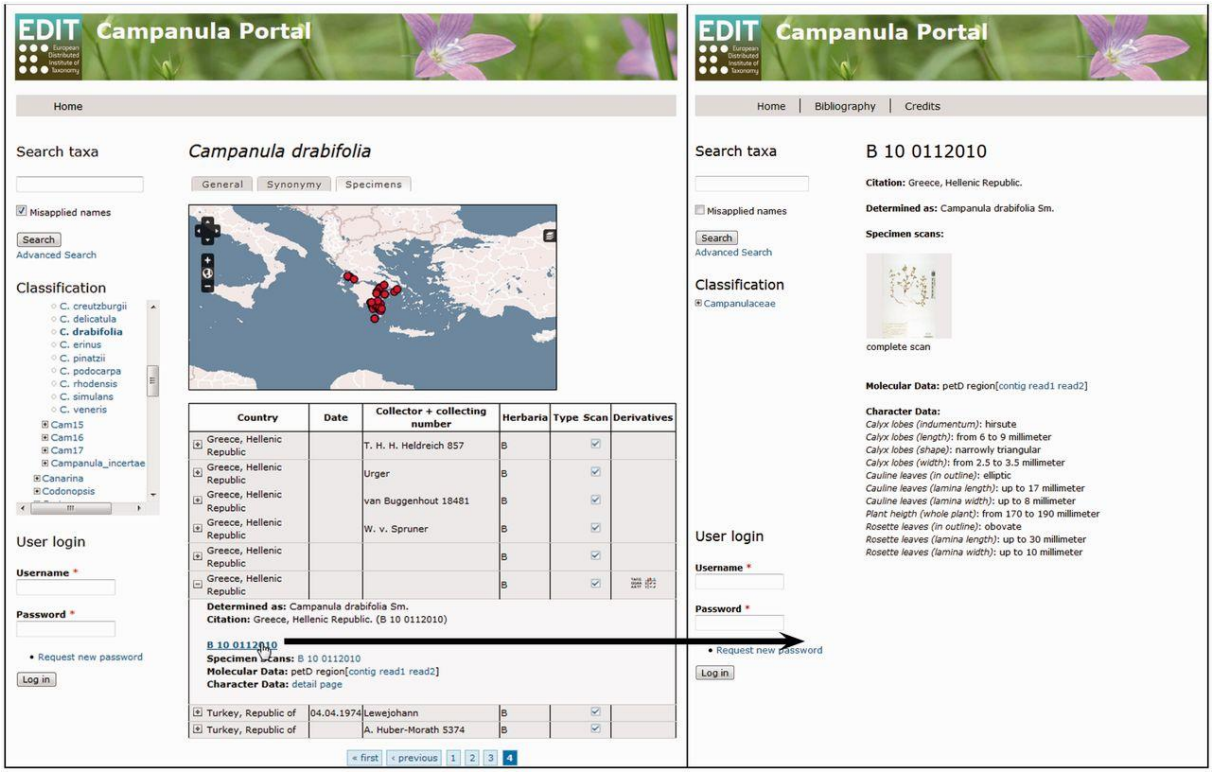


Figure 4.6 Data Portal of the* EDIT Platform*: screenshot of the* Campanula data portal *displaying the specimen tab visualizing the specimens and their derivatives available for a taxon

The Derivatives column indicates availability of additional datasets by displaying the respective icons. Clicking on a row (A) folds out the table cell and the listed items (here, specimen scan, DNA sequence contig and trace files) can be accessed by following the links given. The specimen ID functions as a link (B) to a separate specimen page where all derivatives of this specimen are clearly arranged, character datasets are provided and respective files are linked; clicking on the specimen scan thumbnail (C) opens the specimen scan in a separate browser window.

4.4.6 Data exchange via standard exchange formats and enabling persistent, specimen-linked storage in research collections

Exchange of sample metadata between the *EDIT Platform*, research collections, biodiversity networks and collaborators is managed using *ABCD* [157] and *Darwin Core* [158] as the standard exchange formats. Both formats allow the import and export of the combined sample metadatasets of entire derivative hierarchies, such as represented, e.g. by the specimen with its scan and tissue sample collection for DNA extraction. Moreover, data import of such a derivative hierarchy further extended for isolated DNA sources plus marker consensus sequences with their contig files and corresponding pherograms has successfully been tested from the *GGBN network* [182] to the *CDM Platform*. Even our still further reaching concept of persistently and stably linking morphological and other types of character data with sample metadata and combined sample metadatasets of entire derivative hierarchies is already possible. *ABCD* currently offers a container element (`<MeasurementsOrFacts>`), which can be used as a workaround to store atomized data, a complete character data matrix or a link to such a matrix. In this way, the exchange of the sample metadata with the respective research collection can include the information about and, optionally, a stable link to existing character datasets, or even the stored character datasets themselves. In the proposed follow up project and in connection with the development of *ABCD* 3.0, we envisage a more straightforward implementation for the exchange of associated structured character datasets. This will lay the foundation to popularize the association of (structured) character data with sample metadata, as well as their display and effective accessibility for humans and machines in research collections, ensuring high visibility and instant re-usability of character data through research collections.

4.5 Perspectives

Our solution emphasizes the editing and enrichment of specimen metadata (e.g. taxon identifications, nomenclatural type status designations, georeferencing) by the researchers in the course of their examination of the material, as well as on the synchronization of edited data with the existing datasets. Doing so, it takes into account that the rapid advancements in the digitization of research collections have conducted the work and data flows related to collections in an analogous and a digital branch. Consequently, solutions have to be designed for the various use cases to ensure that revised and enriched metadatasets can conveniently be connected to the collections [183,186].

Moreover, our solution, which streamlines the taxon characterization through establishing a persistent unambiguous relation between each sampled individual and the corresponding data, also opens new opportunities for the old problem of securing raw data associated with the research process in systematic biology. Primary research data do not only include pure data but also digital representations of preparations from specimens, ranging from light or scanning electron micrographs to sequencing trace files. Currently, if specimen-based character data are recorded, these are frequently treated as raw data, not usually included in publications, or, e.g. micrographs, published in a very limited selection only. At best they have, in more recent times, been deposited in repositories [42,187,188], otherwise they are still frequently considered only worth short-term preservation and disposed after the compulsory periods of record keeping, if not earlier [189]. The deeper reason for not preserving raw data is often the lack of appropriate means to document, persistently link and visibly store them. Additionally, individual research databases are often not integrated in institutional data management strategies [190]. National research funding bodies increasingly recognize the need for permanent storage facilities for primary research data [188,191]. However, the investment of extra work for long-term storage of specimen-based character data in a meaningful way is not economic as long as their re-use is not well organized. Primary research data therefore must appear effectively visible in a potential use context, must be technically compatible and so on. Evidently, the mere presence of data in some sort of public repository does not ensure their actual availability in a relevant research context. To become effectively visible, a firm, persistent link from the metadata of the deposited specimen to the respective character data in a repository would be a solution. Such links can be stored and conveniently exchanged in the standard metadata exchange formats for specimens (*ABCD* or *Darwin Core*). When accessing such a specimen, e.g. via online specimen catalogues, the link to existing character data sets becomes readily available. Alternatively, the array of specimen-associated data can be extended to also include character datasets themselves. Recently, a system of persistent http-URI identifiers for collection items associated with the digital representation of a specimen was suggested by Hyam et al. [192], which immediately gained wide acceptance and has been further elaborated since [193]. Using this system, the inclusion of character data into the array of specimen-associated data would make an attractive functional solution, facilitating brief, precise and convenient reference in scientific publications to a specimen with its digital image (if available), its metadata and existing character datasets. Such a solution would certainly help to increase significantly the visibility and re-usability of character datasets. Research collections are currently in a far-reaching process of transformation from curating pure analogous to sizable and complex collections of analogous and digital objects with the related datasets. Extending curation to specimen-based character data may secure research collections to play an appropriate key role in current and future research in systematic biology and thus in biodiversity assessment and analysis.

Our solution blazes a trail in systematic biology research for a streamlined process of taxon characterization and the additivity and re-usability of character data. The implementation is expected to be operational and available for download by the end of the project in December 2015. We have started to use this implementation in the integrative and dynamic approach for monographing the angiosperm

order Caryophyllales [128]. The current implementation has focused on various aspects of sample and data associations, while has relied on available software for the handling of morphological character data and for their aggregation from specimens to taxon characterization as well as from lower to higher taxon levels. The entire field of morphological character data aggregation, however, is waiting to become a subject of further developmental work. This concerns in particular three complexes: the modelling of character data [125]; semantic web solutions for ontologies of descriptive terms [125,194]; the exchangeability of data and the interoperability of different character data matrices (e.g. merging procedures for data matrices).

4.6 Acknowledgements

The concept of the modules for handling metadata of DNA-related derivatives has been developed in coordination with and based on work by Gabriele Droege (*BGBM*, Berlin) for the *GGBN network*. The concept of linking derivatives and their visualization in the *EDIT Data Portal* has been developed in discussion with and using the work by Wolf-Henning Kusber (*BGBM*, Berlin) for *AlgaTerra*. The distributed database query addressing specimen data providers on standardized protocols was implemented in cooperation with Jörg Holetschek (*BioCASe network*) and Patricia Kelbert (*DFG* project *BinHum*). The work done by the developers of the other open source projects (from the *Eclipse* and *Apache foundations*, *BioJava*[b], *NeXML*) our software is built on is highly appreciated. We thank Katy Jones (Berlin) for her critically reading and linguistically polishing an earlier version of the text and two anonymous reviewers for valuably commenting on the original submission.

4.7 Funding

German Research Foundation (*DFG*, *Deutsche Forschungsgemeinschaft*) within the *Scientific Library Services and Information Systems programme* (KI 1175/1-1, MU 2875/3-1). Funding for open access charge: *DFG*-funded *Open Access Publication Fund* of the *Freie Universität Berlin*.

Conflict of interest: None declared.

[b] The molecular components of the *Taxonomic Editor* were using *BioJava* at the time of publication of this chapter in 2015. This dependency has been removed since then.

5 The molecular components of the *Taxonomic Editor*

Stöver BC[1]*, Plitzner P[2], Kilian N[2], Henning T[2], Müller A[2], Güntsch A[2], Berendsohn WG[2], Borsch T[2], Müller KF[1]

[1]Institute for Evolution and Biodiversity, WWU Münster, Hüfferstraße 1, 48149 Münster, Germany

[2]Botanical Garden and Botanical Museum Berlin-Dahlem, Dahlem Center for Plant Sciences, FU Berlin, Königin-Luise-Straße 6-8, 14195 Berlin, Germany

*Author for correspondence: stoever@bioinfweb.info

Own contribution

I implemented the molecular module (module `eu.etaxonomy.taxeditor.molecular`) of the *Taxonomic Editor* that is described in this chapter in greater detail than the previous chapter that dealt with the general concepts of the *Taxonomic Editor* and the way it models taxonomic workflows. Patrick Plitzner helped with the integration of the new components into the application and coordinated the work between Münster and Berlin. Andreas Müller added required implementations for the *CDM API* to provide interoperability with the new molecular functionality. I developed the concept for the required libraries *LibrAlign* (chapter 3, page 46) and *JPhyloIO* (chapter 2, page 33) and implemented them, designed the export feature for molecular data of the *Taxonomic Editor* with the required draft for a respective ontology (see Figure 5.3) and wrote the manuscript.

Abstract

The functionality of the *Taxonomic Editor* of the *EDIT Platform for Cybertaxonomy* has been extended to model molecular sequence data within taxonomic workflows. Single read sequences are generated from samples of specimens and these are combined to a consensus marker sequence using a contig alignment. To model this, while still linking all data to the initial source specimen, an editor component for such contig alignments has been developed for the *Taxonomic Editor* based on the GUI components provided by *LibrAlign*. It allows to align a set of single reads sequences to create a consensus sequence and displays the pherogram raw data directly attached to the corresponding single read. Complete contig alignments can be exported using a new feature based on *JPhyloIO*. That allows to write to different alignment formats and exports links between sequences and their pherograms, if the metadata model of the target format allows it. Exports to *NeXML* contain all available annotations using a newly defined ontology that provides *RDF* predicates to formulate necessary statements describing a contig alignment.

5.1 Introduction

As described in the previous chapter the *Taxonomic Editor* of the *EDIT Platform for Cybertaxonomy* [102] manages taxonomic workflows with different types of character data while persistently linking them to individual source specimens [24]. This includes molecular data, i.e., DNA marker sequences obtained from isolations of probes of specimens and their source pherograms.

To fully model the workflow and to preserve the links between molecular data and the specimens it was derived from, the *Taxonomic Editor* requires components that allow to display, assemble and edit DNA sequences within the derivative hierarchy. The multiple sequence alignment editor *PhyDE* [195] was chosen as a possible provider for required features, since it has a phylogenetic focus, is developed in *Java* (like the *Taxonomic Editor*) and became open-source before the start of the project. *PhyDE* was undergoing major revisions (described in chapter 6, page 84) and future versions of *PhyDE* were to be based on *LibrAlign* (chapter 3), which now provides all necessary GUI components for visualizing and editing MSAs, including functionality for pherograms from Sanger sequencing. *LibrAlign* therefore also

became the basis for the new molecular components of the *Taxonomic Editor* and was designed to provide all its components not only for *Swing* GUIs (as required for *PhyDE* and other software), but also in a version for *SWT*, which is the GUI toolkit used by the *Editor*.

Since the data model of the *EDIT platform* models source data, like pherograms, persistently liked to derived sequences and contigs, the new components should ideally be able to display pherograms closely together with their base call sequences and allow to export contig alignments of these to formats that can model the links between the sequences and their source data. The data areas of *LibrAlign* that allow displaying any type of raw- and metadata directly within the GUI components used to edit sequences and MSAs, provide the necessary functionality to fulfill these needs. Their close integration with *JPhyloIO* (chapter 2) that provides access to a variety of file formats with full metadata access allows to easily create the required I/O functionality.

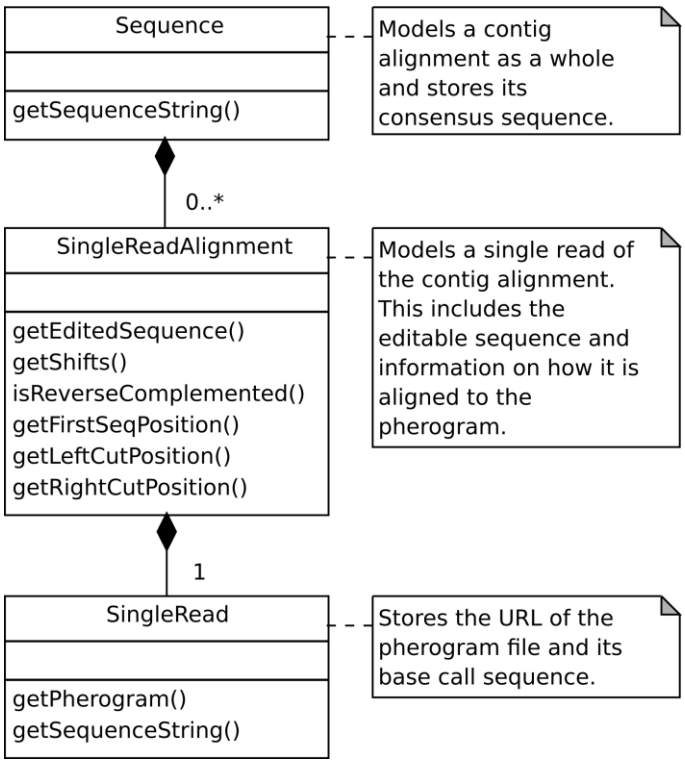


Figure 5.1 The* cmd-lib *modelling of an alignment of single reads

This UML *class diagram shows how the data of an alignment of overlapping single reads and the resulting consensus sequence is modeled by the* Java *API of the* common data model *of the* EDIT platform*. (A class diagram containing all CDM components that model molecular data is available at http://r.bioinfweb.info/CDMMolData.)*

5.2 Implementation

The *Taxonomic Editor of the EDIT Platform* (also called *EDITor*) is a *Java* application based on the *Eclipse Rich Client Platform* (*RCP*) [105]. *RCP* allows to specify so-called editors and views to display and edit different types of data, which can be freely combined and arranged by the user. The molecular components of the *Taxonomic Editor* that have been developed within this thesis are contained in the *Eclipse RCP* plugin `eu.etaxonomy.taxeditor.molecular`, while the necessary dependencies are provided by `eu.etaxonomy.taxeditor.molecular.lib`.

All data to be displayed and edited using the *EDITor* is provided by a data service following the standard of the *common data model* (*CDM*) of the *EDIT platform* [161,196]. Technically, the *EDITor* uses the *Java* API *cdm-lib* to access data from a respective database. Figure 5.1 shows how an alignment of single reads is modeled in *cdm-lib*.

To display and edit alignments of single reads and their consensus sequences, a special alignment editor *RCP* component has been developed that makes use of the GUI components provided by *LibrAlign*. As Figure 5.2 shows, the single reads and their source pherograms are displayed in an `AlignmentArea`, while another `AlignmentArea` allows to edit the consensus sequence of all reads. Both are grouped together by a `MultipleAlignmentsContainer`. (See chapter 3.2.1 on page 47 for details on *LibrAlign* components.) In addition to the alignment editor, a view component to display pherograms separately is also based on *LibrAlign*. The user therefore has the options to view pherograms directly attached to the respective single read sequence and to display a larger visualization of the trace curves in a separate view component.

Exporting the alignments of single reads and their consensus sequence to different multiple sequence alignment formats is achieved using *JPhyloIO* (cf. chapter 2.2.2, page 38) and the I/O module of *LibrAlign* (cf. chapter 3.2.4, page 48).

5.3 Results and discussion

Figure 5.2 Screenshot of the* Taxonomic Editor *with an opened alignment editor and pherogram view

This screenshot shows an alignment of single reads that is opened using the alignment editor component from the `eu.etaxonomy.taxeditor.molecular` *module that was developed within this thesis. An alignment area contains the editable single read sequences with their attached source pherograms and another alignment area below contains the consensus sequence of all reads together with a bar-sum chart showing the nucleotide frequencies in all reads for each position.*

The pherograms displayed below the single read sequences are displayed in a way that their peaks match the position of the corresponding nucleotides. By double-clicking a pherogram, the pherogram view (at the bottom of the window) is opened providing an undistorted representation of the trace curves together with their base call sequence.

DNA sequences are generated from specimens and are modeled as their derivatives in the data model of the *EDIT platform* (see [24] for details). Consensus sequences are derived from a set of single reads (e.g. from Sanger sequencing) that are overlapped and aligned to form a contig. As mentioned above, the *RCP* components “alignment editor” and “pherogram view” (visible in Figure 5.2) have been added to the *Taxonomic Editor* to display and edit such contig alignments. Each single read can be imported from a pherogram file in *AB1* or *SCF* format. The alignment editor component distinguishes between the immutable base call sequence read from the pherogram file (displayed directly above the trace curves with white background) and an editable copy of it (displayed above the immutable base call sequence with colored background, Figure 5.2). This way, manual corrections and edits of the base call sequence are directly visible in comparison to the original sequence. Information on how to align the original sequence with its editable copy is collected during editing and stored in the data model. When editing a single read the user can select whether to insert additional nucleotides on the right or the left side of the cursor (Figure 5.2), which differs in whether the trace curve is distorted to the left or to the right to match the additional nucleotide.

```
<?xml version="1.0" ?>
<nexml xmlns="http://www.nexml.org/2009" version="0.9"
    generator="JPhyloIO 0.2.0-1355 alpha"
    xmlns:nex="http://www.nexml.org/2009"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns:ra="http://bioinfweb.info/xmlns/PhyDE/ReadAlignment/Predicates/"
    xmlns:radt="http://bioinfweb.info/xmlns/PhyDE/ReadAlignment/DataTypes/"
    xmlns:pha=
        "http://bioinfweb.info/xmlns/PhyDE/ReadAlignment/PherogramAlignment/">
  ...
  <characters>
    ...
    <matrix>
      <row id="seqSingleRead0" about="#seqSingleRead1" label="Single read 0"
          otu="undefinedOTU0">
        <meta id="seqSingleRead0METAisSingleRead" xsi:type="nex:LiteralMeta"
            property="ra:isSingleRead" datatype="xsd:boolean">true</meta>
        <meta id="seqSingleRead0METAisRCed" xsi:type="nex:LiteralMeta"
            property="ra:isRCed" datatype="xsd:boolean">false</meta>
        <meta id="seqSingleRead0METAhasLeftCutPosition" xsi:type="nex:LiteralMeta"
            property="ra:hasLeftCutPosition" datatype="xsd:int">4</meta>
        <meta id="seqSingleRead0METAhasRightCutPosition" xsi:type="nex:LiteralMeta"
            property="ra:hasRightCutPosition" datatype="xsd:int">834</meta>
        <meta id="seqSingleRead0METAhasPherogram" xsi:type="nex:ResourceMeta"
            rel="ra:hasPherogram" href="http://example.org/Pherogram.scf"></meta>
        <meta id="seqSingleRead0METAhasPherogramAlignment"
            xsi:type="nex:LiteralMeta" property="ra:hasPherogramAlignment"
            datatype="radt:pherogramAlignment">
          <pha:shifts>
            <pha:shift pha:pos="27" pha:shift="-1"></pha:shift>
            <pha:shift pha:pos="295" pha:shift="1"></pha:shift>
            ...
          </pha:shifts>
        </meta>
        <seq>ACTTTCCGAAA...</seq>
      </row>
      ...
      <row id="seqConsensus" about="#seqConsensus" label="Consensus sequence"
          otu="undefinedOTU1">
        <meta id="seqConsensusMETAisConsensus" xsi:type="nex:LiteralMeta"
            property="ra:isConsensus" datatype="xsd:boolean">true</meta>
        <seq>TCCGAAA...</seq>
      </row>
    </matrix>
  </characters>
</nexml>
```

Figure 5.3 Example output of a single read alignment from the* Taxonomic Editor *using the* PhyDE *ontology

The NeXML *file shown here contains a single read alignment that has been exported from the* Taxonomic Editor *of the* EDIT platform *using its new molecular components. In addition to an example single read and a consensus sequence, multiple* `meta` *tags nested within the* `row` *tags are shown, which contain metadata modeled by the* EDITor.

The consensus sequence that is constructed by overlapping and aligning the single reads is displayed below and can be created automatically and edited manually. The bar-sum diagram below the consensus sequence allows to directly inspect differences (e.g. due to manual edits or conflicts) between the single reads and the consensus sequence. If additional single reads are added later, a special operation to update the consensus sequence is available that will optionally preserve previous manual edits.

A wizard based on *JPhyloIO* was added to export contig alignments of single reads and the resulting consensus sequence from the *Taxonomic Editor* to all alignment formats supported by *JPhyloIO*. This currently includes *FASTA*, *Phylip*, *Nexus* and *NeXML*. (See 2.2.3 on page 38 for details). The user can choose whether to include only the consensus sequence, the single reads or both into an export. For simple alignment formats like *FASTA,* the two separate types of sequences are treated as sequences of one alignment, while metadata is exported to *NeXML* that identifies each sequence as a single read or a consensus sequence. Additionally, a link to the pherogram file and all information to align the original base call sequence with the exported single read sequence is written to *NeXML*. Figure 5.3 shows an example *NeXML* file exported by the *Taxonomic Editor*. An ontology with necessary predicates was defined to link this information to the sequences. It will also be used and possibly extended in future versions of *PhyDE 2* (chapter 6).

The molecular plugins providing the functionality described here are incorporated into current versions of the *Taxonomic Editor*, which is available for download at https://cybertaxonomy.eu/taxeditor/. The latest source codes of the plugins can be found at http://r.bioinfweb.info/EDITorMolSrc.

5.4 Conclusion

The new components developed in this thesis extend the functionality of the *Taxonomic Editor* to model molecular data as a derivative of specimens. Having this functionality directly integrated into the *EDITor* allows to persistently link DNA data to the specimen it was extracted from and not only to a taxon as done by almost all established databases. Additionally, the source pherograms of all single reads of each sequence and all changes made to them are modeled and documented, resulting in greater reproducibility and reusability of data. The export functionality to various alignment formats ensures interoperability to a large number of other software and databases.

5.5 Availability and requirements

Project name: Molecular components of the *Taxonomic Editor* of the *EDIT platform for Cybertaxonomy*

Project web page: https://cybertaxonomy.eu/taxeditor/ (Project page for the *Taxonomic Editor* as a whole. Downloads include the molecular components.)

Source codes: http://r.bioinfweb.info/EDITorMolSrc

Operating system: Platform independent

Programming language: *Java* (*Eclipse RCP*)

Other requirements: *Java* Runtime Environment 8 or higher (Versions with a bundled *JRE* with reduced functionality are also available.)

License: *Mozilla Public License* Version 1.1

Any restrictions on use by non-academics: None

5.6 Acknowledgements

The work of the contributors of the open source projects used is highly appreciated (*Eclipse RCP*, *Apache commons*, *BioJava*, *OWL API*, *SWT*, *JUnit*, *Hemcrest*). Funded by grant MU 2875/3-1 to Kai Müller by the *German research foundation* (*DFG*).

6 A new version of the alignment editor *PhyDE* based on the recently developed functionality of *JPhyloIO* and *LibrAlign*

Stöver BC[1]*, Bohn J[1], Quandt D[2], Müller KF[1]

[1]Institute for Evolution and Biodiversity, WWU Münster, Hüfferstraße 1, 48149 Münster, Germany

[2]Nees Institute for Biodiversity of Plants, University of Bonn, Meckenheimer Allee 170, 53115 Bonn, Germany

*Author for correspondence: stoever@bioinfweb.info

Own contribution

See section 6.6.1 on page 89.

Abstract

PhyDE is a multiple sequence alignment (MSA) editor for phylogenetic purposes that is used in the scientific community since 2005. Recently the *Java* library *LibrAlign* was developed, which provides flexible reusable GUI components to display and edit MSAs and a new version of *PhyDE* was created based on the components.

Compared to the previous version, *PhyDE 2* now has a much easier extendable and maintainable code base that simplifies its future development and adjustment to the needs of the phylogenetic community. The project is currently in a proof-of-concept state and does not yet provide all features of the previous version, but the currently available release already supports editing and displaying MSAs together with their character sets and reading and writing of more alignment formats than the initial version. Its I/O functionality is based on *JPhyloIO*, another recently developed *Java* library that provides access to a variety of phylogenetic file formats, including their full metadata model. *PhyDE 2* uses *NeXML* as its main format, replacing the *PDE* format of the previous version to achieve greater interoperability. *LibrAlign* contains modules that directly integrate its components with *JPhyloIO* and both libraries put a special emphasis on modeling any type of metadata closely together with the phylogenetic data they are attached to.

The future development of *PhyDE 2* will first focus on reaching feature equivalence with the previous version and then extended modelling of sequence and alignment metadata will be a key aim. As currently implemented in *TreeGraph 2*, future versions of *PhyDE 2* are also planned to enable its users to make full use of the *RDF*-based metadata model of *NeXML* in a convenient way and therefore increase reusability of their data and reproducibility of their studies by providing necessary annotations. Creating a workbench application that combines the functionality of *PhyDE 2* and *TreeGraph 2*, including their extended metadata models, covering the major types of phylogenetic data, is another future perspective that is opened-up by the codebase of the new version.

6.1 Introduction

The first version of the alignment editor *PhyDE* [195] was initially released in 2005 and focuses on sequence alignment for phylogenetic purposes. Key features are various options for efficient manual alignment and linking sequences to raw data in Sanger sequencing pherograms.

Recently the *Java* library *LibrAlign* was developed, which provides flexible GUI components for displaying and editing multiple sequence alignments, including functionality to display raw- and metadata, like pherograms or character sets (chapter 3). It is already used in the molecular components of the

Taxonomic Editor to add functionality for creating alignments of single reads to combine them to a consensus sequence (chapter 5).

Since the old code base of *PhyDE* was difficult to extend and *LibrAlign* now provides most of the necessary functionality, it makes sense to base future versions on the library, so that they can directly benefit from improvements and extension made in *LibrAlign*. *JPhyloIO* (chapter 2) is another recently developed library that provides access to a variety of multiple sequence alignment file formats that are relevant in phylogenetics and is integrated into *LibrAlign*. This way new *PhyDE* versions based on *LibrAlign* can automatically make use of the functionality provided by *JPhyloIO* and support more formats for greater interoperability without additional effort. A new version of the alignment editor based on *LibrAlign* and *JPhyloIO* has been developed in this thesis and this chapter describes the currently available functionality of *PhyDE 2* and perspectives for its future development.

6.2 Implementation

PhyDE 2 is a *Swing* application that makes use of the GUI components provided by *LibrAlign*. The main window contains a `MultipleAlignmentsContainer` in which multiple `AlignmentArea`s are nested. (See chapter 3.2.1 for details on *LibrAlign* components.) As shown in the screenshot in Figure 6.2, the first alignment area contains a ruler with the column indices and the second one displays character sets. The third one contains the actual alignment and allows to edit it. The relation between the three alignment areas, their container and the other components of the application are shown in Figure 6.1.

The model of the application consists of a multiple sequence alignment that is stored in an instance of `PackedAlignmentModel` and a list of character sets that are stored in an instance of `CharSetDataModel`. Both of these model classes are implementations available in *LibrAlign* and are grouped together by the *PhyDE 2*-specific class `Document`. All model classes are shown in yellow in Figure 6.1. What is not shown in the figure (for clarity) is that a *LibrAlign* decorator instance is wrapped around `PackedAlignmentModel`, which creates respective undo objects for each change of the document contents. With this decorator, *LibrAlign* already provides functionality to undo and redo alignment edits in a flexible way that allows the application to define the actual type of undo class to be used. To provide undo functionality also for character set-related modifications of the model, respective undo objects for modifications of the character sets have been implemented in *PhyDE 2*. *LibrAlign* provides hooks to wrap its own alignment undo objects into application-specific objects, which allows to manage all undo objects together in a single undo manager.

All operations available in *PhyDE 2* are implemented as separate action objects, which allow easily to offer the same operation at multiple positions in the GUI, e.g., in the main menu and the toolbar. Those actions that trigger modifications of the model, create the undo objects mentioned above that perform the actual modification.

Reading and writing of alignment files is done using the I/O module of *LibrAlign*, which is based on *JPhyloIO*. Therefore, all formats supported by *JPhyloIO* are also supported by *PhyDE 2*, while *NeXML* [35] is the main format. *PhyDE 2* distinguishes between general *NeXML* files created by other applications and files created by *PhyDE 2*. The latter contain a metadata element on the document level that identifies them as *PhyDE 2* files. While such files can be directly opened and are overwritten when the user saves, other *NeXML* files (without the metadata marking) are treated as imports (like files in other formats) and the user needs to select a new location before saving them. This is done because the current version of *PhyDE 2* only supports to write a single alignment to a file, while *NeXML* in general supports to have multiple alignments together with phylogenetic trees, OTU lists and various metadata within the same file. If a *NeXML* file is identified as a native *PhyDE 2* file using the respective metadata element, the application can be sure that it models all contained data so that it will not lose information

when writing back to the file. Files from other applications are never overwritten, to make sure that no additional information is lost.

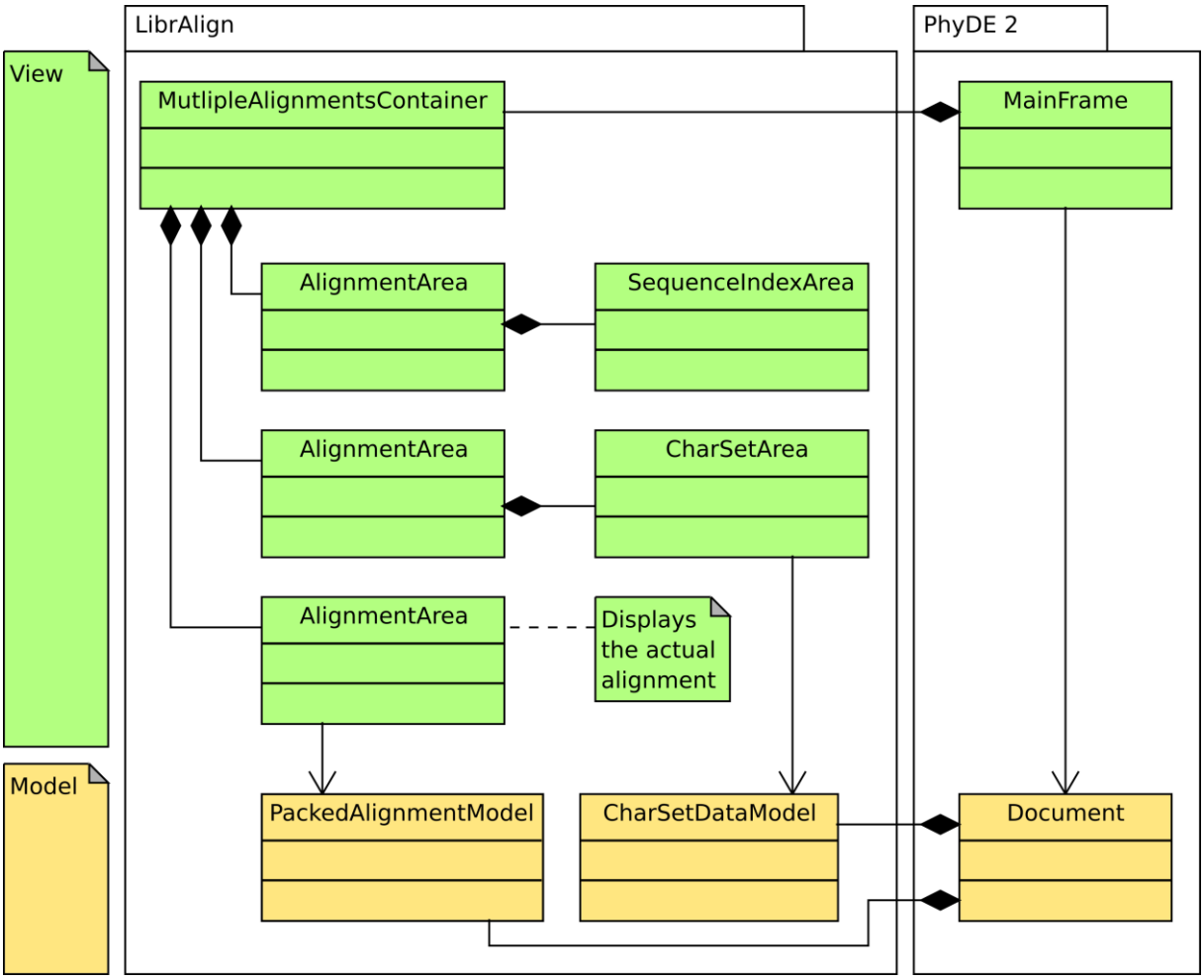


***Figure 6.1 UML diagram showing the internal architecture of* PhyDE 2**

View instances (according to the model-view-controller paradigm [106]) are shown in green, while model instances are in yellow. The `MainFrame` *is the top-most view of the application containing multiple views from* LibrAlign*. Two alignment areas only contain a single data area and the third one displays the actual alignment. Both the third* `AlignmentArea` *and the* `CharSetArea` *have an associated model implementation from* LibrAlign*. The* `Document` *class from* PhyDE 2 *represents the main model class and groups the two different* LibrAlign *models together. (Note that the used alignment model is actually nested within another decorator instance that creates undo objects, which is not shown here.)*

6.3 Results and discussion

6.3.1 User interface

The *PhyDE 2* main window (Figure 6.2) contains *LibrAlign* components to edit a multiple sequence alignment and a set of attached character (column) sets, as mentioned above. The user can navigate through the alignment and the character set area using the mouse or the keyboard and select a rectangular set of cells or a character set. Operations like editing, adding or removing sequences or character sets are available from the main menu and from the tool bar or via keyboard shortcuts. Sequence editing can additionally directly be done via keyboard inputs, which is a feature from *LibrAlign*. The height of the alignment cursor of *LibrAlign* can be changed allowing users to edit multiple sequence at a time. (See chapter 3.3.1 on page 49 for details.) All editing operations can be undone and redone at any time using the toolbar, the keyboard shortcut or the edit list from the main menu.

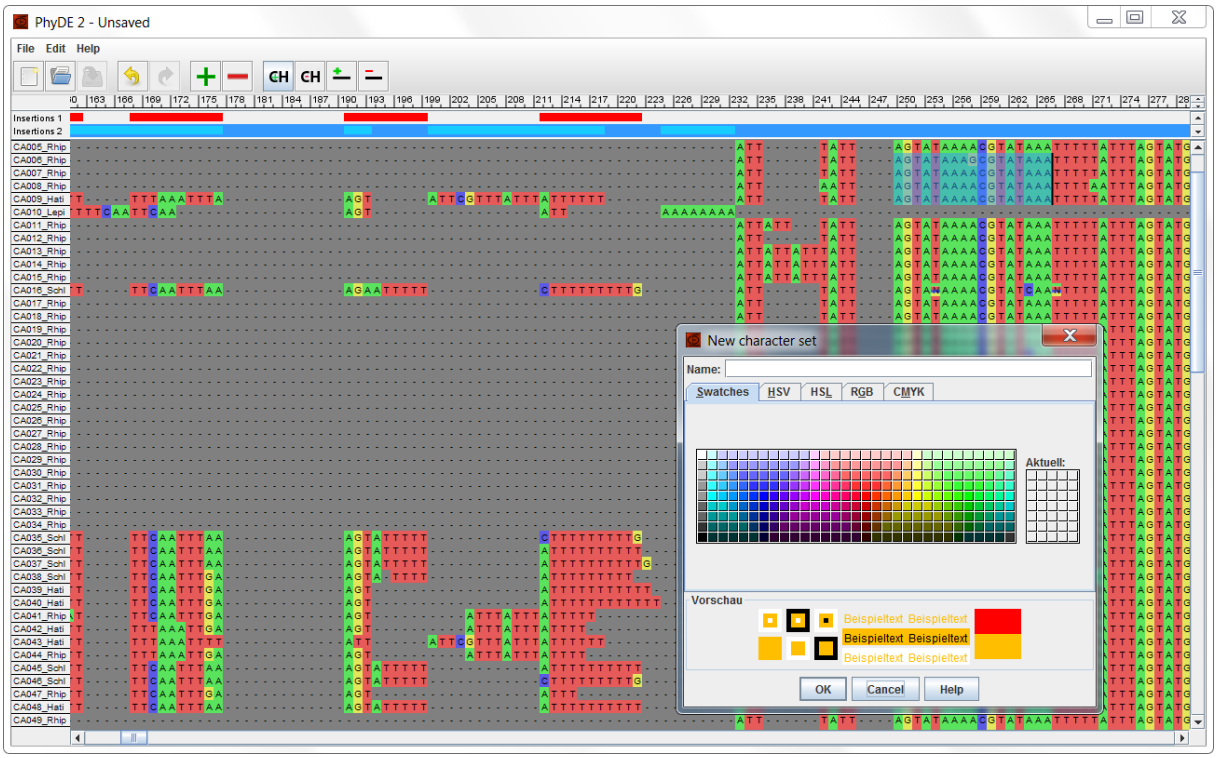


Figure 6.2 Screenshot of the* PhyDE 2 *main window with an opened dialog

This screenshot shows an opened alignment with two character sets. Both the character set and the alignment area allow user selections by mouse or keyboard, which are visible as blue shaded areas. The currently opened dialog allows to create a new character set by entering a name and a color. All currently available features are accessible from the main menu and selected ones additionally from the tool bar.

6.3.2 Supported formats

Since *PhyDE 2* uses *JPhyloIO* to read and write multiple sequence alignment files, it supports all formats that are supported by this library. It can therefore read and write *NeXML* [35], *FASTA*, *Nexus* [31], *Phylip* [32] and *Relaxed Phylip* [33] and additionally read files produced by *MEGA* [69] and the *PDE* format used by the previous version of *PhyDE*. Chapter 2.2.3 contains details on the formats and the different supported elements. If an input file contains more than one alignment, the user can select the one to be imported into *PhyDE 2*. Besides alignment data, *PhyDE 2* is also able to read and write character sets from all formats that model them, i.e. *NeXML* and *Nexus*. Character set definitions from *MEGA* and *PDE* files can also be imported.

As explained in chapter 6.2, *NeXML* has replaced *PDE* as the main format of *PhyDE*. This and the increased number of supported formats, significantly increases the interoperability of *PhyDE 2*. It now allows to import and export the majority of alignment formats relevant in phylogenetics, which is the domain current versions are mostly used in.

6.3.3 Comparison to other software

Other editors for multiple sequence alignments exist and numerous additional ones have been developed since the first release of *PhyDE* in 2005. *JalView* [78,197], *Mesquite* [111], *MEGA* [69], *AliView* [110], *STRAP* [112], *PFAAT* [113], *SeaView* [114], *DNAAlignEditor* [115] or *ALINE* [116] are examples of alternative alignment editors that focus on different aspects, like supporting very large alignments, additionally visualizing three dimensional protein structures, producing publication-quality alignment images or providing an integrated phylogenetic workspace. While it does not make much sense to compare these alternative applications in detail with the current version of *PhyDE 2*, since it is still in an early development stage and not yet feature-complete and comparing the first version of *PhyDE*

would be beyond the scope of this chapter, *PhyDE 2* is now already the only application besides *Mesquite* that supports the same number of phylogenetically relevant alignment formats. This is also due to other established alignment formats in other subdisciplines of biology, which are more the focus of the many of the other alignment editors. An important reason why the first version of *PhyDE* was developed and is used until today by many phylogeneticists, are the versatile manual alignment options that are not offered by most other editors. These will continue to be a focus in the future development of both *PhyDE 2* and *LibrAlign* and to some degree they are already available in their current versions.

6.3.4 Future development

The current version of *PhyDE 2* is an initial draft and a proof-of-concept for using *LibrAlign* together with *JPhyloIO* as the basis for a fully operational alignment editor. While *PhyDE 2* already offers functionality not available in the initial *PhyDE* (e.g. the support for more formats), there are other features of the first version that are currently not available. Future development will first focus on integrating the existing functionality of *LibrAlign* to display pherograms as a replacement for the pherogram view of the first *PhyDE* version. In addition to a component that displays a pherograms in a separate component, *LibrAlign* offers a data area that can display a pherogram directly below its corresponding editable sequence in an alignment and distorts its trace curves to match the equal widths of the displayed nucleotides. This functionality is already in use in the *Taxonomic Editor* (chapter 5) and goes beyond the capabilities of the old *PhyDE* version. *PhyDE 2* will then also be interoperable with the single read alignment editor of the *Taxonomic Editor*, by exchanging respectively annotated data via *NeXML*. An additional functionality to accomplish feature parity with the first version is translation between nucleotide and amino acid sequences, including the ability to define the reading frame on the borders of spacers or intron.

After all features necessary to fully replace the initial version of *PhyDE* have been completed, the further development could focus on making full use of the metadata support of *JPhyloIO* and *LibrAlign*. Similar to the extended metadata model of the phylogenetic tree editor *TreeGraph 2* (described in chapter 9.3.6), *PhyDE 2* could also allow to attach any combination of *RDF*-based metadata to sequences and whole alignments and store them in *NeXML*. This would bring significant advantages for conveniently creating alignment files that make phylogenetic studies more reproducible and their data more reusable. See chapter 1.1 (page 20) and chapter 9 (page 137) for more information on the relevance of data annotation and the usage of externally defined ontologies and *RDF*. Supporting the use of externally implemented data areas in *PhyDE 2* as a counterpart to externally defined ontologies, could also be a future perspective. Further details on that can be found in chapter 3.3.6 (page 54) and 13.1.3 (page 182).

As the data model of *NeXML* and *Nexus* files allows the combination of matrices and phylogenetic trees, it would be beneficial to have an editor capable of visualizing and modifying the whole contents of such files. Distinguishing between general *NeXML* files and *PhyDE NeXML* files, as described in chapter 6.2 would not be necessary anymore in such an application. In parallel to the individual applications *PhyDE 2* to process alignments and *TreeGraph 2* to process phylogenetic trees, a combined editor offering the features of both applications would be desirable. Further details on that can be found in chapter 9.3.8.3.

6.4 Conclusion

The currently available basic version of *PhyDE 2* is not yet a complete replacement for the initial *PhyDE* (although it even in this early stage offers some features the old version did not have), but represents an important step in restarting its development, which was inactive before. By making use of the functionality of *LibrAlign* and *JPhyloIO* that was implemented in this thesis, it creates a foundation for the

sustainable future development of *PhyDE* and makes the new achievements of the two libraries available to the *PhyDE* user community.

Although many other feature rich alignment editors exist today, an application that strongly focuses of phylogenetic purposes, e.g., by supporting respective formats and providing versatile editing options is still highly valuable for the scientific community. Furthermore, the ongoing and future extensions of the metadata models of *PhyDE 2* and *TreeGraph 2* can be an important step in providing users with tools to increase reproducibility and data reusability.

6.5 Availability and requirements

Project name: *PhyDE 2*

Project web page: http://bioinfweb.info/PhyDE2

GitHub Repository: https://github.com/bioinfweb/PhyDE2

ResearchGate project page: http://r.bioinfweb.info/RGPhyDE2

Operating system: Platform independent

Programming language: *Java*

Other requirements: *Java* Runtime Environment 8 (or higher)

License: *GNU General Public License* Version 3 (*GPL*)

Any restrictions on use by non-academics: The restrictions specified in the *GPL* apply. (See http://bioinfweb.info/PhyDE2/License.)

6.6 Declarations

6.6.1 Authors contributions

Ben Stöver developed the concept and architecture, wrote the manuscript and contributed to the implementation. Jonas Bohn implemented the current release during a research module supervised by Ben Stöver. Dietmar Quandt contributed to the concept of the software. Kai Müller contributed conceptually and to the manuscript.

6.6.2 Acknowledgements

We thank Jörn Müller for his work on the first version of *PhyDE* and his contributions the concept that was a model for the development of *PhyDE 2*. We are also thankful to Christoph Neinhuis for financial and other support of the development of the previous version. The work of the contributors of the open-source projects used is highly appreciated (*Apache Commons*, *Google Guava*, *OWL API*, *Hemcrest*, *JUnit*).

7 *AlignmentComparator*: Comparing alternative multiple sequence alignments of the same dataset

Stöver BC[1]*, Müller KF[1]

[1]Institute for Evolution and Biodiversity, WWU Münster, Hüfferstraße 1, 48149 Münster, Germany

*Author for correspondence: stoever@bioinfweb.info

Own contribution

See section 7.7.1 on page 126.

Abstract

Multiple sequence alignment (MSA) is an important step in phylogenetic workflows, but also in many other parts of the life sciences. Numerous different methods have been developed to date, while there is currently no agreement on which ones produce optimal MSAs, especially not under the criterion of homology for which no benchmark datasets exist, as they are available, e.g., for alignment by structure. Therefore, researchers are well advised to compare the results of different methods and identify regions of agreement and conflict, also because of the possible large influence of down-stream analyses.

We developed *AlignmentComparator*, a platform-independent application that allows to visually compare several alternative MSAs of a dataset and to annotate the differences. A fine-grained comparison is made possible by inserting gaps into all columns of some compared alignments to position equivalent regions closely together, even if the compared MSAs considerably differ in length. A set of developed algorithms produce different arrangements to address different user needs. *AlignmentComparator* is based on *JPhyloIO* and therefore allows importing MSAs from several different formats and uses *NeXML* (and its flexible *RDF*-based metadata model) to store comparison results, therefore ensuring high interoperability.

Beyond its use in examining differences between the results of alternative MSA algorithms, *AlignmentComparator* is also helpful to inspect subsequent changes made to MSAs during a workflow, either manually or using postprocessing software, and therefore makes such workflows more reproducible. It can also be used as a visualization tool in the development of new MSA algorithms and teaching.

Since it uses extensible GUI components from *LibrAlign* to display its comparison results, it would easily possible to display any type of sequence-related metadata directly within the comparison. Allowing users to consider the influence of any type of metadata on the results of MSA methods, using externally implement data areas of *LibrAlign* for externally defined annotations and therefore making full use of the potential of *NeXML* for *AlignmentComparator* is a perspective for its future development.

7.1 Introduction

Multiple sequence alignment (MSA) plays a key role in various parts of biology and bioinformatics and in many cases comparing alternative MSAs of the same dataset can be necessary. Bioinformaticians who work on improving alignment algorithms may want to compare differences between resulting MSAs from different versions of their implementations depending on the changes they made. If algorithms using iterative approaches (like [198–200]) are developed, a comparison tool could be used to inspect the differences between single iterations. Researchers can compare the outputs of various available software for automated multiple sequence alignment (e.g. [60–68]) to get information on which algorithm to use best for their specific problem (e.g. phylogeny reconstruction or many other applications of MSA) or compare automated MSAs with alternative manual alignments. In other cases existing MSAs are later modified, either manually or by software that performs post-processing (e.g.

[201,202]) for quality improvement or other modifications and the different versions may need to be compared to see what changed. Besides the application in research, programs for MSA comparison can also be of great use in teaching.

A detailed inspection of alignment differences is valuable in all the mentioned cases to asses, e.g., which parts of an alignment differ most, and which uncertainties exist that may also influence downstream analyses. The problem of creating optimal MSAs under the criterion of homology, structure or common function remains to be the subject of ongoing research, especially with regard to sequence alignment for phylogenetic purposes [203]. Therefore, comparing results from alternative approaches remains to be a relevant task for phylogeneticists and other researchers. Doing such a comparison manually, e.g., with two documents opened in a typical alignment editor, can be very time-consuming or sometimes even impossible, especially when length differences between the compared MSAs are large, due to different gap patterns or shifts between the single sequences. This problem is exacerbated for longer alignments, frequently found nowadays. To simplify such tasks, software would be helpful that visually compares alternative MSAs, e.g., by performing a superalignment of multiple alternative MSAs that displays similar regions close to each other, even if significant length-differences exist between the alternatives. Only few tools to visually compare MSAs are currently available, each with a different focus, and in particular performing a fine-grained comparison down to the nucleotide or amino acid level with a convenient and performant graphical tool remains an unsolved issue. (See chapter 7.4.4 for details on related software.)

Here we present *AlignmentComparator*, a tool for visual alignment comparison to address the described needs, which is based on the recently developed library *LibrAlign* (chapter 3) using its customizable GUI components to display multiple sequence alignments.

7.2 Algorithms

AlignmentComparator offers a set of alternative comparison algorithms that perform a superalignment, each of which has its strength in different applications. (See section 7.4.3.) We call a superalignment an alignment of alternative MSAs (usually of the same data set) that is used to visualize differences and agreements between them. Superalignment is done so that regions of agreement are displayed as closely together horizontally as possible. Gaps inserted to align the MSAs will be called supergaps. The following definitions describe this more formally:

Definition 7.1: A *supergap* in a multiple sequence alignment A is a gap that is present in all sequences in one or a set of subsequent columns of A. (It is inserted to create a superalignment of A with one or more other multiple sequence alignments.)

Definition 7.2: Be $A_1..A_n$ a set of $n \geq 1$ alternative multiple sequence alignments. A *superalignment* of $A_1..A_n$ is an alignment of $A_1..A_n$ that is achieved by inserting zero or more supergaps at any position(s) into $A_1..A_n$ so that they have the same length and their columns are aligned underneath each other by an optimality criterion. If $n = 1$ the superalignment is identical with A_1 and does not contain any supergaps. (Different optimality criterions may be used in different approaches.)

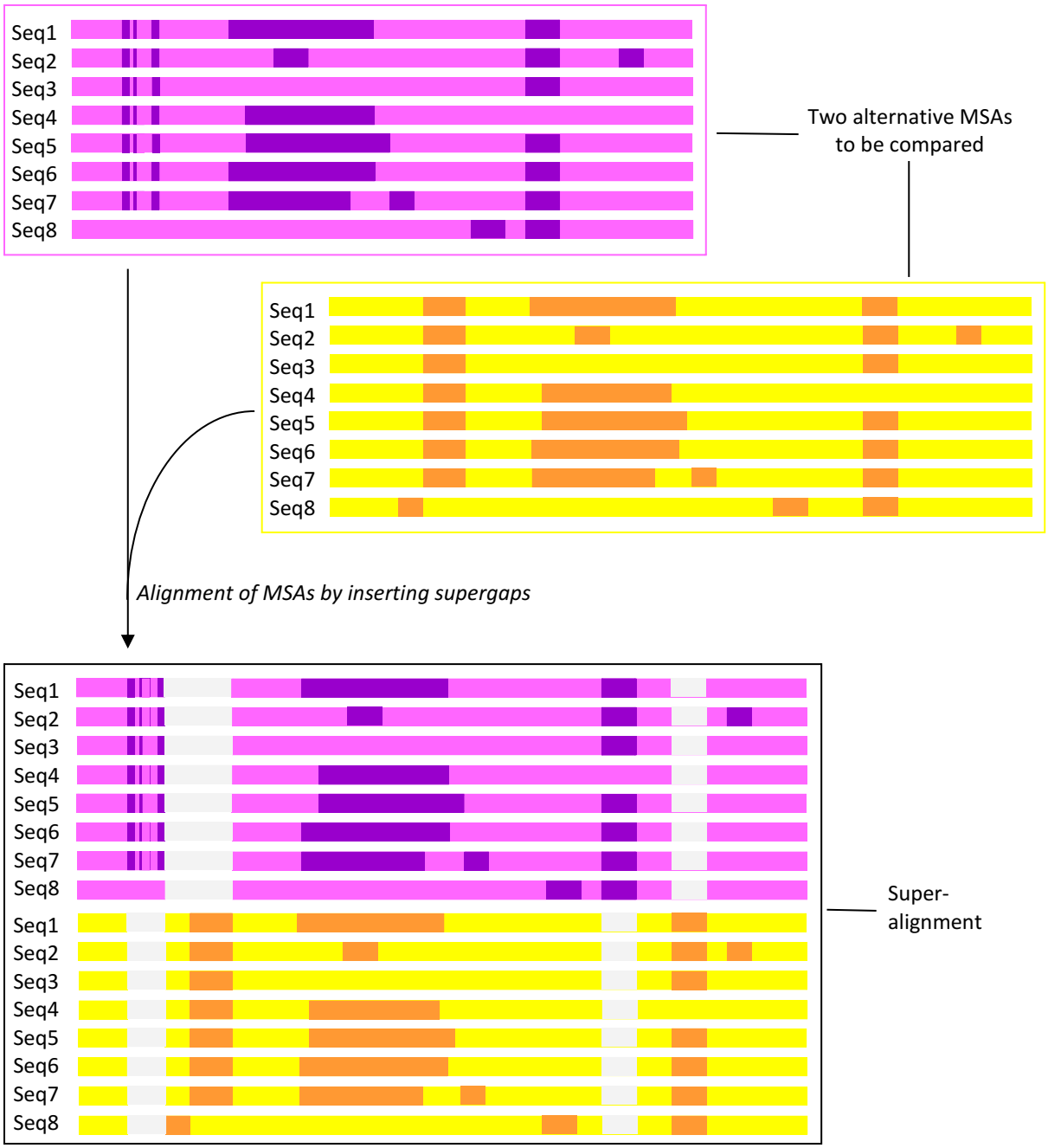


Figure 7.1 The principle of combining MSAs into a superalignment

The pink and the yellow MSAs align 8 sequences differently. The gaps are shown in the darker colors purple and orange. Some of them in the middle of both MSAs are identically placed while the gap patterns in the left and right regions differ between the alternatives. On the left, sequence 8 without gaps is longer compared to the other sequences. The pink alignment inserted multiple gaps into the shorter sequences to closely align them to sequence 8, while the yellow alignment is staggered and has larger gaps in both groups of sequences. On the right sequence 4 is longer than the other sequences and both alignments inserted one large gap into all other sequences to align them. The difference is that the pink alignment moves the gap further to the left than the yellow one, which could be due to a repetitive pattern in sequence 4.

To visualize these differences more clearly, supergaps (shown in gray) are inserted at positions where the MSAs differ to superalign them. As a result, regions with agreement are now arranged on top of each other and regions of disagreement are indicated by the presence of supergaps.

Definition 7.3: A *superalignment index list* is a list of integers that has the length of its corresponding superalignment. There is one superalignment index list for each MSAs that is aligned within the superalignment. Each entry is either a column index of the corresponding MSA (not the superalignment) or -1 indicating that this position of the superalignment contains a supergap inserted into the corresponding MSA. (By using superalignment index lists, only one list of integers needs to be stored per superaligned MSA, no matter how many sequences it contains.)

The way how supergaps are inserted into each MSA to achieve the best arrangement of similarly aligned regions depends on the superalignment algorithm that is used. *AlignmentComparator* currently implements three different algorithms that are described in the following sections. Figure 7.1 further illustrates the principle how supergaps are inserted to create a superalignment.

7.2.1 Profile alignment approach

Maybe the most straightforward approach to create a superalignment of two alternative MSAs is to combine them by performing a profile-profile-alignment. This operation is used in many implementations of MSA algorithms applying the progressive pairwise approach [204–206] to combine subalignments along the nodes of a guide tree. Among others, *MUSCLE* [68] offers command line options to create a profile-profile-alignment from two input files and is used by *AlignmentComparator* in the implementation of this algorithm. The positions of supergaps are then reconstructed from the combined output MSA. Algorithm 7.1 formally describes how superalignments are created using the profile alignment approach.

The space and time complexity are mainly determined by the profile alignment, which can be done in $O(n^2 + m^2)$ time and $O(n^2 + n \cdot m)$ space [207], where n is the average number of columns and m the number of sequences in each MSA. Algorithm 7.1 iterates over all sequences for each column and therefore has a time complexity of $O(n \cdot m)$, which is lower than the complexity of creating the profile alignment. The space complexity to store the resulting superalignment is also $O(n \cdot m)$. If this approach would be extended to superalign more than two MSAs along a guide tree (see chapter 7.4.5.1, page 122), the complexity of the additional steps would be as described in chapter 7.2.3.3 (page 110).

Algorithm 7.1 Creating a superalignment using the profile-profile-alignment approach

See http://r.bioinfweb.info/ACProfileImpl for the implementation of this algorithm.

An alternative way to design the function `createIndexList()`, would be to just check if all sequences with one prefix contain a gap at a certain position and insert a supergap there. The advantage of the chosen design is that it can successfully distinguish between supergaps from the profile-profile-alignment and columns in the initial alignment that contain only gaps (which is the case in some datasets).

Input:
- Two alternative MSAs to be compared: *A*, *B*

Output:
- Two superalignment index lists: L_A, L_B (See Definition 7.3.)

```
1  Add the prefix "A_" to all sequence names in A;   // Make sure all sequences from both MSAs
2  Add the prefix "B_" to all sequence names in B;   // have different names.
3  Create a profile-profile-alignment of A and B with MUSCLE and store the result in S.
4  L_A := createIndexList(S, A, "A_");
5  L_B := createIndexList(S, B, "B_");
6
7  function L := createIndexList(combinedAlignment, singleAlignment, prefix);
8     unalignedIndex := 0;   // The current column index in superAlignment
9     for superIndex := 0..length(combinedAlignment) - 1 do   // length() returns the # of columns
10       gap := false;
11       for all sequences c_i in combinedAlignment do
12          if sequence name of c_i starts with prefix then
13             Set s_i to the sequence in singleAlignment that corresponds to c_i.
14             if c_i[superIndex] != s_i[unalignedIndex] then
15                gap := true;
16             end if
17          end if
18       end for
19       if gap then
20          Add -1 to the end of L;   // Indicate a supergap here.
21       else
22          Add unalignedIndex to the end of L;   // Reference a column in the MSA here.
23          unalignedIndex := unalignedIndex + 1;
24       end if
25    end for
26 end function
```

7.2.2 Average position approach

The "average position" approach assumes that a good superalignment positions the corresponding tokens from corresponding sequences closely together. Since positioning tokens from different sets of sequences may be in conflict, the average distance between corresponding tokens from all sequences is minimized here to achieve an optimal superalignment. In contrast to the previous approach, this algorithm does not rely on the actual sequence tokens (e.g. nucleotides or amino acids) to calculate matching positions, but it considers the unaligned index of each non-gap token. If all alternative MSAs are derived from the same dataset (the same set of unaligned sequences), the corresponding token in one alignment can be found by calculating its index in the unaligned sequence (i.e., the sequence without gaps) and locate the token in the corresponding sequence of another MSA with the same unaligned index. (Note that this assumption does not hold, if the unaligned sequences of which the compared MSAs are made differ, e.g., due to different hot spot deletions. See also chapter 7.4.5.1 for a possible solution of this problem.)

Based on these ideas, this approach calculates the average position for every column of each MSA by summing up all unaligned positions from the tokens in a column and divides their sum by the number of sequences. (Note that taking all sequences into account for each column necessitates to calculate unaligned positions also for gap tokens, which is by definition not possible. A solution for this is described below.) Now that an average position is assigned to each column of each MSA, a superalignment can be created by inserting supergaps in a way that aligns columns with the closest average position together.

Unlike the other algorithms described here, this approach can be directly applied to an unlimited number of MSAs to be compared. The following sections describe this approach in detail.

7.2.2.1 Calculating the unaligned positions

In order to use average column positions for superalignment, they should be monotonically increasing. This way, the superalignment can be optimized with respect to preferably superalign columns with minimal distances while only having to compare distances of direct neighbors. (Otherwise a column might have its minimal distance to another non-neighboring column.) If average column positions would only be calculated from sequences that have non-gap tokens, average column positions would not necessarily be always monotonically increasing. Sequences may be heavily shifted between each other (with respect to their unaligned position in one column) and the average position might therefore even decrease from left to right, if different sets of sequences (without a gap in that column) are taken into account each time.

To solve this problem, this algorithm also assigns position values to gap tokens. These positions can be considered as equivalent to unaligned positions of non-gap tokens. Within a gap the position is calculated as the weighted average between the unaligned positions of the non-gap token before and after the gap, while the weighting depends on the relative position within the gap. To take into account that leading gaps have no token before them and trailing gaps have none after them, the start and the end of the alignment are considered as separate unaligned positions. Equation 7.1 and Equation 7.2 describe the calculation of unaligned positions in detail and Figure 7.3 (page 107) illustrates this on an example. The implementation of calculating the average position for a token can be found in the method `calculateUnalignedPosition()` at http://r.bioinfweb.info/ACAverageTokenImpl.

$$p_j = \begin{cases} i_j & , \text{if } t_j \in T_{non-gap} \\ i_{before,j} \cdot \left(1 - r_{gap,j}\right) + i_{after,j} \cdot r_{gap,j} & , \text{if } t_j = '-' \end{cases}$$

***Equation 7.1 The unaligned position of a sequence token* t_j within and outside of gaps**

t_j: *The token with the aligned index* j *in the aligned sequence*
$T_{non\text{-}gap}$: *The set of non-gap tokens used in the sequence*
p_j: *The unaligned position of the* t_j *(may be a non-integer values within gaps)*
i_j: *The absolute unaligned index of the* t_j *(starting with 1)*
$i_{before,j}$: *The absolute unaligned index of the first token before the gap (starting with 1) or 0 for leading gaps*
$i_{after,j}$: *The absolute unaligned index of the first token after the gap (starting with 1) or* l_s *+ 1 for trailing gaps*
$r_{gap,j}$: *The relative position of* t_j *within a gap (between 0 and 1) as calculated using Equation 7.2*
l_{gap}: *The length of the gap that contains the* t_j

The upper part of the equation is used to calculate the position of a non-gap token and is directly defined by its unaligned index.

The lower part is used to calculate the unaligned position of gap tokens. Since gap tokens by definition do not have unaligned indices but sequences with gaps should still contribute to the average position of an alignment column (see text), their position is calculated from the indices of the neighboring non-gap tokens ($i_{before,j}$ *and* $i_{after,j}$*). The gap border positions are weighted by the distances to* t_j *relative to the length of the gap, which is calculated using Equation 7.2.*

(All indices in this equation are assumed to start with one, which keeps it simpler in this case. In the implementation, all indices start with 0 for technical reasons and due to conventions. The resulting position p_j *lies between 0 and* l_s *+ 1 both here and in the implementation, since one is added to all indices in the implementation before calculating* p_j*.)*

$$r_{gap,j} = \frac{i_j - i_{before,j} - \frac{1}{2}}{l_{gap}}$$

Equation 7.2 The relative position within a gap used to weight the unaligned positions on its borders

$r_{gap,j}$: *The relative position of* t_j *within a gap (between 0 and 1)*
i_j: *The absolute unaligned index of the* t_j *(starting with 1)*
$i_{before,j}$: *The absolute unaligned index of the first token before the gap (starting with 1) or 0 for leading gaps*
l_{gap}: *The length of the gap that contains the* t_j

The position of a token t_j *within a gap of the length* l_{gap} *could be calculated by simply dividing its absolute index counted from the start of the gap by the length of that gap. If indices within a gap start with 1,* r_{gap} *would be 1 for the last position of the gap, which would mean that* p_j *calculated using Equation 7.1 for this position would be identical to* p_j *for the first token after the gap. To avoid that, one half is subtracted from all absolute positions in gap, making sure that all unaligned positions calculated for tokens within a gap lie between the positions of the tokens at the gap borders. (Note that in the implementation of the algorithm one half is added and not subtracted, since all absolute indices there start with 0 and not with 1.)*

7.2.2.2 Performing the initial superalignment

Now that average unaligned positions can be calculated for each column of each MSA to be compared as described in the last chapter, a superalignment can be performed based on this information. The simplest approach may be to align columns from the different MSAs in a way that each superalignment column contains only MSA columns with one average position that is strictly increasing over the columns from left to right. The result would be a superalignment where each column contains one or sometimes more MSA columns with the exact same average position and supergaps in all other MSAs. Although this way a very wide superalignment is produced that would only be of limited use as a visualization of alignment differences, it can be used as a starting point for further improvement steps. Algorithm 7.2 shows how this initial superalignment is created. The following definitions will be used in the further description of this superalignment approach. Superalignment average position lists according to Definition 7.4 differ from superalignment index lists according to Definition 7.3 by their elements, which are real numbers describing average unaligned positions instead of integers describing column indices.

Definition 7.4: A *superalignment average position list L* is a list of floating point values that has the length of its corresponding superalignment. There is one superalignment average position list for each MSAs that is aligned within a superalignment. Each entry is either an unaligned average position of a column in the corresponding MSA (not the superalignment) or NaN, indicating that this position of the superalignment contains a supergap inserted into the corresponding MSA.

Definition 7.5: An *average position superalignment S* is a matrix that combines a set of superalignment average position list $L_1..L_n$ with the length m (one for each compared MSAs of a superalignment), so that $S = \begin{pmatrix} L_1[1] & \cdots & L_1[m] \\ \vdots & \ddots & \vdots \\ L_n[1] & \cdots & L_n[m] \end{pmatrix}$.

Algorithm 7.2 Creating the initial superalignment using the average unaligned position approach

In every step, this algorithm searches for the next minimal average position in all input lists. The identified minimum is then removed from the start of all lists that contain it (usually only one or few) and then added to the end of the respective superalignment average position lists. (Note that minimal values will always be at the start of the input lists, since average positions are monotonically increasing. See text above.) In the same step, a supergap (indicated by "NaN") is added to all other output lists, so that the resulting initial superalignment will only contain one value (possibly the same multiple times) in one column that is aligned with supergaps and all superalignment average positions lists have the same length after each iteration.

See method `superalignPositions()` *at* http://r.bioinfweb.info/ACAverageTokenImpl *for the implementation of this algorithm.*

Input:

- Two or more lists of average unaligned positions (one for each MSA to be compared): $P_1..P_n$

Output:

- One superalignment average position list (Definition 7.4) for each compared MSA: $L_1..L_n$

```
1  nextPos = nextMin(P1..Pn);
2  while nextPos != NaN do    // while not all lists are empty
3     for i := 1..count(P1..Pn) do
4        if length(Pi) > 0 and Pi[0] = nextPos then
5           Add nextPos to the end of Li;
6           Remove first element of Pi;
7        else
8           Add NaN to the end of Li;    // NaN indicates a supergap at this position.
9        end if
10    end for
11    nextPos := nextMin(P1..Pn);
12 end while
13
14 function min := nextMin(P1..Pn);
15    min := ∞;
16    for P := P1..Pn do
17       if length(P) > 0 and P[0] < min then
18          min := P[0];
19       end if
20    end for
21    if min = ∞ then    // All lists are empty.
22       min := NaN;
23    end if
24 end function
```

7.2.2.3 Improving the superalignment

Improving a superalignment created as described in the previous chapter means to shorten it by removing supergaps, since only columns with the exact same average position are aligned together in such an initial superalignment. To achieve this, more MSA columns should be superaligned with each other by removing supergaps. Columns of the average position superalignment (Definition 7.5) produced from the output of by Algorithm 7.2 that have smaller differences between their average positions should be superaligned before those with larger differences. This way, subsequent merges of neighboring columns with more distant average indices may be blocked by previous merges but merging the columns with the lowest distance between their average positions is preferable. Blocked merges lead to supergaps that remain in the resulting improved superalignment.

To merge neighboring column pairs of the initial superalignment in the optimal order, the distances between the average positions of all neighboring column pairs are calculated. A multimap (a map that can hold more than one value per key) is created that uses these distances as its keys and the indices of the first columns of the neighboring column pair in the average position superalignment as the value. For each key (each encountered distance) the mapped set of values is kept sorted by the column index. Iterating over the key set of this map allows to traverse all pairs of neighboring superalignment columns in the order of their distance, starting with the closest pair. Algorithm 7.3 shows how shortening a superalignment can be achieved this way.

To avoid having to update the indices stored in the multimap during the execution of Algorithm 7.3, supergaps are not actually removed (in line 4 of the algorithm) but are marked for later removal. This way, no shifts in the superalignment average position lists happen. The way supergaps are marked for removal can be described with a text substitution system consisting of the rules listed in Table 7.1. When two neighboring columns of the initial average position superalignment are merged, one of these rules is applied to each row of the two columns. While the order in which column pairs are merged is defined by the multimap, the rule to be used is defined by the two tokens on the left side of each rule. For each row, exactly one rule will be applicable. (Note that one rule will always be applicable in each line of column pairs selected for merging, since rules exists for all cases, except for combinations of only “F” and “R”. Such combinations cannot occur in column pairs to be merged, since that is checked by the condition in line 3 of Algorithm 7.3.)

Rules 1 and 2 handle the simple case that one column contains an average position (indicated by `F`, which stands for “floating point value”) and the other a supergap marking (`-`). (Note that “-“ is used as the symbol for supergaps here, although this usually denotes a gap present in an input MSA and supergaps are indicated by “.”. Since gaps from initial alignments cannot occur here, as this step is dealing with average column positions, the two cases do not need to be differentiated.) In that case, marking the supergap for removal is the only option to shorten the superalignment, which is why the gap token is replaced by a removal marking (`R`). Rules 3 and 4 are similar, only that one column contains a removal marking from a previous neighboring merge instead of an average position.

Rule 5 applies if the cells of both columns to be merged contain a supergap marking in the same row. In this case, one cell could just arbitrarily chosen to be marked for removal, but this would not lead to an optimal superalignment. Putting a removal into the left column could possibly block a future merge between this column and its further left neighbor if that neighbor also has a position entry or a removal marking in the respective row. Putting it into the right column could hinder future merging with its further right neighbor for the same reason. If such conflicts exist with both neighbors, the algorithm should prefer to allow the merging of the neighboring column pairs with the lower average position difference. To postpone the decision, which supergap token to delete, two removal option markings (`O`) replace the two supergap tokens that indicate that a deletion may be performed at either position. (Comparing the distances of both possible future merges now to directly make a decision on which supergap to remove is not efficient, since possible blockings in all rows, even the ones not processed yet, need to be taken into account as well to make this decision. Additionally pairs of removal option markings may occur is distant columns due to the subsequent application of other rules described below, which would not be possible if a decision would be made directly.)

If two optional removal markings are found in the same row of two neighboring columns to be merged, these markings cannot be a pair (i.e., cannot result from a single application of one of rules 1-4). Instead the left token must be the right part of a pair located left of the current merge, while the right token is the left part of a pair located right of the current merge. Both token pairs must have been the

result of previous merges of neighboring column pairs. In such a case, rule 6 defines that both encountered optional removal markings should be replaced by non-optional removal markings. By doing this, the decision is also made for the left removal option pair to delete its right cell and for the right pair to delete its left cell. Besides these two removals, a third removal is necessary in order to merge the current column pair. Anyway, no further editing of cells is necessary, since the remaining two outer removal option tokens are still present left and right of the current columns. These now form a new token pair, since their former counterparts have been deleted. Due to this rule, removal option token pairs are not necessarily direct neighbors, but an unlimited number of non-optional removal markings may be present between them. (More than two removal markings may occur in between, if more than two directly neighboring removal option token pairs occur and rule 6 is applied between all of them. Rules 7 and 8 also insert tokens between a pair of optional removal markings as described below. Figure 7.2 illustrates such a situation.)

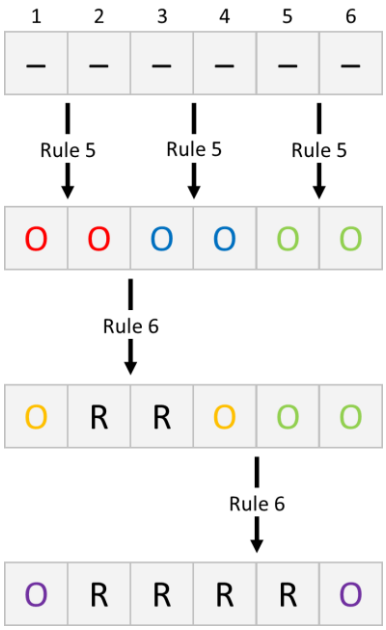


Figure 7.2 Example applications of rules 5 and 6 from Table 7.1

In this hypothetical situation shown for only one row, the column pairs (1, 2), (3, 4) and (5, 6) are merged in the first three steps (due to their low average position differences, not shown). Since all cells contain gaps, rule 5 is used and creates three pairs of optional removal markings (shown in red, blue and green). In a forth step columns 2 and 3 are merged and rule 6 applies. Therefore the outer optional removal markings of the red and blue pairs are transformed to a new pair (shown in orange) with two non-optional removal markings in between. The fifths step merges columns 4 and 5, which leads to another application of rule 6 and the transformation of the orange and green pairs into a new purple pair with 4 non-optional removal options in between.

Rules 7 and 8 deal with the situation that a supergap token is located next to an optional removal marking. The behavior here is similar to rule 6. In rule 7 the right removal option token of a pair located left of the current merge is moved one position to the right and a non-removal option token is inserted into the pair (in the left column of the current merge). This replacement of the initial removal option token by a non-optional one (and therefore the decision to definitely remove the supergap in the left column) is done since both possible column pairs containing this position now have been merged and there is no need to leave an option not to remove this column. Leaving the option on the right side instead makes sense, since it is not clear yet, whether another merge involving the right column of the current merge is still to come. Rule 8 does the same for the inverted situation.

Rules 9 and 10 define what to do when a combination of an optional and a non-optional removal token is encountered. There formulation is a bit more complex, since such a situation requires to edit a cell outside of the two columns that are currently merged. Rule 9 is applied to any situation where "`OR`" is encountered and rule 10 for "`RO`". The provided context describes the only environment of these token sequences that could have been previously produced by the application of any rule. As described above and shown in Figure 7.2, an unlimited number of non-optional removal tokens could be located in between a pair of removal option tokens, which is indicated by the expression `R*`. (Although a context is given in these rules, their application is not context-dependent. Rule 9 could, e.g., also be formulated as "`OR` → `RR` and replace the next `O` on the left by `R`" and the formulation of rule 10 could be made accordingly.) Concretely, rule 9 makes the decision on the neighboring removal option token pair to delete its gap in the left, since another (the current) merge on the right is taking place afterwards. Due to the current merge, the other removal option token is replaced by non-optional removal token as well, since this is the only position left, where another position could be deleted. Rule 10 does the same inversely.

Algorithm 7.3 Shortening an initial superalignment using the average unaligned position approach

This algorithm removes supergaps from an initial superalignment created by Algorithm 7.2 to superalign more columns from different alignments together. The superalignment of columns with less distant average indices is preferred over superaligning columns with more distant average indices. See method `shortenAlignment()` *at* http://r.bioinfweb.info/ACAverageTokenImpl *for the implementation of this algorithm.*

Input:

- One average position superalignment (according to Definition 7.5) produced from the output of Algorithm 7.2: *S*
- A multimap with all differences between the average indices of two neighboring columns of S as keys and a sorted set of indices of the first column of each pair as values: *M* (Note that S will never have different average index entries in one column according to Algorithm 7.2.)

Output:

- The edited input average position superalignment: *S*

```
1  for all keys d_i in M do    // Iterate over all column distances.
2    for all column indices i_j mapped by d_i in M do    // Iterate over column indices for this distance.
3      if no row in the combined columns (S[i_j], S[i_j + 1]) contains a position or remove marking in
           both cells then
4        Apply the rules defined in Table 7.1 to all lines in the combined columns (S[i_j], S[i_j + 1]);
5          // Mark current column pair to be merged.
6      end if
7    end for
8  end for
```

Rules 11 and 12 are equivalent to rules 9 and 10 with the only difference that here a pair of removal option tokens is located next to a numeric value (an average index indicated by `F`) instead of a non-optional removal token. The way these rules work is identical, since both average indices and non-removal tokens cannot be replaced by other tokens.

After Algorithm 7.3 terminated, possibly remaining pairs of optional removal markings are processed by replacing the left token of each pair with a removal marking and the right token with a supergap token. (See Algorithm 7.4.) This is the only time supergap tokens that have been marked for optional removal are restored. (All rules in Table 7.1 replace removal option tokens by non-optional ones, since they need to mark an additional position for deletion.)

In a further step all removal markings are removed from the lists, which results in a shortened superalignment with as many MSA columns superaligned as possible, while preferring to align columns with lower distances between their average positions, if topological conflicts occur. The formulation of the text substitution rules and the condition in line 3 of Algorithm 7.3 ensure that exactly one superalignment gap is removed from each row of the superalignment when two columns are merged. Therefore, the shortened average position lists still have equal lengths to form a valid superalignment.

Table 7.1 Rules of a text substitution system to merge two columns of an average position superalignment

The following rules can be used to process all lines in two neighboring columns to be merged. Supergaps are replaced by different removal markings, while average index values are never replaced. In every rule one additional supergap is marked for deletion, either by adding a removal mark or two removal options or by replacing two options by two removal marks.

Note that the order of token pairs (lines in column pairs) to which these rules are applied is determined by sorting of the keys and values of the multimap M *used in Algorithm 7.3 and is not necessarily starting with the first possible pair on the left as, e.g., in a text substitution system using a Markov strategy.*

If the two tokens on the left side of a production are encountered the respective rule can be applied. Some rules require a wider context, since additional neighboring columns need to be edited. In such cases the values in the columns to be merged are underlined. Anyway, the two tokens encountered in the columns to be merged are always sufficient to unambiguously determine which rule to be used. A wider context that is possibly given is not required for this decision and is only used to show that remove option tokens always occur in pairs and both need to be replaced, although only one is contained in the columns to be merged. The following alphabet is used:

"-": *Supergap*
"F": *General representation of any average position floating point value in a cell (This is formally not a token of an alphabet but a placeholder for all possible average indices. It is used here like a token to allow better readable rules.)*
"R": *Marking to remove this cell later*
"O": *Marking to optionally remove this cell later (This marking is always used in pairs.)*

The implementation of this substitution system can be found in the method `markTwoColumns()` *at* http://r.bioinfweb.info/ACAverageTokenImpl.

	Rule	Description
1	`-F` → `RF`	Mark a gap next to a position for deletion.
2	`F-` → `FR`	See rule 1.
3	`-R` → `RR`	Mark a gap next to a deletion mark for deletion.
4	`R-` → `RR`	See rule 3.
5	`--` → `OO`	If both columns contain a gap, either one of them could be removed.
6	`OO` → `RR`	Both neighboring regions already carry remove options. Together with the current merge this necessitates three merges. Two are marked by the newly inserted removal marks and the third is marked by the outer removal option marks present left and right if the current columns.
7	`O-` → `RO`	The left element of an optional remove token pair is moved further left.
8	`-O` → `OR`	The right element of an optional remove token pair is moved further right.
9	`O(R*)OR` → `R$1RR`	The left side of an optional remove token pair is selected and an additional supergap is deleted on the right for the current merge. ("$1" means leaving the unchanged capturing group from the left at this position.)
10	`RO(R*)O` → `RR$1R`	The right side of an optional remove token pair is selected and an additional supergap is deleted on the left for the current merge.
11	`O(R*)OF` → `R$1RF`	See rule 9.
12	`FO(R*)O` → `FR$1R`	See rule 10.

Algorithm 7.4 Processing remaining pairs of removal option tokens

After all markings to merge columns have been placed as outlined in Algorithm 7.3, some pairs of removal option tokens may still be in the lists, as they were not necessarily all replaced by applications of rules 5-12 in Table 7.1. This algorithm replaces the left token of each pair by a non-optional removal token and the right one by a super-gap. The side of which the actual deletion is performed can now be chosen arbitrarily, since no further merges of columns neighboring removal option tokens are still to come. As shown here, the assignment of the removal option tokens into pairs can easily be done by processing one by one and assuming the add ones as left and the even ones as right elements. (It is therefore not necessary to use different token for left and right elements of pairs.)

The implementation of this algorithm can be found in the method `processRemoveOptions()` *at* http://r.bioinfweb.info/ACAverageTokenImpl.

Input:

- One superalignment average position list (Definition 7.4) for each compared MSA: $L_1..L_n$

Output:

- The edited input superalignment average position lists: $L_1..L_n$

```
1  for L := L1..Ln do
2     isLeft := true;
3     for i := 1..length(L) do
4        if L[i] = "O" then
5           if isLeft then
6              L[i] := "R";    // Definitely remove at left option.
7           else
8              L[i] := NaN;    // Mark right option as supergap again.
9           end if
10          isLeft := !isLeft;
11       end if
12    end for
13 end for
```

7.2.2.4 Space and time complexity

To describe the size of the input we define the following:

- *a*: The number of alternative MSAs to be compared
- *n*: The average number of columns in each MSA
- *m*: The number of sequences in each MSA (All MSAs will have the same number of sequences, since they are all derived from the same dataset.)

7.2.2.4.1 Time complexity

First, the unaligned index for every position needs to be calculated, which can be done by iterating over all tokens of each sequence, while counting the number of encountered gaps. This needs to be done for each alignment, resulting in $O(a \cdot n \cdot m)$ steps.

To calculate the average position of one column, the unaligned positions of each *m* lines in that column need to be summed up. This needs to be done for all columns of all alignments, which will also need $O(a \cdot n \cdot m)$ steps.

To perform the initial superalignment as described in chapter 7.2.2.2, it is required to search for the next minimal average position at the beginning of all position lists. Since there is one list for each compared MSA and the search can be done in $O(a)$ time. The iteration in Algorithm 7.2 is repeated as long as unprocessed positions are available in any input list and, in the worst case, all positions from all columns of all MSAs differ, resulting in $a \cdot n$ iterations. Within each iteration, all lines are also checked for the current minimum, which also requires $O(a)$ iterations. Searching for the next minimum and

processing all lists can both be done together in $O(a)$ (since both steps happen sequentially) and this needs to be repeated for at most $a \cdot n$ iterations, resulting a time complexity of $O(a^2 \cdot n)$.

For shortening the initial superalignment (chapter 7.2.2.3), first all average position differences between neighboring columns need to be calculated and stored into the multimap described above. For each column the first cell with a non-gap value needs to be found, which would take at most $O(a)$. The multimap uses a binary search tree internally to sort and access its key set. Inserting into such a tree requires searching for the closest value, which scales linearly to the height of the tree. In the case of a height balanced search tree with $O(a \cdot n)$ elements, its height is $O(\log(a \cdot n))$, which is therefore also the complexity of inserting into it. (Ensuring the search tree to be balanced can be done, e.g., by using a red-black tree [208,209].) Since the previous step created at most $a \cdot n$ columns, the overall time complexity of this step is $O(a^2 \cdot n \cdot \log(a \cdot n))$.

The multimap will have one entry for each pair of neighboring columns, at most $a \cdot n - 1$. For each of these column pairs, the algorithm checks if the current pair can be combined, by checking all a lines in both columns. The marking of two columns to be merged is done in linear time, resulting in a maximum runtime of $O(a^2 \cdot n)$ for this step of the algorithm. (Random access to the map in $O(\log(a \cdot n))$ time is not required in this step, since the whole map is iterated by its order.)

To process the remaining removal option markings and to de facto remove marked cells, an iteration over all columns (at most $a \cdot n$) and all sequences (a) will be necessary, which again leads to maximal complexity of $O(a^2 \cdot n)$.

Combining all these sequentially executed steps, the time complexity of this superalignment approach is $O(a^2 \cdot n \cdot \log(a \cdot n) + a \cdot n \cdot m)$.

7.2.2.4.2 Space complexity

When calculating the average positions, one value will be stored for each column of each input MSA, which will need $O(a \cdot n)$ space. For each compared MSA, one superalignment average position list (see Definition 7.4) will be created when performing the initial superalignment. For the same reason described for the time complexity each of these a lists will have at most $a \cdot n$ entries, which leads to a space complexity of $O(a^2 \cdot n)$ for this step. Storing the differences between the average positions of all neighboring column pairs in the initial superalignment, will require is list with $O(a \cdot n)$ elements ($a \cdot n - 1$ column pairs, see above). The remaining steps of this approach work on the position lists created for the initial superalignment and will only edit and remove but never add cells. Therefore, they do not require additional space.

The highest space complexity of this approach is encountered when creating the initial superalignment so that $O(a^2 \cdot n)$ is also the space complexity for the whole algorithm. The amount of additional memory used will be significantly lower than the size of the input (which is $O(a \cdot n \cdot m)$) if there are significantly less MSAs to be compared than sequences present in each MSA, which is usually the case. If the memory used to store the input data as well is taken into account, the overall space complexity would hence be $O(a^2 \cdot n + a \cdot n \cdot m)$.

7.2.2.5 Example

Figure 7.3 shows how three alternative MSAs of the same dataset are superaligned using the average position approach. The sequences in the simple example dataset could be considered as an inverted repeat of “ACC” followed by a tandem duplication of “TGA”. *Alignment 1* is the shortest MSA of the five sequences and does not consider the first part to be an inverted repeat but assumes two indels to explain the different positions of the “A” in the different sequences. For the tandem repeat it assumes two homologue periods with a point mutation between “A” and “C” in one of them. *Alignment 2* on the other hand is the longest, interprets the first three nucleotides to belong to an inverted repeat,

and separates both orientations into different columns. (Note that aligning homologue nucleotides in inverted repeats is not directly possible in a column-based multiple sequence alignment and therefore both previous alternatives could be considered.) The right half of the sequences is grouped into three different tandem repeat periods considered homologue. *Alignment 3* in contrast considers the pattern “TCA” and “CCA” to be homologue.

The figure shows all steps described in this chapter to produce a superalignment, which are explained in detail in the figure caption. The resulting superalignment arranges corresponding tokens of the alternative MSAs closely together on average but does not necessarily maximize the number of corresponding tokens superaligned in the exact same superalignment column. Columns 10-12 in *Alignment 1* are an example, as they look like shifted by one to the right compared to the other two MSAs. For clarification, it should be noted that only sequences *B*, *C* and *E* contain tokens in columns 10-12 of *Alignment 1* that correspond to those in columns 9-11 in *Alignment 2* or 8-11 in *Alignment 3*. The other two sequences have their corresponding tokens further right in *Alignments 2* and *3*. Therefore, the algorithm positioned the tokens in column 10-12 of *Alignment 1* in between of their counterparts in the other two MSAs, since the average column positions match best this way. Minimizing the distance between all corresponding tokens on average is the main characteristic of this approach.

The next chapter describes an alternative approach offered by *AlignmentComparator* that focuses on correctly superaligning a maximal number of tokens instead of minimizing their distance on average.

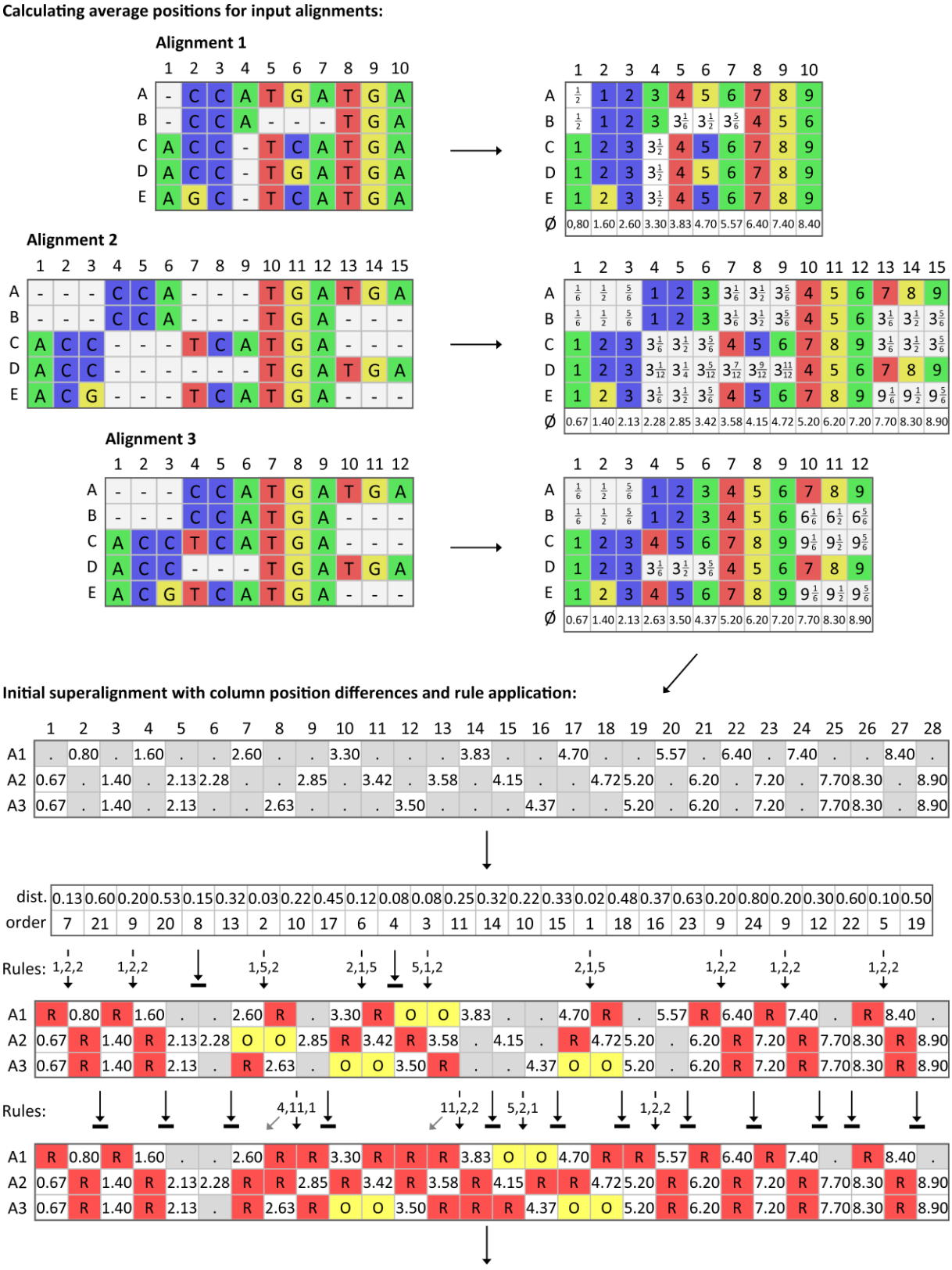


Figure 7.3 *(Continued on the next page.)*

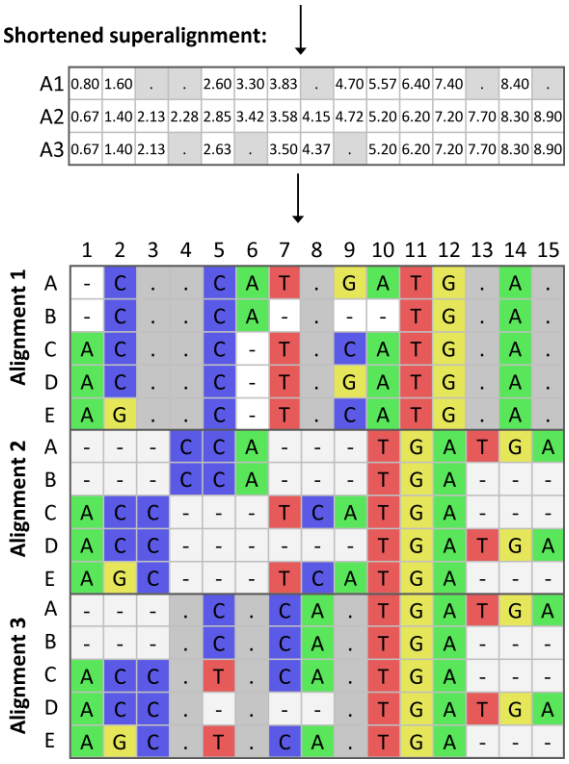


Figure 7.3 Example of a superalignment of three MSAs using the average position approach

Right of the three input MSAs the positions for all cells are calculated (using Equation 7.1) and the average positions of all columns are shown. Below that, the initial superalignment derived from the calculated positions is shown, as it would be produced by Algorithm 7.2.

The table below the initial superalignment contains the differences between the average positions of all neighboring columns in the first row and the order in which columns will be merged in the second row. As mentioned in chapter 7.2.2.3, column pairs with lower distances are merged first (and pairs with equal distances are merged from left to right). The same index in the order row indicates pairs with the exact same distance. (Some pairs with identical distance entries in the first row still have different order indices. Their values differ in a decimal place not shown here due to rounding.)

The bottom of the first part of the figure shows how the substitution rules from Table 7.1 are applied in the defined order. (See also Algorithm 7.3.) The first superalignment with removal markings shows the state after steps 1-9 have been applied and the second shows the state after all substitutions have been performed. The superalignment is shown in two states to illustrate the intermediate states before steps 10 and 11 overwrite previously inserted removal option tokens. The rules applied for the different rows in each step are shown on the respective arrow. Arrows without rule numbers ending on a block indicate that no rule is applied for this column pair, since previous neighboring merges block a further merge here. (This is checked in line 3 of Algorithm 7.3.)

The second part of the figure shows the shortened superalignment after all marked cells have been removed and underneath the actual superalignment that is the result of processing this example dataset using the average position approach.

7.2.3 Maximum sequence pair match approach

In contrast to the previously described average position algorithm that superaligns multiple MSAs in a way that the average shift between corresponding sequences in each alignment is minimized, the maximum sequence pair match algorithm superaligns columns with each other that contain a maximum of corresponding tokens in corresponding sequences. The average position approach would also consider columns of two MSAs *A* and *B* as optimal matches that do not share a single corresponding token because, e.g., half of the sequences in *B* is shifted to the right and the other half the same way to the left, compared to those in *A*. In contrast, the algorithm described here will always choose matching columns that share the maximum amount of corresponding tokens, even if the average of all sequence positions together does not match optimally.

7.2.3.1 Performing a superalignment between two other superalignments or MSAs

Superaligning two alternative MSAs of the same dataset by maximizing the number of superaligned corresponding tokens in all columns is an optimization problem that is very similar to the pairwise global alignment of two sequences [210]. A dynamic programming matrix (DP matrix) can be constructed using the number of aligned corresponding tokens between two columns as the score (instead of, e.g., the Levenshtein distance [211] or other distances used for pairwise sequence alignment). As in the previous chapter, two tokens in different compared MSAs are considered as corresponding if they are contained in two sequences with the same name and have the same unaligned index.

Note that the score definition used here does not include any form of gap penalty. Supergaps are inserted by the algorithm described below in order to achieve a higher number of superaligned corresponding tokens but do not contribute to the score directly. In contrast to standard pairwise sequence alignment, penalizing gaps is unnecessary in this approach, since corresponding tokens can be identified unambiguously by their unaligned index and there are no alternative alignments with a match for the same token, which could differ in the number of supergaps.

If two alignments *A* and *B* with the columns $a_1..a_n$ and $b_1..b_m$ should be superaligned, a score matrix *M* can be created, where each cell $M[i, j]$ contains the number of matching tokens between a_i and b_j. To calculate the optimal alignment another matrix *M'* can be calculated from *M* using Equation 7.3. The resulting matrix *M'* is a DP matrix similar to that used, e.g., in the Needleman Wunsch algorithm (without gap penalties and affine gap costs). As shown in Equation 7.3 the value of a cell in M' can either be calculated from its top, left, or top left neighbor and the optimal superalignment can be reconstructed by tracing back this way from the bottom right cell to the top left cell in the same way as in the Needleman Wunsch algorithm. Moving diagonally then means to superalign the two respective columns and moving vertically or horizontally means inserting a supergap into one of the MSAs.

$$M'[i,j] = max\begin{cases} M'[i-1,j-1] + M[i,j] \\ M'[i-1,j] \\ M'[i,j-1] \end{cases}$$

Equation 7.3 Calculating the cells of the DP matrix used in the maximum sequence pair match superalignment approach

M*: A matrix containing the score (number of corresponding tokens) for each possible pair of columns of two MSAs to be superaligned (See text for details.)*

M'*: The DP matrix containing the summed up scores allowing back tracking the path of the optimal superalignment (See text for details.)*

Algorithm 7.5 calculates the two-dimensional matrix *M*, from which *M'* can be calculated in the next step as described above. Instead of two MSAs, it takes two superalignments as its input, which are now denoted by *A* and *B*. (Note that a superalignment may also contain only one MSA as denoted in

Definition **7.2** on page 91.) These two superalignments are processed in the same way as just described for two MSAs with the extension that corresponding tokens are counted between all possible pairs of MSAs contained in *A* and *B* and supergaps are of course inserted into all MSAs of one superalignment. Chapter 7.2.3.4 and Figure 7.4 show the application of Algorithm 7.5 and Equation 7.3 on an example.

Algorithm 7.5 Calculating the initial score matrix for a superalignment step in the maximum sequence pair match approach

The algorithm shown here creates a score matrix M*, where each cell* M*[i][j]stores the number of matching tokens, if the* i*th column from the first the* j*th column from the second input superalignment would be positioned underneath each other.*

The implementation of this algorithm is the calculation of `scoreMatrix` *in the method* `calculateDirectionMatrix()` *at http://r.bioinfweb.info/ACMaxSeqPairImpl.*

Input:
- Two superalignments (according to
- Definition **7.2**): *A*, *B*

Output:
- A two-dimensional array containing the pairwise scored between all columns of *A* and *B*: *M*

```
1  Create M with noOfColumns(A) columns and noOfColumns(B) rows;
2  Set all cells of M to 0;

3  for c := 1.. noOfColumns(A) do    // Iterate over all columns of A.
4    for all MSAs A_i in A do
5      for all Sequences S_A,i,j in A_i do
6        if S_A,j[c] ∈ T_non−gap then    // If the token of S_j in column c is not a gap
7          i_unaligned := unalignedIndex(S_A,i,j[c]);    // Determine the unaligned index of S_A,J[c].
8          for all MSAs B_i in B do
9            Determine the sequence S_B,i,j in B that corresponds to S_A,i,j;
10           M[c][alignedIndex(S_B,i,j, i_unaligned)]++;    // Increase the score for the
11                 // match of the two columns containing the current token.
12         end for
13       end if
14     end for
15   end for
16 end for
```

7.2.3.2 Superaligning more than two MSAs

The approach described in the previous chapter (7.2.3.1) calculates the superalignment between two smaller superalignments and cannot directly calculate an alignment between more than two inputs. It could theoretically be modified to use an *a*-dimensional matrix to superalign *a* MSAs. The space and time complexity of such an algorithm to superalign *a* MSAs with *n* columns would though have a complexity of $O(n^a)$, just like it is for maximizing the sum of all pairs score for an MSA with *a* sequences of the length *n* using the same approach [212], which is NP complete [213]. Since that would be impractical for nearly all datasets, the progressive approach [204,205] was chosen that is widely used in MSA algorithms. Applying it to this problem means performing one pairwise superalignment (as described above) for each internal node of a guide tree with all MSAs to be compared attached to terminal nodes.

That guide tree is calculated using the neighbor joining method [214,215], which requires a distance matrix that specifies the distances between all possible pairs of input MSAs to be compared. For the purpose of this superalignment approach, the distance between two MSAs is defined in Equation 7.4 as the number of superaligned token pairs in a pairwise superalignment (as described in chapter 7.2.3.1) subtracted from the overall number of tokens in one MSA. (Note that the number of tokens – not including gaps – is identical for each MSA to be compared, since it is a requirement that these are derived from the same set of unaligned sequences. Therefore, the pairwise distances are comparable.)

$$d = t_{all} - t_{superaligned}$$

Equation 7.4 The distance between two MSAs

d: *The distance between two MSAs*
t_{all}: *The number of tokens in one MSA*
$t_{superaligned}$: *The number of superaligned token pairs between both MSAs*

Although other distance measures exist to compare MSAs, the measure defined here uses the same algorithm to calculate the distance that is also used to process the data along the guide tree. This means that pairs of MSAs that are similar according to this comparison approach will be superaligned first before MSAs that are more distant are processed.

When superaligning along a guide tree, all internal nodes that have at least one non-terminal node as a child will require to superalign an existing superalignment with another one or an MSA. For that reason, Algorithm 7.5 takes two superalignments instead of two MSAs as its input. (Note that Definition 7.2 allows a superalignment to consist of only one MSA, as is the case when performing the initial superalignments along the guide tree.)

Consequently, to create a superalignment of a set of more than two alternative MSAs of the same dataset using the maximum sequence pair match approach, the following steps are performed:

1. Calculate all pairwise distances between the MSAs to be compared
2. Infer a guide tree from these distances, which has a terminal node for each input MSA
3. Superalign the MSAs along the guide tree

For the case that only two MSAs are to be compared, it is of course sufficient to directly calculate a superalignment between them without the need for a guide tree.

7.2.3.3 Space and time complexity

We use again the symbols *a*, *n* and *m* for the number of MSAs, the number of columns per MSA and the number of sequences per MSA respectively. In addition, we define a_A and a_B as the number of MSAs contained in the input superalignments *A* and *B* respectively in Algorithm 7.5. (Note that a_A and a_B vary between different superalignments along the guide tree, but $a_A + a_B = a$ will always be true and *a* is constant.)

7.2.3.3.1 Time complexity

Algorithm 7.5 iterates over all *n* columns of the first input superalignment *A* in its outer loop and over all a_A MSAs and all *m* sequences in each MSA in the next two tested loops. If no gap is found at the current position, an additional nested loop iterates over all a_B MSAs in *B*. This algorithm therefore has a time complexity of at most $O(a_A \cdot a_B \cdot n \cdot m)$ for worst case with no gaps in the inputs. Since a_A and a_B are proportional to a, we can equivalently assume $O(a^2 \cdot n \cdot m)$.

For calculating the pairwise distances to infer a guide tree, superalignments between all possible pairs of MSAs need to be performed first. This splits again into calculating the score matrix and constructing a DP matrix from it. Algorithm 7.5 calculates the initial score matrix with the complexity described above, but in the case of superaligning two single MSAs a_A and a_B are 1 and the complexity of calculating a single score matrix is $O(n \cdot m)$. The DP matrix is calculated next which uses a constants amount of time per cell resulting in complexity of $O(n^2)$. (Note that reserving the memory for the initial score matrix would also have a complexity of $O(n^2)$.) To calculate a distance-based guide tree, all pairwise distances need to be calculated and number of possible pairs is given by Equation 7.5.

$$p = \sum_{i=1}^{a-1} i = a \cdot \frac{a-1}{2} \Rightarrow O(a^2)$$

Equation 7.5 The number of possible different pairs* p *in a set of* an *elements and the resulting complexity.

The overall time complexity for calculating a distance matrix will therefore be $O\left(a^2 \cdot (n^2 + n \cdot m)\right)$, while n^2 will be dominant over $n \cdot m$ if the input MSAs have more columns than sequences, which is the expected major use case. The complexity would then be $O(a^2 \cdot n^2)$. The neighbor joining tree can be calculated from the matrix in $O(a^3)$ time if the improvement of the original Saitou Nei algorithm [214] by Studier and Keppler [215] is used.

The next step is the combination of superalignments along the internal nodes of the binary guide tree. The complexity of a superalignment step on one internal node is that of Algorithm 7.5 denoted above. When performing superalignments along the guide tree, the number of supergaps and therefore the number of columns will increase when moving closer to the root. While *n* is the average number of columns in the single input MSAs, we define n_S as the average number of columns in the input superalignments of Algorithm 7.5. If we assume a constants growth of the number of supergaps in a superalignment with the number of added MSAs (as similarly observed for progressive pairwise multiple sequence alignment, e.g. in [207]), n_S would be proportional to $n + a$. The complexity added to the algorithm by additional encountered supergaps is $O(a_A \cdot a \cdot m) = O(a^2 \cdot m)$, since the number of supergap columns is proportional to *a* (replacing the *n* in the complexity of Algorithm 7.5 as denoted above) and the dependency from a_B does not apply, since the inner loop of Algorithm 7.5 is not executed on gaps. Therefore, the complexity of Algorithm 7.5 remains unchanged even when considering supergaps, since it would now be $O(a^2 \cdot n \cdot m + a^2 \cdot m) = O(a^2 \cdot n \cdot m)$. (Note that the complexity will usually even be slightly lower, since the initial input MSAs also contain gaps (not supergaps) at some positions.)

Gaps though effect the complexity of calculating the DP matrix from the score matrix subsequently, which is $O({n_S}^2)$ hard. Since ${n_S}^2 = (a+n)^2 = a^2 + 2an + n^2$, the complexity of calculating the DP matrix is $O(a^2 + n^2)$.

A superalignment needs to be performed on each internal node of the guide tree, which has *a* terminal nodes and therefore $a - 1$ internal nodes, as it is a binary tree. Since the number of superalignments is then $O(a)$, the overall time complexity for superaligning aligning along the guide tree is $O\left(a \cdot \left(a^2 \cdot n \cdot m + (a^2 + n^2)\right)\right) = O(a^3 \cdot n \cdot m + a^3 + a \cdot n^2) = O(a^3 \cdot n \cdot m + a \cdot n^2)$.

The overall time complexity for this algorithm is the combination of the described complexities for creating the distance matrix $O\left(a^2 \cdot (n^2 + n \cdot m)\right)$, inferring the neighbor joining tree $O(a^3)$ and superaligning along it $O(a^3 \cdot n \cdot m + a \cdot n^2)$, resulting in $O(a^3 \cdot n \cdot m + a^2 \cdot n^2)$.

In practice the complexity of n^2 in creating the distance matrix and performing the superalignment is most relevant, since the number of columns (*n*) is much higher than the number of compared MSAs (*a*) for most use cases and the number of sequences per MSA (*m*) is usually not significantly higher than the number of columns (*n*). A test of the current implementation of *AlignmentComparator* that superaligned ten alternative MSAs of a Bryophyte DNA dataset with 137 sequences and 25,000 resulting superalignment columns (3,000 to 10,000 columns in the input MSAs) used a little more than half of the time to calculate the distance matrix and nearly all the remaining smaller half for the superalignment along the guide tree. The results for tests with other datasets produced similar results. The time to calculate the neighbor joining tree itself did not contribute relevantly to the calculation time.

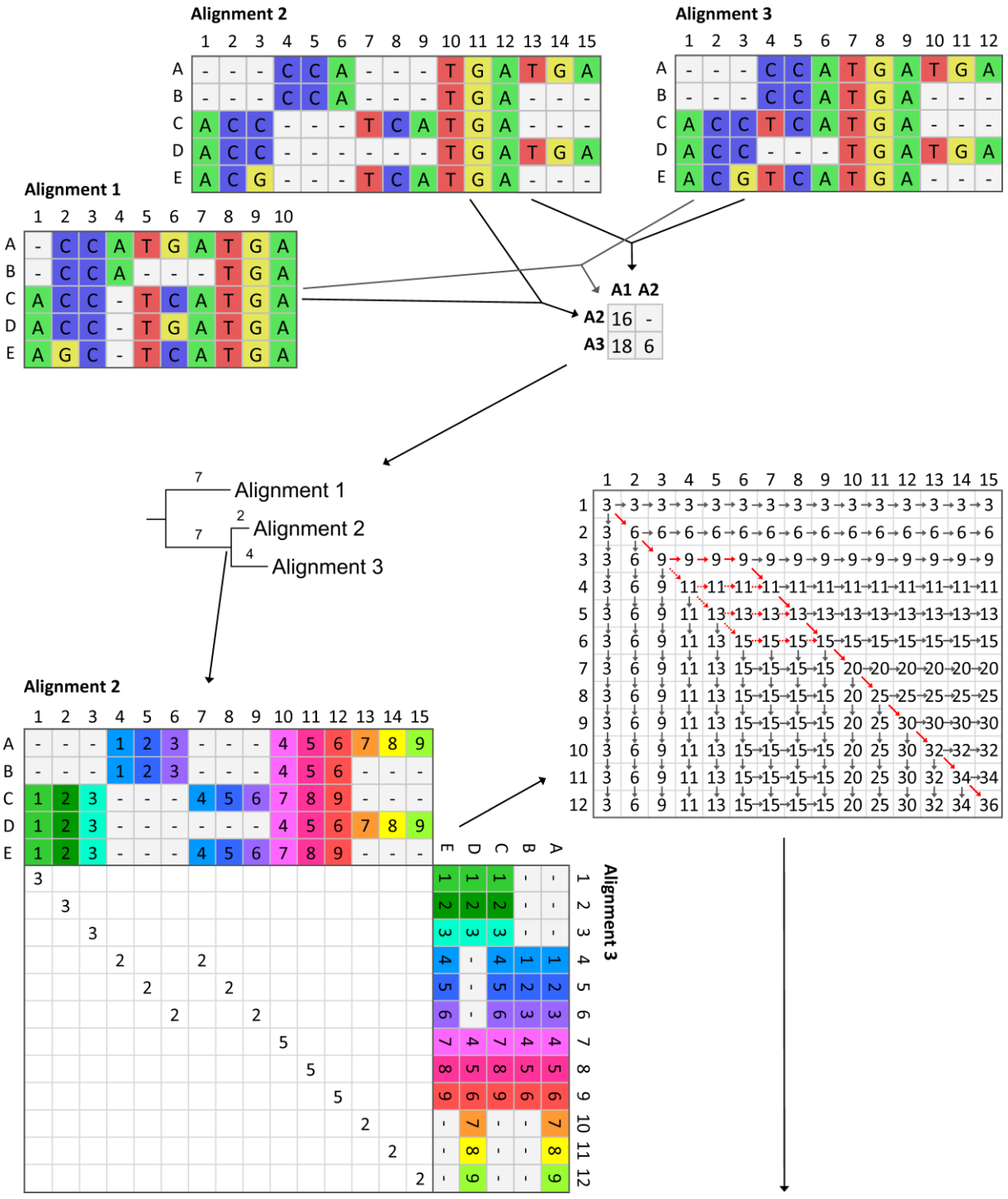


Figure 7.4 *(Continued on the next page.)*

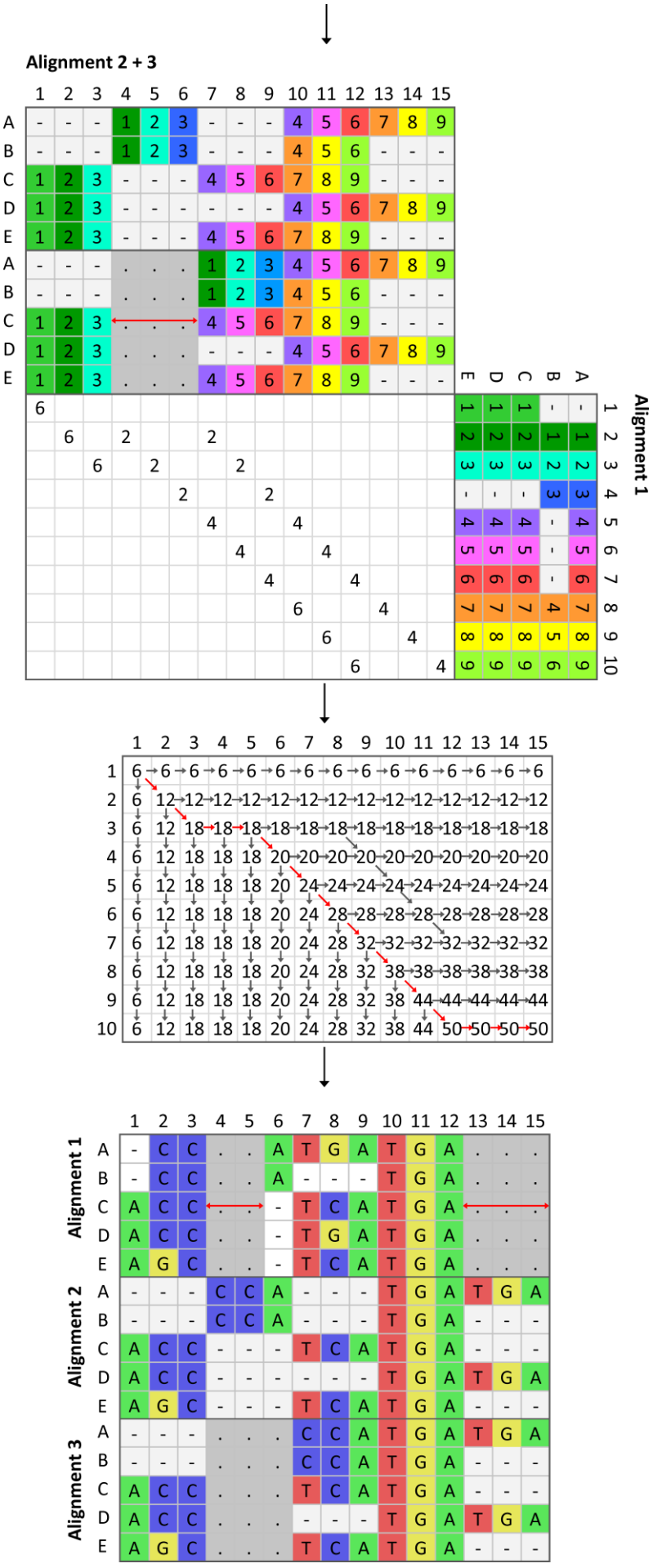


Figure 7.4 Example of a superalignment of three MSAs using the maximum sequence pair match approach

A distance matrix is calculated from all pairwise distances between the three input alignments according to Equation 7.4. The number of matching superaligned token pairs is determined by calculating the respective DP matrix, which is not shown here. The distance matrix is used to calculate a neighbor joining tree that is used as the guide tree.

According to the guide tree a superalignment between Alignment 2 *and* Alignment 3 *is performed first. The two alignments are shown with their unaligned indices (instead of the nucleotides) along the superalignment score matrix. Each column in* Alignment 3 *is highlighted in a different color and the corresponding tokens in* Alignment 2 *are highlighted in the same color. The calculation of the score matrix according to Algorithm 7.5 is illustrated by the colors. The vertically positioned MSA defines one color for each row of the score matrix and the score for each cell in one row can be read directly by the number of tokens in the same column with the same color in the horizontally positioned MSA. Cells with no score are left empty for clarity.*

In the next step the DP matrix is calculated from the score matrix according to Equation 7.3. The arrows indicate the direction in which values were calculated from values of neighboring cells. Red arrows indicate the optimal path through the matrix that is used to perform the superalignment, while arrows with dotted red lines indicate alternative optimal paths. New supergaps inserted due to non-diagonal steps in the DP matrix are also marked with red arrows in the resulting superalignments.

The superalignment resulting from Alignment 2 *and* 3 *is then superaligned with* Alignment 1 *according to the guide tree, which results in the superalignment at the end of this figure.*

7.2.3.3.2 Space complexity

A score or DP matrix used to perform a superalignment has n^2 cells for calculating the values of the distance matrix or to perform superalignments between two terminal nodes of the guide tree. n_S^2 cells are needed for performing a superalignment along the other internal nodes of a guide tree. All superalignments can be performed subsequently, so only one matrix needs to be stored at a time. The matrix at the root of the guide tree will be the biggest, since its two input superalignments contain the most supergaps and therefore the most columns. The space complexity for the matrices is therefore $O({n_S}^2)$, which is equal to $O(a^2 + n^2)$, as described for the time complexity above. Although not yet implemented in the current version of *AlignmentComparator*, only one columns of a DP matrix needs to be in memory at a time when applying the approach of Myers and Miller [216], while keeping the same time complexity. Algorithm 7.5 allows the same reduction for calculating the initial score matrix line by line and each line can already be converted to a line of the DP matrix prior to continuing the loop. This way, the space complexity of performing pairwise superalignments can be reduced to $O(n_S) = O(a + n)$.

The distance matrix for the neighbor joining tree needs $O(a^2)$ space. The tree nodes require $O(a)$ space.

Loading the input MSAs into memory will use $O(a \cdot n \cdot m)$ space, if no sequence compression, e.g. by locating redundancies is performed. The position of supergaps can be stored independently of the input MSAs, instead of storing of copy of all data in separate superalignment matrix. (See, e.g., Definition 7.3 on page 93.) If we again assume the number of inserted supergaps to be proportional to the number of compared MSAs, as described above, the complexity for storing the supergaps of inserted into each MSAs is $O(a^2)$.

This leads to a combined space complexity of $O(a^2 + a \cdot n \cdot m)$, where $O(a \cdot n \cdot m)$ is the dominant part for most uses cases, since the number of compared MSAs will usually be significantly lower than the number of columns per MSA. Therefore, the needed space for the maximum sequence pair approach scales nearly linearly with the size of the input data.

7.2.3.4 Example

Figure 7.4 shows how the three alternative MSAs described in chapter 7.2.2.5 (page 104) are superaligned using the maximum sequence pair match approach. As mentioned there, this hypothetical example dataset consists of a left part with a pattern that could be originated from an inversion event, while the right part could be the result of one or two tandem duplications.

To calculate the guide tree necessary for this approach, a distance matrix is calculated from the three alignments using Equation 7.4. To determine the number of matching token pairs, a superalignment DP matrix is calculated for all three possible MSA pairs in the same way as shown for superaligning along the guide tree in Figure 7.4, although the respective score and DP matrices are not shown in the figure. (Note that the shown superalignment between *Alignment 2* and *3* is the same that would be used to calculate the distance between these two alignments. The additionally performed superalignments between *Alignment 1* and *2* and between *1* and *3* are not shown in the figure.)

Due to the guide tree *Alignment 2* and *3* are superaligned first, resulting in four alternative optimal paths through the DP matrix. For the next superalignment step, the top-most alternative is chosen arbitrarily. The alternative paths are explained by the equal scores in the score matrix in columns 4 to 9.

The second superalignment is performed between *Alignment 1* and the superalignment from the previous step. Token pairs are matched between *Alignment 1* and the superaligned version of *Alignment*

2 and between *Alignment 1* and the superaligned version of *Alignment 3* independently according to Algorithm 7.5 and the results are added up.

The resulting superalignment at the end of the figure shows that supergaps have been inserted into *Alignment 1* and *3* in the left part to deal with the different alignment of the inverted repeat region and another supergap is present on the right of *Alignment 1* resulting from the different alignment of the inverted repeat region. The differences in that region could be superaligned between *Alignment 2* and *3* without the need to insert supergaps. The result visualizes the differences between the alternative input MSAs by superaligning a maximal number of corresponding tokens underneath each other, which is the main characteristic of the maximum sequence pair match approach. (See chapter 7.4.3 on page 119 for a comparison between the superalignment approaches offered by *AlignmentComparator*.)

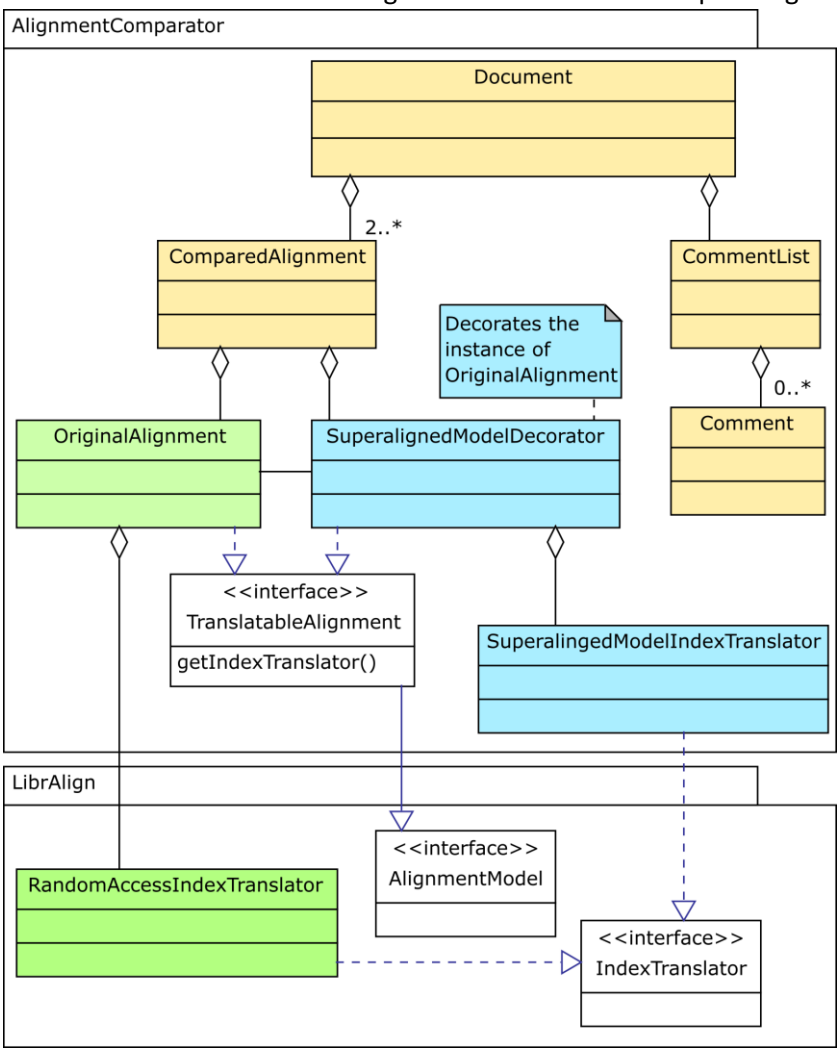


***Figure 7.5 UML class diagram of the data model of* AlignmentComparator**

General model classes are shown in orange. Classes dealing with the original input alignment are shown in green, while classes dealing with superalignment information are shown in blue.

One comparison of multiple MSAs is modeled by one instance of the class `Document`*, which is composed of one instance of* `ComparedAlignment` *for each compared MSA and an instance of* `CommentList` *that stores comments attached to the comparison by the user. Each instance of* `ComparedAlignment` *is composed of an* `OriginalAlignment` *that stores one input MSA using an instance of a* LibrAlign *model class and an index translator to translate between aligned and unaligned indices. Additionally, an instance of* `SuperalignedModelDecorator` *stores the supergaps inserted during the comparison. It does not store a copy of the whole alignment but decorates* `OriginalAlignment` *and extends its data by supergap tokens at respective positions to save memory. The decorator uses* `SuperalignedModelIndexTranslator` *as a special index translator, which relies in the data provided by the index translator of* `OriginalAlignment` *but additionally takes the supergap positions into account and therefore translates between superaligned and unaligned indices. Both* `OriginalAlignment` *and* `SuperalignedModelTranslator` *are accessible using the shared interface* `TranslatableAlignment`*, which inherits from* `AlignmentModel` *of* LibrAlign.

7.3 Implementation

AlignmentComparator is implemented in *Java* and provides a *Swing* GUI. It makes use of the functionality provided by *JPhyloIO* (chapter 2, page 33) and *LibrAlign* (chapter 3, page 46) as described in the following subchapters.

7.3.1 Data model and user interface

As shown in Figure 7.5, `Document` and the classes it is composed of store the input MSAs and the comparison results, as well as user comments attached to certain superalignment columns. (See figure description for details.) The shared interface `TranslatableAlignment`

allows easy access to both the MSAs and their superaligned versions including an `IndexTranslator` to determine unaligned indices that also considers supergaps. During the superalignment along the guide tree as done by the maximum sequence pair match approach (see chapter 7.2.3 and Algorithm 7.5, page 109) both raw input alignments and previously created superaligned versions can accessed as an instance of `TranslableAlignment` without the need for modeling that difference in the superalignment algorithm implementation. `ComparedAlignmentModel` decorates (see decorator pattern in [88]) an instance of the *LibrAlign* class `PackedAlignmentModel` that allows efficient storage of sequence tokens reducing the memory footprint of *AlignmentComparator*. (See also chapter 3.2.3 on page 48.)

The data stored in the described model classes is displayed using the GUI components provided by *LibrAlign* (see chapter 3.2.1 on page 47 and chapter 3.3.1 on page 49), which act as the respective views in the model view controller paradigm [106]. A `MultipleAlignmentsContainer` that allows displaying multiple MSAs underneath each other is used and contains an instance of `AlignmentArea` for each compared MSA. `AlignmentArea`s display the data provided by `SuperalignedModelDecorator` that includes supergaps.

In addition, *AlignmentComparator* currently displays two kinds of metadata using custom *LibrAlign* data areas. (See chapter 3.3.1.) One is located in a separate `AlignmentArea` below all compared alignments and allows displaying and editing user comments attached to certain columns of the comparison. The other one is attached at the bottom of each compared MSA if the average position superalignment approach is used and displays the average unaligned position for each column.

7.3.2 I/O and NeXML metadata

Reading input MSAs is done using *JPhyloIO* (chapter 2.2.1, page 35) that provides a module that allows to directly create instances of *LibrAlign* model classes. These are then used with the decorators of the of *AlignmentComparator*'s data model as described above.

Reading and writing completed comparisons is also done using *JPhyloIO*. *NeXML* [35] was selected as the main format of *AlignmentComparator* to save its results. The format allows storing several MSAs within one document including metadata attached using RDF attributes. The current format version of *AlignmentComparator* is also written as document metadata and is used to distinguish between *NeXML* files that store an alignment comparison with all necessary metadata and general *NeXML* files that cannot be interpreted as a comparison document. This technique is similar to how *PhyDE 2* differentiates between its own and general *NeXML* files as described in chapter 9.2 (page 138). (See also chapter 7.4.2 and Figure 7.7.)

To read and write comparison documents, *AlignmentComparator* makes use of the flexible architecture of *JPhyloIO* that allows directly reading and writing from *LibrAlign* MSA model classes without the need for any additional implementations in the application, while still providing full access to application-specific attached metadata from within the application code.

7.4 Results and discussion

7.4.1 Features and user interface

AlignmentComparator allows to visually compare alternative multiple sequence alignments of the same datasets using three different compare algorithms described in chapter 7.2 (page 91). It is open source and available under the terms of version 3 of the *GNU General Public License*. Due to its implementation in *Java*, it is executable on all operating systems with a *Java* runtime environment available (e.g. *openJDK*). The input alignments and the compare algorithm to be applied can be selected in a graphical user interface. The resulting superalignment is displayed in the main window of *AlignmentComparator*, where each input alignment with the inserted supergaps is displayed underneath

each other. The compared MSAs can be scrolled together horizontally and vertically, while *AlignmentComparator* ensures the same sequences are visible in all MSAs. (See Figure 7.6.) Furthermore, the whole superalignment can be freely zoomed using the mouse wheel.

Columns of the superalignment can be selected with the mouse or the keyboard in one MSA and corresponding tokens in all others are automatically highlighted. In addition to the comparison information provided by the positioning of MSA columns in the superalignment itself, this feature additionally simplifies to inspect the horizontal distribution of corresponding tokens among all compared MSAs. (See Figure 7.6 and Figure 7.8.)

After having performed a comparison, the user can attach comments to sets of neighboring columns of the superalignment, e.g., to annotate details on the differences between the compared MSAs. All operations performed by the user can be undone and redone from options in the edit menu.

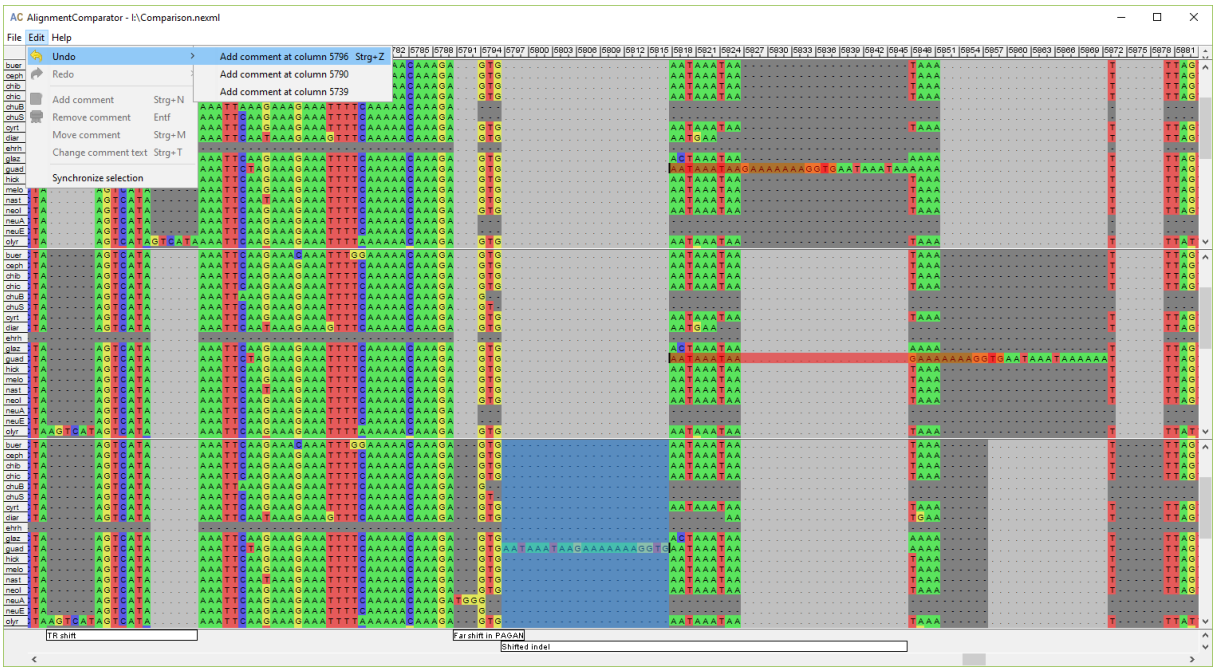


***Figure 7.6 Screenshot of a comparison of three MSAs opened in* AlignmentComparator**

A superalignment of three alternative MSAs of the same DNA dataset was performed using the maximum sequence pair match approach and the result is displayed in the main window of AlignmentComparator. *Gaps already present in the input MSA are shown in dark gray, while supergaps are in light gray. A set of columns in the third (lowest) MSA is selected (in blue) and the corresponding tokens in the other two MSAs are highlighted (in red). Some differences between the MSAs have been annotated by the user with the comments displayed at the bottom of the window.*

7.4.2 Supported formats and storage of comparison results

Since AlignmentComparator is based on *JPhyloIO*, it can currently read MSAs in *FASTA*, *Phylip*, R*elaxed Phylip*, *Nexus*, *NeXML*, *MEGA* and *PDE* format. (See chapter 2.2.3 on page 38 for further details.) It will automatically read all additional formats possibly supported by *JPhyloIO* in the future.

```
<?xml version="1.0" ?>
<nexml xmlns="http://www.nexml.org/2009" version="0.9"
    generator="AlignmentComparator 1.0.0-250 beta using JPhyloIO 0.5.2-1609 alpha"
    xmlns:nex="http://www.nexml.org/2009"
    xmlns:ac="http://bioinfweb.info/xmlns/AlignmentComparator/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <meta id="meta1" xsi:type="nex:LiteralMeta" property="ac:formatVersion"
      datatype="xsd:token">1.0</meta>
  <meta id="meta2" xsi:type="nex:LiteralMeta"
      property="ac:applicationVersion" datatype="xsd:token">1.0.0-250 beta</meta>
  <meta id="meta3" xsi:type="nex:LiteralMeta"
      property="ac:tokenType" datatype="xsd:token">NUCLEOTIDE</meta>
  <meta id="meta4" xsi:type="nex:ResourceMeta" rel="ac:comments">
    <meta id="meta5" xsi:type="nex:ResourceMeta" rel="ac:comment">
      <meta id="meta6" xsi:type="nex:LiteralMeta" property="ac:text"
          datatype="xsd:string">Inversion</meta>
      <meta id="meta7" xsi:type="nex:LiteralMeta" property="ac:start"
          datatype="xsd:int">0</meta>
      <meta id="meta8" xsi:type="nex:LiteralMeta" property="ac:end"
          datatype="xsd:int">5</meta>
    </meta>
    <meta id="meta9" xsi:type="nex:ResourceMeta" rel="ac:comment">
      <meta id="meta10" xsi:type="nex:LiteralMeta" property="ac:text"
          datatype="xsd:string">Tandem repeat</meta>
        ...
      </meta>
    </meta>
    ...
  </meta>
  ...
  <characters id="c0_" about="#c0_" label="Alignment 1"
      otus="undefinedOTUs4" xsi:type="nex:StandardSeqs">
    <meta id="c0_indices" xsi:type="nex:LiteralMeta"
        property="ac:unalignedIndices" datatype="xsd:string">
        0 1 2 - - 3 4 5 6 7 8 9 - - -</meta>
    ...
    <matrix>
      <row id="c0_id0" about="#c0_id0" label="A" otu="undefinedOTU3">
        <seq>0 2 2 1 4 3 1 4 3 1</seq>
      </row>
      ...
    </matrix>
  </characters>
  <characters id="c1_" about="#c1_" label="Alignment 2"
      otus="undefinedOTUs4" xsi:type="nex:StandardSeqs">
    ...
  </characters>
</nexml>
```

***Figure 7.7 Example of a* NeXML *document containing a comparison result of* AlignmentComparator**

This basic example NeXML *document contains the comparison of two MSAs "Alignment 1" and "Alignment 2" with one* `characters` *tag for each of them. Especially relevant parts are highlighted in bold. In the root tag the namespace* `ac` *is declared that contains all necessary* RDF *predicates to attach* AlignmentComparator*-specific metadata. On the document level, this includes the format and application versions, which are used to determine whether a* NeXML *file can be interpreted as a comparison document and a set of user comments. The latter is modeled by a resource metadata element for the whole comment list using the predicate* `ac:comments`*. Nested within this element are one or more resource metadata element modeling a single comment, which in turn contain three literal metadata elements for the comment text, the start, and the end column. A metadata element is attached to each MSAs using the predicate* `ac:unalignedIndices`*, which contains a string describing the*

Figure 7.7 *(Continued from previous page.)*

positions of the supergaps, which are not included in the actual alignment data.

(For technical reasons, the current version of AlignmentComparator stores all compared MSAs using the NeXML type `StandardSeqs`*, instead of using the respective DNA, RNA or amino acid types, which is why integers are contained in the* `seq` *tags. There translation declarations to nucleotides are omitted in this example. Future versions are planned to use the most appropriate* NeXML *token type whenever necessary to further increase interoperability. See chapter 7.4.5.4 on page 125 for further details.)*

Comparison results, including attached comments can be saved to *NeXML* and loaded again later. (See also chapter 7.3.2.) When a comparison document is saved, each compared MSA is written separately into the same *NeXML* file and the positions of the superalignment gaps are stored as metadata attached to each MSA. In addition, general metadata and the comments attached to certain columns by the user are written to the document level, which is also supported by *NeXML*. An ontology has been defined to attach the metadata necessary for a comparison Figure 7.7 shows an example of a comparison stored in *NeXML* using this ontology.

7.4.3 Differences between the comparison algorithms

As described in chapter 7.2, *AlignmentComparator* currently offers three alternative approaches to compare different MSAs of the same or slightly different dataset. The profile alignment approach (chapter 7.2.1, page 93) is currently only able to compare two MSAs, while the other two can compare any number of alternative MSAs. It could though be extended to compare more than two MSAs at a time by using the same guide tree-based algorithm as the maximum sequence pair match approach (chapter 7.2.3, page 107) currently does. (See also chapter 7.4.5.1, page 122.) Besides the number of currently supported MSAs, the main difference between the profile alignment approach and the other two is that the profile alignment uses information from the actual sequence tokens (e.g. nucleotides or amino acids) to determine which columns to superalign, while the two alternative approaches use the unaligned index of each token instead. As a result, the profile alignment approach is also able to compare MSAs that do not contain the exact same unaligned sequences, e.g., due to different hotspot deletion. (Note that chapter 7.4.5.1 describes possible extensions of the average position and the maximum sequence pair match approaches to also support differences in the unaligned sequence of compared MSAs.) While the profile alignment approach has a benefit on datasets with different unaligned sequences, considering actual tokens instead of their unaligned indices has disadvantages when dealing with repetitive sequences that contain simple sequence repeats, more complex repetitive patterns, or even poly-A regions. In such cases the profile alignment approach is not necessarily able to distinguish between different repeats of the same pattern (e.g. the different periods of the tandem repeat in the example dataset used in chapters 7.2.2.5 and 7.2.3.4), while approaches relying in the unaligned index to identify corresponding tokens are not affected by sequence repeats at all.

As already briefly mentioned in the previous chapters, the main difference between the average position (chapter 7.2.2, page 94) and the maximum sequence pair match approaches are there optimization strategies. The average position approach focuses on superaligning in a way that fits best for all tokens on average, which may result in a superalignment that positions corresponding tokens closely together but does not necessarily align any of these tokens in the exact same superalignment column. In contrast to that, the maximum sequence pair match approach focuses on a selected subgroup of tokens that can be optimally aligned and accepts a relatively distant superalignment position for the remaining sequences. The subset of tokens to be optimally aligned is selected in a way that its size is maximal.

Figure 7.8 shows an example for different superalignments using these two approaches. Both strategies have their advantage and disadvantage in best visualizing the differences between alternative MSAs. On the one hand, the supergaps inserted by the maximum sequence pair match approach may

be more meaningful when focusing on the selected (largest) subgroup of sequences with the respective superaligned tokens (see description of Figure 7.8). On the other hand, the user may more easily think that the tokens of the remaining sequences *A* and *D* in the superalignment columns 10-12 of *Alignment 1* in Figure 7.8 are also equivalent, which they are not. The average position superalignment, in contrast, does not visualize any direct relation between the tokens in the columns 11-14 of *Alignment 1* by superaligning them with corresponding tokens from the other alignment in the same column. Instead, it visualizes the position conflict between the two groups of sequences, by positioning the tokens in *Alignment 1* between their counterparts in *Alignment 2* and *3*. (Both visualization strategies are complemented by the highlighting of corresponding tokens selected in one alignment in all others, as described in chapter 7.4.1.)

The average position approach has a time complexity of $O(n \cdot \log(n))$regarding the average number of columns of the input MSAs (see chapter 7.2.2.4, page 103), while the maximum sequence pair match approach has a quadratic complexity , which makes the average position approach significantly faster when comparing large MSAs. Although both approaches have advantages in visualizing alignment differences, the maximum sequence pair match approach could be considered to produce the more intuitive results, but this comes at the cost of a longer runtime.

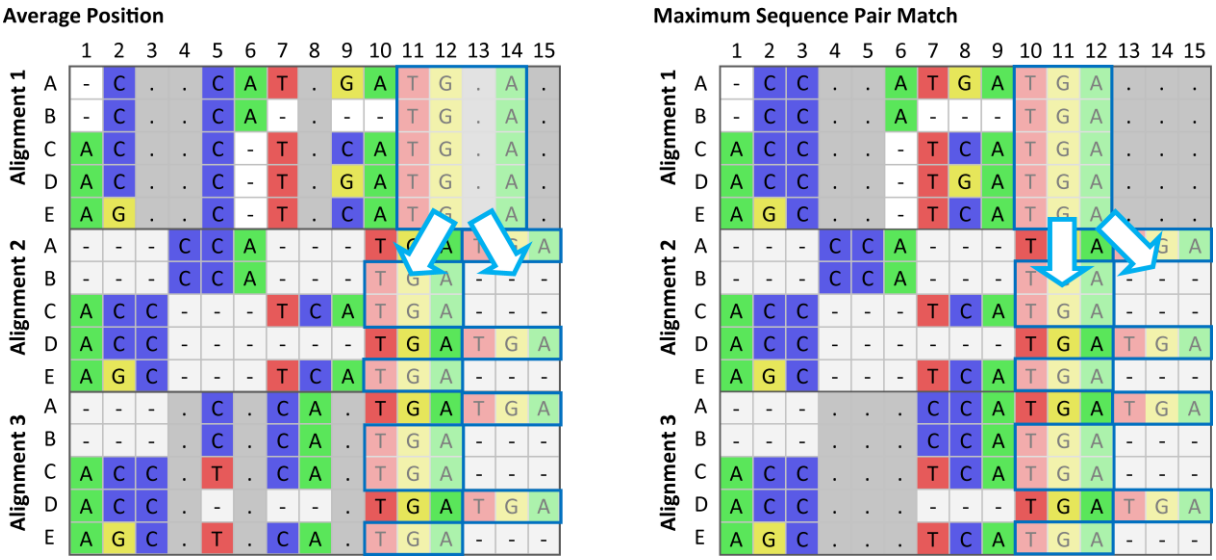


Figure 7.8 Comparison between superalignments produced by the average position and maximum sequence pair match approaches

Three alternative MSAs of the same example dataset are superaligned using the two approaches as shown in Figure 7.3 (page 107) and Figure 7.4 (page 113). The last three columns of Alignment 1 *and their corresponding tokens in the other two alignments are highlighted.* Alignment 2 *and* 3 *use the same alignment of these nucleotides, which is different from* Alignment 1*. The corresponding tokens in the sequences* B*,* C *and* E *are positioned in other columns then those in the sequences* A *and* D*. As indicated by the arrows, the average position approach superaligned the highlighted tokens in* Alignment 1 *in between of their corresponding counterparts in* Alignment 2 *and* 3 *and therefore minimizes their distance on average. In contrast, the maximum sequence pair match approach selects the sequence set of* B*,* C *and* E *in* Alignment 2 *and* 3 *and superaligns them in the same column as their counterparts in* Alignment 1*. As a result, the distance between corresponding tokens of the sequences* A *and* D *is higher than in the average position superalignment. The maximum sequence pair match approach chose the largest subset of sequences (*B*,* C *and* E*) which could be optimally superaligned and accepted a larger distance for the tokens in the other sequences. (The same principle applies for all other aligned tokens, which are not highlighted in this figure.)*

***Table 7.2 Comparison of the alternative superalignment approaches offered by* AlignmentComparator**

*The symbols used for the space and time complexity are as described in chapter 7.2.2.4 (page 103). (*a *is the number of compared MSAs,* n *the number of columns and* m *the number of sequences per MSA.) There are no complexities given for the profile approach, since it currently does not allow comparing the same number of MSAs. If it would be extended to support an unlimited number of MSAs (see chapter 7.4.5.1, page 122), its complexity would be similar to the maximum sequence pair match approach.*

Algorithm	**Strategy**	**Space complexity**	**Time complexity**	**No. of MSAs**
Profile	Token-based	-	-	2
Average position	Unaligned position-based, minimal average distance	$O(a^2 \cdot n + a \cdot n \cdot m)$	$O(a \cdot n \cdot m + a^2 \cdot n \cdot \log(a \cdot n))$	∞
Maximum sequence pair match	Unaligned position based, best superalignment for a maximal subset of sequences	$O(a^3 \cdot n \cdot m + a^2 \cdot n^2)$	$O(a^3 \cdot n \cdot m + a^2 \cdot n^2)$	∞

7.4.4 Comparison to other software

To compare alternative MSAs, software is available that calculates a score to quantify how different two MSAs are. Such software is useful for getting a basic overview on how a set of alternative MSAs compare to each, especially for many alternatives. It will not provide any detail on where and how compared MSAs differ and is therefore complementary but not an alternative to the functionality provided by *AlignmentComparator*. Examples are the implementations of the *Sum-of-Pairs* (*SP*) or the *Total Column* (*TC*) scores, e.g., in *BAliBASE* [217], the *Cline Shift Score* [218] or the different scores calculated by *MetAl* [219].

AlignStat [220] allows to compare two alternative MSAs of the same dataset and therefore calculates a set of similarity and dissimilarity matrices, among which is also one that is very similar to the score matrix calculated by Algorithm 7.5 (page 109) as part of the maximum sequence pair match comparison approach (chapter 7.2.3.1). In contrast, to *AlignmentComparator*, no superalignment is reconstructed based on this matrix but the content is displayed directly as a heat map that provides information on how similar which columns of both compared MSAs are. This is a different visualization approach then intended with *AlignmentComparator* and is limited to comparing two MSAs. *AlignStat* is available as an *R* package (which is of course accessible only for users, who are able to program) and a webserver-based version, which is limited to compare two MSAs with at most 1000 columns. *AlignmentComparator* does not require any programming skills and is able to compare much larger MSAs. The overlap between the functionality provided by *AlignStat* and *AlignmentComparator* is minor and both could complement each other. Some of the information *AlignStat* visualizes could even be displayed attached to the superalignment of *AlignmentComparator* in future versions to combine the information provided by both pieces of software. (See 7.4.5.3 below for details.)

MSA Comparator from the *SuiteMSA* tool collection [221] allows to visually compare alternative multiple sequence alignments by highlighting the position of corresponding characters in one alignment that are selected in another. No superalignment is performed, which can make the comparison less clear, depending on the way the MSAs differ. Unlike in *AlignmentComparator*, the comparison is limited to two MSAs. *Pixel Plot* is another tool from the *SuiteMSA* collection that allows to compare more than two MSAs at a time, but each nucleotide or amino acid is displayed only as a single pixel without the possibility to zoom in. If an area in one MSA is selected, the respective tokens in the other MSAs are colored differently. This way differences between MSAs can be visualized on broad scale, but *Pixel Plot* does not allow to inspect single nucleotide or amino acid patterns in areas that differ that may have led to the different alignment of an area, as *AlignmentComparator* does. *Pixel Plot* neither does perform a superalignment between MSAs.

Although mainly aimed at comparisons at a genome scale, *SinicView* [222] switches to a detail mode when setting the visual area to 100 columns of below. This mode displays multiple alternative MSAs that are aligned to each other and therefore is (to the best of our knowledge) the only other available software that performs some kind of superalignment. The superalignment is though only based on a single sequence, which can be selected by the user, and the application does not offer superalignment approaches that take all sequences into account to produce a globally optimal superalignment, as *AlignmentComparator* does. Due to the different focus, the GUI of *SinicView* is also limited to displaying 100 columns and 8 sequences from each MSA at the same time and does not provide any zooming, sequence coloring or highlighting of corresponding tokens. Furthermore, the program crashes on many input MSAs and cannot compare them.

VerAlign [223] is a web server that allows to compare two MSAs by differently coloring all columns of one MSAs and highlighting corresponding tokens in the other in the same color. Unlike *AlignmentComparator*, the comparison is again limited to two MSAs and no superalignment is performed. The webserver returns a HTML page containing all data from both compared MSAs as tables, which leads to significant performance problems in both loading and displaying time when the compared MSAs have more than a few thousand columns, which is not an issue when in *AlignmentComparator*.

AltAVisT [224] was a web server able to highlight corresponding nucleotides or amino acids between two MSA, similar to *MSA Comparator* from *SuiteMSA*, but the service has been suspended and the software is therefore no longer usable.

Many of the available programs for comparing alternative MSAs have a different focus then *AlignmentComparator* and can be useful in combination with it, rather than as an alternative. Most of the few applications available for detailed MSA comparison on the nucleotide or amino acid level are limited to two MSAs at a time and visualize differences only by highlighting corresponding tokens and not additionally by a superalignment. *SinicView* is the only other software that also allows to superalign alternative MSAs, but only takes a single sequence into account and therefore does not offer functionality that is comparable to the superalignment methods offered by *AlignmentComparator*. None of the other available applications support a comparable variety of input formats as *AlignmentComparator* with *JPhyloIO* or saves its results into generally accepted standard such as *NeXML*.

7.4.5 Future development

7.4.5.1 Extension and improvement of the comparison approaches

In all three currently available comparison approaches (described in chapter 7.2) each sequence of the compared MSAs is considered for creating a superalignment. For some use cases, it can be beneficial if a user could select a set of sequences that are especially relevant for the current comparison and exclude others. Such a set could be selected prior to a comparison and the algorithms would only take these sequences into account. Still, all sequences would be displayed but the comparison would only be based on the currently selected subset, which will influence the resulting superalignment in many cases. All three algorithm implementations could easily be adjusted to accept a respective set parameter. Such a feature would be useful for more fine-grained analyses of alternative MSAs. A user could better visualize how different sequences influence the differences between MSAs by creating separate superalignments based in different sequences sets than it would be possible with a single superalignment based on all sequences. The question which sequences and therefore which of the contained nucleotide or amino acid patterns influence the behavior of different MSA algorithms would be an example of a research question that could easier be answered with an extension like this.

The profile superalignment approach offered by *AlignmentComparator* (chapter 7.2.1, page 93) currently only allows to compare two MSAs, in contrast to the other supported superalignment methods. To add support for more MSAs, profile alignments could be performed along a previously calculated

guide tree, similar to how it is already done in the maximum sequence pair match approach. Implementing this would be straightforward but was not a focus of the current development.

Although compared MSAs are often created from the exact same unaligned sequences, there are other use cases where unaligned sequences slightly differ between the MSAs (e.g. due to hot spot deletion). As mentioned in chapter 7.4.3, the profile superalignment approach is currently the only one that supports comparing such MSAs, since it relies on the actual sequence tokens (e.g. nucleotides or amino acids) instead of their unaligned position (which also has the disadvantages described in chapter 7.4.3). An alternative approach that could be explored to deal with such cases and more than two MSAs at a time could create consensus sequences from all MSAs and align these using available MSAs algorithms. From the MSA of these consensus sequences, a superalignment can be reconstructed. Such an approach may suffer from the loss of information when reducing a whole MSA to a consensus sequence but would allow to easily take advantage of the optimization strategies available in the many different MSA algorithms available. (Providing existing implementations with unaligned consensus sequences is straight forward, while providing a set of profiles to perform use for a multiple sequence alignment is not always directly supported.) It would have to be explored how such an approach would perform compared to sole profile aligning along a guide tree as described above and to the alternatives currently provided by *AlignmentComparator*.

When determining the position of supergaps by calculating the DP matrix in the maximum sequence pair match approach, several optimal paths through the matrix may occur and currently the top-most path is selected arbitrarily. (See example in Figure 7.4.) As an extension of this approach, the alternative superalignments resulting from the alternative paths could be saved and tried when performing subsequent superalignments along the guide tree to improve the overall superalignment. (Note that equally optimal pairwise superalignments are not necessarily equally good for subsequent superalignment steps, since superaligning along a guide tree is a heuristic approach.)

Estimating the guide tree of the maximum sequence pair match algorithm with alternative methods to neighbor joining could also be tried in the future to optimize superalignment results when comparing a larger number of MSAs. UPGMA [225] is, e.g., reported in MUSCLE [207] to provide better guide trees then neighbor joining in most cases for multiple sequence alignment. It is however not trivial select a guide tree method. Which guide tree is optimal depends on the optimality criterion of the algorithm (that is here also used to create the distance matrix) and good guide trees for different MSA algorithms are not necessarily good guide trees for superalignment algorithms. Benchmark tests would have to be performed to test, which method is optimal for which superalignment approach and which datasets. Since the number of compared MSAs is usually small, the influence of the guide tree on the result is more limited than in multiple sequence alignment but evaluating the optimal tree inference method could be task for the future development.

If the unaligned sequences of compared input MSAs just differ by a certain amount of deletions, e.g., due to different hot spot deletion as mentioned above, it would be possible to detect the sequence parts that have been deleted from the unaligned sequences of one MSA by comparing each sequence to its counterpart in another MSA without these deletions. Since usually a small number of longer deletions and no substitutions would be to expect due to, e.g., hot spot deletion, identifying deleted regions would be less complex than usual pairwise sequence alignment. (Ambiguous situations might occur when parts of repetitive patterns are removed, but these would only lead to incorrect unaligned indices assigned to tokens that are part of the repeat. All subsequent tokens would still have correct unaligned indices.) If deletions in the respective input MSAs are identified, it is easily possible to correct the unaligned token indices behind such deletions to match those of other compared MSAs by just adding the number of previously deleted tokens. If that preprocessing would be done, datasets not

originating from the exact same unaligned sequences could also be compared using the average position and the maximum sequence pair match approaches.

On the implementation side, it would be possible to reduce the space complexity of the maximum sequence pair match algorithm (as already mentioned in chapter 7.2.3.3.1 on page 110). Similar to the Myers and Miller linear space algorithm for global pairwise sequence alignment [216] that is based on the ideas of Hirschberg [226] for pairwise sequence alignment, the space complexity of calculating the DP matrix of this approach (chapter 7.2.3.1, page 107) can be reduced from quadratic (see chapter 7.2.3.3, page 110) to linear. This can be achieved by only storing one row of the DP matrix at a time and using the divide and conquer approach instead of tracing back through the whole DP matrix to obtain a superalignment. (For calculating the distances between two MSAs in order to obtain the guide tree as described in chapter 7.2.3.2 on page 109, calculating one row at a time would already be sufficient and the divide and conquer approach of Hirschberg, Myers and Miller would not be necessary, since only the score and no supergap positions are needed for this operation. For the subsequent superalignments along the guide tree, the divide and conquer approach would though be needed to reduce the space complex.) Reducing the space complexity was not a focus of the current development, since superalignments of MSAs with tens of thousands of columns are not problematic on current hardware and calculation time slightly increases when using Hirschberg's approach (although the time complexity is identical). It is though planned to apply the space optimization in the future to also support comparing super large MSAs.

7.4.5.2 Additional user interface features

The current version of *AlignmentComparator* displays all compared MSAs underneath each other within separate scrolling containers, as described in chapter 7.4.1. Future versions could allow the user to select some sequences to be hidden from all MSAs in the current view to focus on the remaining ones that are especially relevant in the inspected area and to use the available vertical space more efficiently. Similarly, the GUI could allow users to hide whole MSAs from the current comparison. If this happens, supergaps that are present in all remaining MSAs should be hidden as well, while keeping the same column numbering displayed above the comparison.

When hiding some MSAS, it may even be considered to further optimize the comparison with respect to the currently visible MSAs (e.g. by recalculating the current comparison using a different guide tree in the case of approaches that use one), but this would also mean that the superalignment may change, depending in the currently visible MSAs. Dynamically changing the superalignment based on the currently visible MSAs may be confusing for users and makes anchoring comments more complex. Therefore, it should only be considered as an optional feature that can be turned off by the user.

Comments used, e.g., to describe encountered alignment differences are currently anchored on a start and an end column within the superalignment. This may be further improved by allowing users to select separate columns for all or only a set of compared MSAs instead. This would have advantages, since the MSAs between which the commented differences occurred are then directly modeled and comments would even stay at the correct position if dynamic changes to the superalignment as described above would be implemented. On the other hand, this would also make creating comments more laborious for the user.

In addition to the comparison functionality currently offered by *AlignmentComparator*, the application may also be used to create new MSAs from a set of existing ones that have been compared. A user evaluating an MSA may choose to combine different parts from different MSAs, which can be done most efficiently when looking at a superaligned comparison of these. Since sequences may be differently shifted to each other, combining different MSAs using a usual alignment editor can be cumber-

some, because cutting the different parts at a certain column for all sequences is not sufficient. *AlignmentComparator* could offer functionality to automatically combine different parts from different MSAs by letting the user select and edit the cutting points directly within the comparison visualization. Beyond the functionality of manually selecting parts for an output MSA, *AlignmentComparator* may also aid the user by visualizing the possible alternative combinations, which can be described by a directly acyclic graph [227].

7.4.5.3 Further metadata visualization

In addition to the currently used *LibrAlign* data areas (see chapter 7.3.1 on page 115 and 3.3.1 on page 49) to visualize average column indices and user comments, additional kinds of comparison metadata could be visualized by implementing respective data areas. Information on the number of superaligned tokens per column or data provided by *AlignStat* (see chapter 7.4.4, page 121) would be some of many examples. Visualizing additional metadata can be done with a reduced amount of work due to the flexible data area architecture of LibrAlign.

Since *AlignmentComparator* is based on *LibrAlign* and uses *NeXML* as its main format, all types externally defined metadata could be visualized attached to sequences or whole MSAs within a superalignment. This would allow researchers to take different kinds of metadata into account when investigating MSA differences. An example would be a data area displaying positions of patterns like tandem repeats with different numbers of periods in different sequences, inverted repeats or inversions. Using such a data area with *AlignmentComparator* would allow researchers to directly inspect whether such patterns were aligned correctly and how the influenced the alternative MSAs. As described in chapter 3.3.6 (page 54), externally implemented data areas of *LibrAlign* allow to display metadata from *NeXML* files attached using externally defined ontologies. Applications like *AlignmentComparator* could display such metadata without explicitly modeling it. Providing support for using such externally implemented data areas is a future perspective, when more applications with respective data areas are developed. (See also chapter 13.1.3 on page 182.)

7.4.5.4 Further improved interoperability using *NeXML*

AlignmentComparator writes its comparison results to *NeXML* files that contain the compared alignments and their comparison as metadata. The current version uses the same token set for all nucleotide alignments. This set contains all nucleotide characters and ambiguity codes including “T” for DNA and “U” for RNA to allow the comparison of DNA and RNA alignments or even support alignments that contain DNA and RNA sequences. Because of that, *JPhyloIO* falls back to the `DISCRETE` sequence type of *NeXML*, since neither the `DNA` nor the `RNA` data type allow using both symbols in combination. Although *AlignmentComparator* can read such comparisons again without problems, it may decrease interoperability in some cases, since other applications that may read the comparison output will not be aware of the actual nucleotide sequence types in the file. To maximize interoperability and to make optimal use of the *NeXML* standard, future versions of *AlignmentComparator* should check each alignment on writing and select the `DISCRETE` data type only of both “T” and “U” are present and select the respective *NeXML* sequence type for each alignment separately. Implementing such a behavior was not yet given priority, since it does not influence the core functionality of *AlignmentComparator*, but it will be addressed during the future development.

7.5 Conclusion

A large number of alternative algorithms for automated multiple sequence alignment and alignment postprocessing tools are available and the problem of creating optimal MSAs under the criterion of homology, structure or common function remains to be the subject of ongoing research. Therefore, researchers often need to evaluate alternative MSAs resulting from different methods in their studies

and possibly apply manual changes. *AlignmentComparator* is an application that helps to do this conveniently and efficiently by visualizing differences between alternative MSAs using superalignments created by one out of three available methods, which are not available to date in any other tool. If manual editing of MSAs is done, the concrete changes should be documented, e.g. by providing the initial and the edited versions as separate files with a study. The changes can then easily be comprehended by comparing both files using *AlignmentComparator*. Its functionality is therefore useful in increasing the quality and reproducibility of all kinds of biological studies that rely on multiple sequence alignment.

Since it is based on *JPhyloIO* (chapter 2, page 33), *AlignmentComparator* is very interoperable. It supports comparing MSAs in various formats and saves its comparison results in *NeXML* using externally defined ontologies, therefore making them accessibly in a generally accepted format instead of using its own custom one. Together with the alignment GUI components from *LibrAlign* (chapter 3, page 46), the architecture of *AlignmentComparator* easily allows to handle all kinds of (future) metadata associated with MSAs and their sequences to display it directly in the superalignment, visualizing the relation and possible influence of such metadata to MSA differences. As mentioned in chapter 7.4.5.3, future versions of *AlignmentComparator* could be extended to support visualizing metadata attached in *NeXML* using predicates from any externally defined ontology and display it using respective externally implemented data areas without the need of explicitly modeling any of such metadata within the application. This way, great flexibility is achieved and *AlignmentComparator* (due to the functionalities of the libraries *JPhyloIO* and *LibrAlign* developed in this thesis) can additionally become a sequence-related comparison tool that is useful far beyond the comparison of raw MSAs and address the requirements of research in the age of big data and the semantic web.

7.6 Availability and requirements

Project name: *AlignmentComparator*

Project web page: http://bioinfweb.info/AlignmentComparator/

GitHub Repository: https://github.com/bioinfweb/AlignmentComparator

ResearchGate project page: http://r.bioinfweb.info/RGAlignmentComparator

Operating system: Platform independent

Programming language: *Java*

Other requirements: *Java* Runtime Environment 8 (or higher)

License: *GNU General Public License* Version 3 (*GPL*)

Any restrictions on use by non-academics: The restrictions specified in the *GPL* apply. (See http://bioinfweb.info/AlignmentComparator/License.)

7.7 Declarations

7.7.1 Authors contributions

Ben Stöver developed the concept, implemented the software and wrote the manuscript. Kai Müller contributed conceptually and to the manuscript.

7.7.2 Acknowledgements

We are thankful for the feedback from the users of *AlignmentComparator* and to Jürgen Schmitz for his comments on possible use cases to manually combine alternative MSAs. The work of the contributors of the open projects used is highly appreciated (*Apache Commons*, *Forester*, *Google Guava*, *Tango Desktop Project*, *OWL API*, *JUnit*, *Hemcrest*).

8 *TreeGraph 2*: Combining and visualizing evidence from different phylogenetic analyses

Stöver BC[1,2], Müller KF[1]*

[1]Institute for Evolution and Biodiversity, WWU Münster, Hüfferstraße 1, 48149 Münster, Germany

[2]Nees Institute for Biodiversity of Plants, University of Bonn, Meckenheimer Allee 170, 53115 Bonn, Germany

*Author for correspondence: kaimueller@uni-muenster.de

This chapter has been published in *BMC Bioinformatics* 2010, **11**:7
http://dx.doi.org/10.1186/1471-2105-11-7

Abstract

Background: Today it is common to apply multiple potentially conflicting data sources to a given phylogenetic problem. At the same time, several different inference techniques are routinely employed instead of relying on just one. In view of both trends it is becoming increasingly important to be able to efficiently compare different sets of statistical values supporting (or conflicting with) the nodes of a given tree topology and merging this into a meaningful representation. A tree editor supporting this should also allow for flexible editing operations and be able to produce ready-to-publish figures.

Results: We developed *TreeGraph 2*, a GUI-based graphical editor for phylogenetic trees (available from http://treegraph.bioinfweb.info). It allows automatically combining information from different phylogenetic analyses of a given dataset (or from different subsets of the dataset), and helps to identify and graphically present incongruences. The program features versatile editing and formatting options, such as automatically setting line widths or colors according to the value of any of the unlimited number of variables that can be assigned to each node or branch. These node/branch data can be imported from spread sheets or other trees, be calculated from each other by specified mathematical expressions, filtered, copied from and to other internal variables, be kept invisible or set visible and then be freely formatted (individually or across the whole tree). Beyond typical editing operations such as tree rerooting and ladderizing or moving and collapsing of nodes, whole clades can be copied from other files and be inserted (along with all node/branch data and legends), but can also be manually added and, thus, whole trees can quickly be manually constructed de novo. *TreeGraph 2* outputs various graphic formats such as *SVG*, *PDF*, or *PNG*, useful for tree figures in both publications and presentations.

Conclusion: *TreeGraph 2* is a user-friendly, fully documented application to produce ready-to-publish trees. It can display any number of annotations in several ways, and permits easily importing and combining them. Additionally, a great number of editing- and formatting-operations is available.

8.1 Background

It has become standard to apply multiple inference techniques to a given phylogenetic problem. The recent invasion of phylogenetics by Bayesian techniques (e.g., [59]), the ever improving models and algorithms for tree searches under maximum likelihood (e.g., [228,229]), and the continuously growing processor speed helped these previously computationally very expensive approaches to become a typical component of most phylogenetic studies, accompanying the widespread parsimony and distance-based approaches. At the same time, no single inference technique has consistently proven to be the single best choice. Accordingly, the researcher is well-advised to explore potential method-specific differential results, leaving him or her with the difficulty of visualizing these differences for him- or herself

and for the reader. Frequently, differences are restricted to the magnitude of various measures of statistical support (such as jackknife and bootstrap proportions, Bayesian posterior probabilities), rather than being apparent from the topology. In addition, the frequently reported results from topological tests (e.g., [230]) or tracing of ancestral character states (e.g., [54]) add further importance to being able to assign a variety of numbers and graphical labels to tree nodes.

To address those needs, the first version of *TreeGraph* [231] had been developed, which strongly simplifies the creation of the final tree figure by the automatic positioning and formatting of multiple labels per branch. However, while one support type could directly be imported from the phylogeny inference program output, the *Newick*- and *Nexus* [31] format used by these programs precluded the direct import of more branch labels. For all additional labels (support values), the laborious work of mapping them onto the appropriate nodes remained. The cumbersome drawing part of the publication process was minimized, but it remained the user's responsibility to collect and position all information that was to be displayed at the nodes.

We figured that automating this process would be very useful, particularly so in studies of extensive gene family datasets that may contain several hundred terminals. Gene family studies using phylogenetic approaches have become a major focus with the increasing amount of available fully sequenced genomes. Typically, gene family trees suffer from weak support [232–234]. The entailed caution required when interpreting gene family trees increases the need for testing alternative inference methods, alignment methods, data partitions, and varying treatment of questionable alignment regions.

Similarly, the differential contribution of and potential conflict among different data partitions is frequently estimated by the differential success of resolution and degree of statistical support in various parts of the tree contributed by each partition [235]. This has become particularly important since multigene analysis are the rule rather than the exception, a trend further fueled by the growing availability of complete (organellar) genomes that provide easy access to a large number of genes that can be concatenated in large data matrices and then subjected to phylogenetic analyses, e.g. [236].

These trends call for a tree editor that is able to compare and ultimately visualize congruent and conflicting evidence from different analyses, while guaranteeing flexible editing and production of high-quality tree figures for publications.

8.2 Implementation

TreeGraph 2 is written in *Java* and uses *Swing* for its graphical user interface (GUI) as well as the *Apache Batik SVG Toolkit* (http://xmlgraphics.apache.org/batik/), *FreeHEP* (http://java.freehep.org/), *Java Math Expression Parser* (http://sourceforge.net/projects/jep/) and *BrowserLauncher* (http://browserlaunch2.sourceforge.net/) libraries. Besides its GUI, which makes editing and formatting very intuitive, the current version 2 adds many features previously unavailable in the command line precursor and introduces an *XML*-based native file format (*XTG*).

8.3 Results and discussion

8.3.1 Importing data

TreeGraph 2 can read trees in *Newick* or *Nexus* format (including additional annotations in comments specified by *BEAST* [237]) as well as *phyloXML* tree descriptions [36] and can furthermore import annotations from text files generated e.g. with a spreadsheet application. Besides that, *TreeGraph 2* facilitates combining information from different phylogenetic analyses of a given dataset. This is particularly useful e.g. in the study of extensive gene family datasets with large sets of terminals. The following sections describe this feature in greater detail.

8.3.1.1 Mapping statistical support onto congruent nodes

For each branch of a tree opened in *TreeGraph 2*, the corresponding support from other trees can be mapped whenever the topology defined by the current branch is present in them. Each of these other trees may represent the result from a different analytical approach or different data partition, and support values from these trees are assigned their own label ID by which they are grouped and amenable to future formatting or editing operations. Thus, all support values that stem from a particular analysis can be individually formatted e.g. by their relative position on the branch and/or their font and style.

8.3.1.2 Finding conflicting nodes and mapping contradictory support

In some studies not only the support from different analyses has been mapped onto the branches but also the strongest support for a contradictory topology was determined by inspection via eye [238,239].

TreeGraph 2 uses the following algorithm automate this (for a better understanding it should be kept in mind that each branch splits a tree into exactly two subtrees).

Let *tree1* specify the topology onto which contradictory support from other trees should be mapped (example in Figure 8.1a). For a given branch *branch1* in *tree1*, the maximum support for a conflicting branch *branch2* from another tree *tree2* (example in Figure 8.1b) can be found as follows.

1. Find the *branch2* which defines a subtree *subtree2* with the smallest number of terminals that contains all leafs of a subtree *subtree1* defined by *branch1*.
2. Inside *subtree2* find all branches that define a subtree which are on the one hand fully enclosed by *subtree2* and on the other hand contain at least one terminal which is also part of *subtree1* as well as at least one leaf which is not.
3. The highest support value in the set of these branches is added as a conflicting value onto *branch1*.

This highest conflicting support value can be distinguished from congruent values by user-specified formats, e.g. brackets, asterisks or different colors (see example in Figure 8.1).

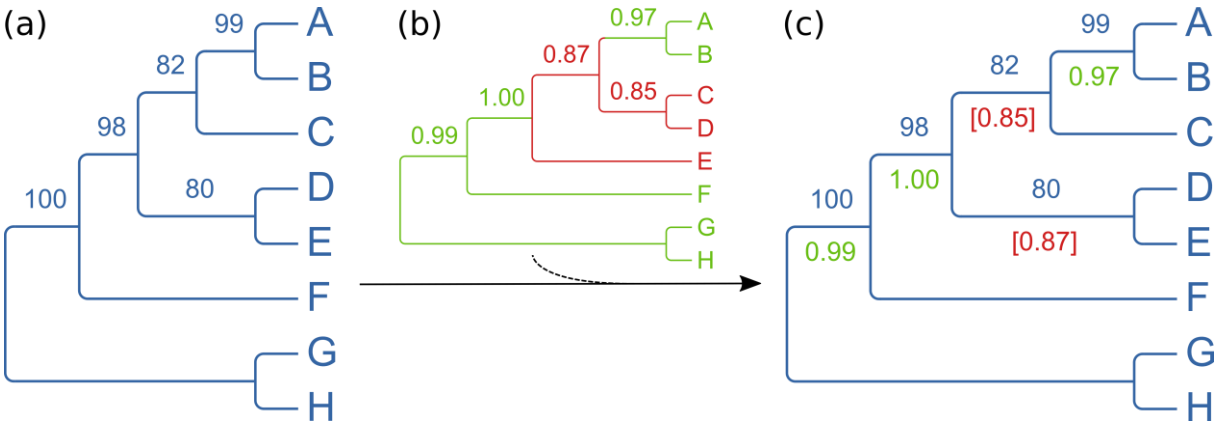


Figure 8.1 Merging support values from different analyses - a simple contrived case[c]

The tree on the left (a) was first opened in TreeGraph 2 *and defines the topology and optionally a first set of support values. (Alternatively a consensus tree of all analyses or any user-defined tree could be used here.) Afterwards the annotations from another tree (b) have been added which resulted in a new group of values (c) supporting (green) or contradicting (red) the initially loaded topology (blue).*

[c] Note that this figure shows the tree merging functionality in 2010, which has been extended since then. The state that was reached on submission of this thesis is described in Figure 9.1 (page 52) in the next chapter.

8.3.2 Editing and formatting capabilities

The program features versatile editing and formatting options, such as automatically setting branch widths or colors according to the value of any of the unlimited number of variables that can be assigned to each node or branch.

8.3.2.1 Editing of node/branch data

Node/branch data imported from spread sheets or other trees (as described above), can be copied from and to other internal variables, be kept invisible or set visible and then be freely formatted (individually or across the whole tree), filtered according to their values or calculated from each other using an integrated mathematical expression parser which can access all node/branch data columns. Figure 8.2 shows a screenshot displaying a tree and its corresponding data table.

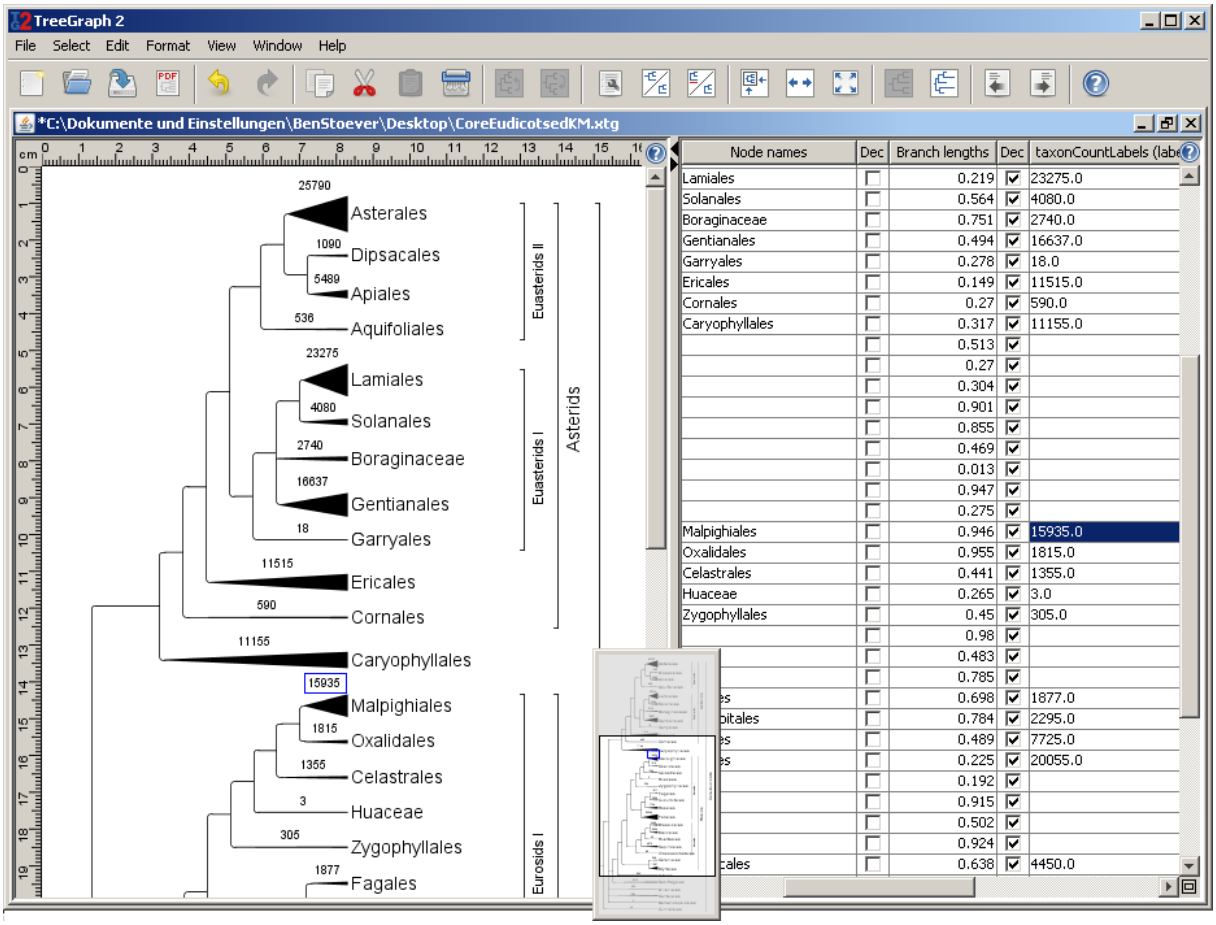


Figure 8.2 Example view of the* TreeGraph 2 *GUI showing taxon counts displayed as branch widths

The taxon counts of all terminal nodes have been imported from a table (text file) to a hidden node data column. The imported annotations have then been used as source data to set the terminal branch widths. For each TreeGraph 2 *document, one can optionally view the node/branch data table in the right part of the document window as shown here.*

8.3.2.2 Editing operations

Beyond typical editing operations such as tree rerooting and ladderizing or moving and collapsing of nodes, whole clades can be copied or cut out and placed into new empty files or inserted (along with all node/branch data) into other trees. Since nodes can also be manually added, whole trees can quickly be manually constructed starting from an empty file.

The editing operations are facilitated by versatile additive selection options that allow selecting many elements in a tree for subsequent formatting with just a few clicks. Additionally, every operation applied to an opened tree can be easily undone or redone using the undo-function.

8.3.2.3 Searching, replacing and translating tree leaf names

Searching and replacing is possible across all node/branch data columns (including taxon names and node labels).

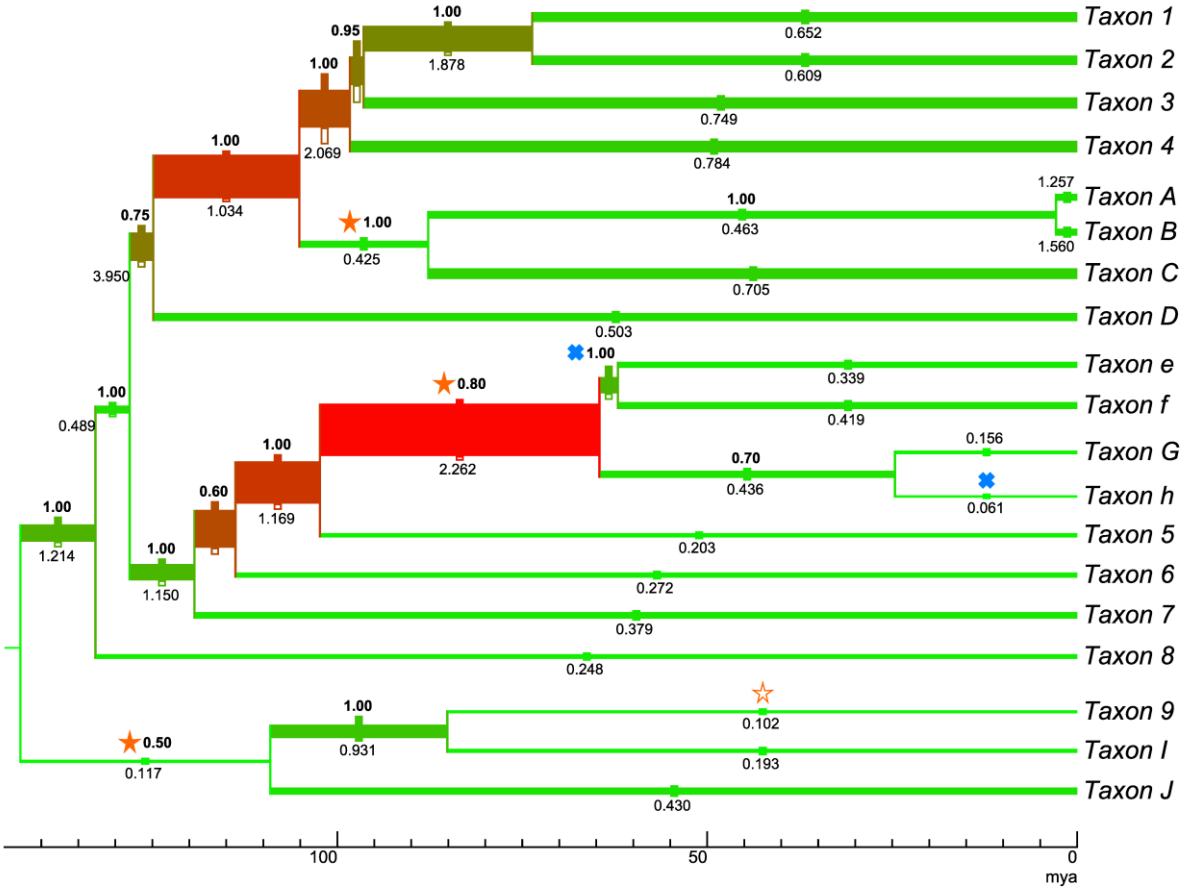


Figure 8.3 Displaying multiple annotations and assigning element formats automatically

The tree in this contrived example contains several annotations including ancestral divergence times (node heights; expressed as branch lengths in an ultrametric tree), DNA substitution rates, posterior clade probabilities as they could have been imported by TreeGraph 2 *from, e.g., a tree file generated with help of* TreeAnnotator *after a* BEAST *analysis. As in a typical chronogram view, the age of the nodes (in million years ago) is expressed by the scale bar at the bottom. In addition,* TreeGraph 2 *was asked to automatically assign branch widths and line colors to illustrate the mean evolutionary rates for each branch, while the accuracy of each rate estimate was illustrated by a filled rectangular label icon above and an unfilled one below each branch (the branch width extended by the size of the upper icon describes the highest rate in e.g. a 95% confidence interval and the branch width reduced by the size of the lower icon describes the lowest rate in the interval). Text labels have been used to show the posterior clade probabilities (above the branches, bold) and the absolute substitution rates in substitutions per site per billion years (below the branches). Furthermore, this example tree contains star and cross icon labels that could be used, e.g., to highlight specific character state transitions (such as orange stars indicating "number to character" shifts (filled) or* vice versa *(not filled), and blue crosses representing "upper case to lower case" shifts).*

More restrictive alignment file formats do not allow lengthy taxon names, so names get truncated. In other cases, the often clumsy taxon- or lab IDs used during a study survive up to the final alignment, phylogenetic dataset and the trees constructed from it until they need to be adjusted for the final tree

to be presented in a paper. *TreeGraph 2* can be requested to apply a translation table to use "cleaned" taxon names for the final output. This translation table can be constructed easily with help of the data export feature and any text editor or spread sheet program. Furthermore, the lab IDs (old terminal names) can be saved in a hidden data field to be able to identify the terminals by these lab IDs so that additional support values could still be added later on.

8.3.2.4 Formatting document elements

Great flexibility is offered by the application as it allows free formatting of line- and text-formats of all document elements like nodes, branches or legends (which mark a group of terminals). Additionally branches can carry an unlimited number of textual annotations (text labels) or icons (icon labels) the color, text style or size of which can also be freely formatted (see Figure 8.3). All distance values in *TreeGraph 2* (e.g. line width or text height) are specified in millimeters or DTP-points (1/72 inch). This feature, along with the image export function (see below), allows the user to design trees in exactly the size they should appear in print or in the exported graphic file. In addition, *TreeGraph 2* offers a feature to proportionally rescale all elements of a subtree or the whole document.

8.3.2.5 Automatically setting line width, text height, and color

TreeGraph 2 allows automatically setting all formats (e.g. branch widths, branch colors, text colors, text heights, icon sizes) according to the value of a chosen node/branch data column. This provides a very intuitive way to graphically present the relative magnitude of, e.g., certain types of support or rates assigned to branches (see Figure 8.2 and Figure 8.3 for examples).

8.3.3 Different view modes

All editing operations are facilitated by a very convenient way to zoom in and out, fitting the zoom to the window size, and a miniature overview (Figure 8.2) for navigating large trees.

When applicable (i.e., given that branch length information is provided), trees can be displayed as phylogram or chronogram (Figure 8.3), with multiple options for adjusting a scale bar (to indicate e.g. time spans in chronograms, rates in ratograms, or branch lengths in phylograms).

8.3.4 Exporting to graphic formats and printing

TreeGraph 2 outputs various vector and (anti-aliased) pixel graphic formats. Among these are *SVG*, *PDF*, or *PNG*, supporting transparent background where this applies. Using the graphic export function of *TreeGraph 2*, the most adequate graphic formats, resolutions, and image sizes for manuscripts, presentation slides, or web pages, respectively, can be specified.

8.3.5 Help

An extensive, continuously updated online help system is available under http://treegraph.bio-infweb.info/Help and can also be accessed (in a context-dependent manner) from within the program. Additionally, several video tutorials are offered there to get started with *TreeGraph 2* (see http://tree-graph.bioinfweb.info/Help/wiki/Tutorial:Main_page).

8.3.6 Comparison to previous software

To date, a variety of tree visualization tools have been released, among which *ATV* [85], *Dendroscope* [240], *FigTree* (the tree editor accompanying *BEAST*), the *MEGA* tree explorer [241], *Mesquite* [111], *PhyloWidget* [242], *TreeDyn* [243] and *TreeView* [244] may be the most widely distributed. In spite of their great usefulness for the purposes they have been developed for, none of these software packages allows simultaneously visualizing, freely editing, properly formatting and exporting or printing trees with heavily annotated nodes (see Table 8.1). Although *TreeDyn* is able to display multiple annotations on one node it is not able to automatically position them in a ready-to-publish way or to combine them from different analyses. *FigTree* is able to read the special *Newick* annotations generated by *BEAST*

and therefore can also store several sets of annotations but only offers a limited number of ways to display them (like branch lengths or one textual annotation per branch). In contrast, *TreeGraph 2* (which is also able to read *BEAST* annotations) can show a nearly unlimited number of textual annotations at a time as well as display data in form of branch widths, line colors or many other formats.

Besides importing additional annotations from tables (which *TreeDyn* also offers), *TreeGraph 2* is the only editor which can combine annotations (e.g. statistical support from different analysis methods) from different trees (with the same set of terminals). The information gained this way has a topological component and can therefore not simply be obtained from data in a table.

A feature closely related to the ones mentioned above is the ability to calculate numeric or textual annotations by mathematical expressions, which can reference other annotations (see above). To date, a similar functionality is not offered by any other tree editor.

TreeGraph 2 features a multitude of format options, which can be combined to every tree element (e.g. branches, nodes or labels) independently. As Table 8.1 shows, no other tree editor currently provides functionalities like element-specific formats for all types of tree elements in combination with advanced selection options or collision free positioning of the whole tree. Moreover, none of the editors that offer at least some of *TreeGraph 2*'s formatting options allow the user to precisely determine the print layout. In contrast to most other editors, our program offers context help buttons (which link to the online help system) everywhere in the program, making it very easy for new users to get started.

It should be noted, however, that *TreeGraph 2* has been optimized as a tree editor for producing high quality tree figures and not as a viewer for trees with many thousands of taxa, which could never be depicted completely in a publication or presentation. The latter is a specialty of software specifically designed for this purpose such as, e.g., *Dendroscope* [240] (Table 8.1).

Since *TreeGraph 2* is written in *Java* and is able to read and write all its supported formats directly from and to streams in would be possible to use it in a web application either on the server (e.g. with *Apache Tomcat*) or the client site (e.g. as an *Java Applet* or a *Java Webstart* application) to display and manipulate trees. As yet, our application would have to be integrated into such a web application by its programmer manually and we do not yet offer a ready-to-use plug-in solution for this. We do, however, offer a full documentation of our source code (including its interfaces) to facilitate such a web integration.

Table 8.1 Comparison to other tree editors *(Table on the next page.)* [d]

I: Import, IE: Im- and export. [1]All programs tested with balanced binary trees in Newick format. The value listed is the number of terminals of the largest tree that could still be opened in less than two minutes on an average desktop computer (2.2 GHz AMD Athlon™ XP *processor, 1 GB RAM)*[e]*. [2]Numerical and textual annotations of nodes and branches can be calculated by any user defined mathematical expression from the values of other annotations in the tree. [3]Any tree element can be copied to any position in the same or another tree (Programs that can only copy whole trees or paste subtrees to a new file are not checked in this column.). [4]User defined text replacement in node names and all annotations. [5]Numerical values of annotations define formatting of tree elements (e.g. color, width, text height). [6]Documentation going beyond the original publication and explaining the different options. [7]Only the last edit can be undone. (In contrast,* TreeGraph 2 *stores a whole undo history which can be undone (and redone) to any point.). [8]Positioning options for the labels are not offered. [9]Only one direction (not up and down). [10]TreeDyn allows labeling a group of nodes with a legend (not automatically positioned), but the label gets lost during edit operations like ladderizing. [11]Specific formats for subtrees are possible. Branches and nodes cannot be formatted independently. [12]Only very brief descriptions.*

[d] Note that this table is a figure in the original publication, due to formatting limitations of the Journal. All information reflects the state in 2010. *TreeGraph 2* has been extended since then and other software may have too. See chapter 9 (page 52) for the state on submission of this thesis.

[e] That was a common hardware configuration at the time of publication. Results on common hardware today may differ.

Table 8.1 *(Description on the previous page.)*[d]

Software (version)		**TreeGraph 2** (2.0.40 beta)	Archaeopteryx (ATV) (0.954 beta)	Dendroscope (1.3.1, build 4)	FigTree (1.1.1)	Mega Tree editor (4, build 4028)	Mesquite (2.7.1, build 514)	PhyloWidget (2009-11-11)	TreeDyn (198.3)	TreeView (1.6.6)
Import / Export	Newick	IE	IE	IE	I	IE	I	IE	I	I
	Nexus	IE	IE	IE	IE		IE	I		IE
	phyloXML	I	IE							
	Nexus (BEAST package data)	I			IE					
	Mapping support from other topolgies (analyses)	✓								
	Annotation import from table	✓			✓				✓	
	Read trees from databases		✓				✓			
	Maximum number of terminals[1]	2^{13}	2^{16}	2^{18}	2^{14}	2^{14}	2^{12}	2^{14}	2^{14}	2^{19}
Graphic formats	SVG	✓		✓	✓				✓	
	PDF	✓	✓	✓	✓		✓	✓		
	PNG	✓	✓	✓	✓				✓	
	Additional vector formats	✓		✓	✓	✓			✓	✓
	Additional pixel formats	✓	✓	✓	✓	✓		✓	✓	
	Customizable output resolution	✓								
Editing	Undo function	✓					✓[7]			✓[7]
	Advanced selection options	✓		✓						
	Manual tree construction	✓	✓					✓		
	Displayable textual annotations per node/branch	∞	∞[8]	2	2	1	3	2	∞[8]	1
	Calculation of annotations[2]	✓								
	Scaling of branch lengths	✓				✓	✓	✓		
	Copying, cutting and pasting of all tree elements[3]	✓	✓					✓		
	Ladderizing	✓	✓	✓	✓		✓	✓[9]	✓	✓
	Rerooting	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Collapsing nodes (to polytomy)	✓					✓	✓	✓	✓
	Search	✓	✓	✓	✓			✓	✓	
	Replace[4]	✓								
Formatting	Collision free positioning of all tree elements	✓								
	Line formats by annotations[5]	✓			✓					
	Text formats by annotations[5]	✓								
	Positioning by annotations[5]	✓								
	Legends for node groups	✓				✓			[10]	
	Advanced legend positioning	✓								
	Text color	✓		✓	✓	✓	✓		✓	
	Text color (element specific)	✓		✓					✓	
	Text size	✓	✓	✓	✓	✓	✓		✓	✓
	Text size (element specific)	✓		✓					✓	
	Font, text style	✓		✓		✓			✓	✓
	Font, text style (element specific)	✓		✓					✓	
	Line color	✓	✓	✓	✓	✓	✓		✓[11]	
	Line color (element specific)	✓								
	Line width	✓		✓		✓	✓	✓	✓[11]	
	Line width (element specific)	✓								
	Independent start and end branch width	✓								
	Proportional rezising of all elements in a subtree	✓								
	Customizable scale bar	✓			✓	✓				
View	Print layout in millimeters or inches	✓								
	Editable annotation table	✓								
	Rectangular cladogram view	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Rectangular phylogram view	✓	✓	✓	✓	✓	✓	✓	✓	✓
	Network view			✓	✓	✓		✓		✓
	Circular view(s)			✓	✓	✓	✓	✓	✓	✓
	Hiding of subtrees		✓	✓	✓	✓		✓	✓	
Help	Documentation[6]	✓	✓[12]			✓	✓	✓	✓	✓
	Context help	✓				✓	✓[12]			
	Video tutorials	✓								

8.4 Conclusions

With its easy-to-use graphical user interface and a number of semi-automatic editing and formatting options, *TreeGraph 2* is a graphical editor useful in the context of any phylogenetic study. It is particularly useful where multiple, potentially conflicting trees are being produced, because its automatic combination of information from different analyses helps to identify and graphically present such incongruences. The way in which data can be imported and then assigned to nodes, manipulated or even converted to color tones, line diameters or other formats allows for a great flexibility in visualizing any kind of data associated with different parts of the tree. Together with the possibility to manually construct new clades or delete clades and the various graphic output formats supported, *TreeGraph 2* greatly reduces the effort during the preparation of tree figures for presentations or publications.

8.5 Availability and requirements

Project name: *TreeGraph 2*

Project home page: http://treegraph.bioinfweb.info/ (including an extensive documentation and a development section with *JavaDocs*)

Operating system(s): Platform independent (*Java* 6 has to be available[f])

Programming language: *Java*

Other requirements: *Java Runtime Environment* 6.0 (or higher)[f]

License: *GNU General Public License*

Restrictions to use by non-academics: none

8.6 Declarations

8.6.1 Authors' contributions

BCS developed *TreeGraph 2*, wrote the online help and contributed to the concept of the software and the manuscript. KFM was responsible for the conception and design of the software, contributed to its help system, and wrote the manuscript. Both authors have given final approval of the version to be published.

8.6.2 Acknowledgements

This work was in part supported by DFG grant MU2875/2 to KFM, since many of the features were added to the program in response to requirements encountered during work in the corresponding DFG project “Carnivory in Lamiales - Understanding character evolution, substitution rate plasticity, and genome miniaturization”. Financial support to KFM by the *Young Academy of the North Rhine-Westphalian Academy of Sciences* (*Nordrhein-Westfälische Akademie der Wissenschaften und der Künste*) is highly appreciated. Thanks to Mark Simmons, Claude dePamphilis, Dietmar Quandt, and Jörn Müller for helpful suggestions. Finally, we want to thank the authors of the open source libraries used.

[f] Note that the requirement is not Java 8 of higher, as pointed out in chapter 9.5 (page 73). The information here reflects that status in 2010, when this paper was published.

9 New features of the tree editor TreeGraph 2 to handle rich metadata and compare phylogenies

Stöver BC[1]*, Wiechers S[1], Brech P[1], Müller KF[1]

[1]Institute for Evolution and Biodiversity, WWU Münster, Hüfferstraße 1, 48149 Münster, Germany

*Author for correspondence: stoever@bioinfweb.info

Own contribution

See section 9.6.1 on page 161.

Abstract

Since its last publication in 2010, *TreeGraph 2* has become a widely used editor for phylogenetic trees and many extensions of its functionality have been made to increase its usability and to keep up with new developments in the scientific community. Sticking to its initial aim, modelling, visualizing and processing of different kinds of metadata attached to nodes and branches of phylogenetic trees has been the focus of the new features, which has become even more relevant in the age of big data and the semantic web to increase reusability of data and reusability of studies.

The feature of mapping support values from different phylogenetic analyses onto one tree topology has been extended to process trees with different (but overlapping) sets of terminal nodes and a new visual interactive tree comparison feature has been developed based in that functionality. Pie chart labels have been introduced to provide more metadata visualization options. Such metadata includes, e.g., ancestral character state probabilities, which can now be easily imported from the analysis software *BayesTraits* using special functionality and from other software using the significantly improved table import feature. More flexible and versatile ways to calculate textual and numerical annotations from each other further contribute to the set of new metadata-related features. *NeXML* and its metadata can now be imported and support for the other tree formats has been improved.

In addition to that, an extension of the metadata model is currently ongoing, that will allow to use *RDF*-predicates from externally defined ontologies to unambiguously link the different kinds of metadata that is modeled and visualized by *TreeGraph 2* and allow to use *NeXML* as the main format of the application. We report on the status of this work and discuss the advantages this new model and the other new features will have to increase the reusability of phylogenetic trees and the reproducibility of workflows. Increasing the scriptability of *TreeGraph 2* further than already done with the new features to calculate annotations from each other and making its functionality directly accessible from popular scripting languages is, among others, a further perspective for its future development.

9.1 Introduction

TreeGraph 2 is a feature-rich editor for phylogenetic trees that provides many editing and formatting options and focuses on the processing and visualization of metadata attached to nodes and branches of a tree. The development started in 2008 and since then 73 versions have been released containing contributions of four developers to introduce new features and bug fixes that address user feedback. At the time of submission of this thesis (May 2018), the first publication of the software ([76], chapter 8) was 447 times cited (including 107 times in 2017) according to the *Web of Science Core Collection* database (619 times according to *Google Scholar*), is downloaded on average 13 times a day and every 15 minutes an instance started by a user contacts our server to check for updates. This shows that *TreeGraph 2* has become widely-used in the scientific community. In order to keep it relevant and adopt it to the upcoming challenges and user needs, numerous new features have been added. Sticking

to the traditional aim of *TreeGraph 2*, most of the new features focus on managing, processing and visualizing phylogenetically relevant metadata.

The long-term availability of data produced in phylogenetic studies is a key requirement for their reusability and the reproducibility of studies [18,19]. Although availability itself remains to be an issue [19,22,47,50], parts of which databases like *TreeBASE* [44] or *Dryad* [42,46] try to address, another important issue is the documentation and annotation of data [20,22]. Without proper metadata provided together with scientific data, reproducing a study or reusing the data is significantly hampered or even impossible, because details on, e.g., how the data was edited or computationally processed or what sequence or node identifiers actually mean remains unclear. Semantic web technologies like *RDF* (*Resource Description Framework*, [37,41]) combined with externally defined ontologies allow to annotate data in an unambiguous and machine-readable way, which would, e.g., allow to automatically collect and process large amounts of specific data for a bioinformatical study without time-consuming manual data searching or even redundant generation of new data. Although formats like *NeXML* [35] that supports *RDF* annotations on all elements of, e.g., phylogenetic trees, minimum information standards like *MIAPA* [40] and phylogenetic ontologies (cf. Table 9.1 and references therein) have been developed in the past, current tree editors still lack the ability to attach and process such metadata (cf. chapter 9.3.7.3). Software can help to address these needs by supporting established well-defined formats [35,245] and providing functionality that allows users to easily describe their data unambiguously. The new I/O functionality and the extended metadata model of *TreeGraph 2* that will be presented in this chapter improve interoperability and are important steps addressing the needs to increase reproducibility and reusability.

Another metadata-related focus of *TreeGraph 2* has always been the comparison and combination of statistical branch support values from different methods. The previously published version offered a feature that allows to map support values from input trees (e.g., from maximum likelihood and Bayesian inferences) onto one target topology and to highlight topological conflicts. We will present improvements, extensions and additional complementary features that make comparing alternative phylogenetic trees even more convenient using *TreeGraph 2* and introduce additional functionality that has been added since its last publication in 2010 to allow its users to increase reproducibility of their studies and reusability of their data.

9.2 Implementation

TreeGraph 2 is written in *Java* and the implementation follow the model-view-controller paradigm. Although not developed as a programming library, the structure of the source codes of TreeGraph 2 allows to reuse its code from other applications. Tool classes like `TreeSerializer` that creates sets of tree elements (nodes, branches, labels, …) to be used with document edits (similar to the options the select menu of the GUI) further simplify the programmatic access to the application's features.

Design patterns have been applied wherever useful to ensure easy maintainability and reusability. Examples are the application of the strategy pattern [88] for reading from and writing to different tree formats or the adapter pattern [88] when accessing different annotation values through a single node/branch data interface or element formats through a format adapter.

Changes of the internal structure or *TreeGraph 2* make it now even easier to maintain and to extend. Introduced functionality like the node change type that is now assigned to every edit, allow more efficient change listeners that react only to certain modifications of an opened document.

Chapter 13.2.2 (page 186) and Figure 13.3 contain an example on how *TreeGraph 2* source codes can be used within another application. Further information for developers and the complete source codes are available at http://treegraph.bioinfweb.info/Development.

9.3 Results and discussion

9.3.1 Interactively comparing trees

As described in [76], combining annotations from different analyses of the same dataset (e.g. support values from a maximum likelihood and a Bayesian tree inference) into a single tree, while taking possible topological conflicts into account, is one of the key features of *TreeGraph 2*. Back then comparison was limited to trees containing the exact same set of terminal nodes, while the current version now allows to merge annotations from trees that only share a subset of terminal nodes. Figure 9.1 illustrates the extended merging algorithm used in recent versions of *TreeGraph 2*.

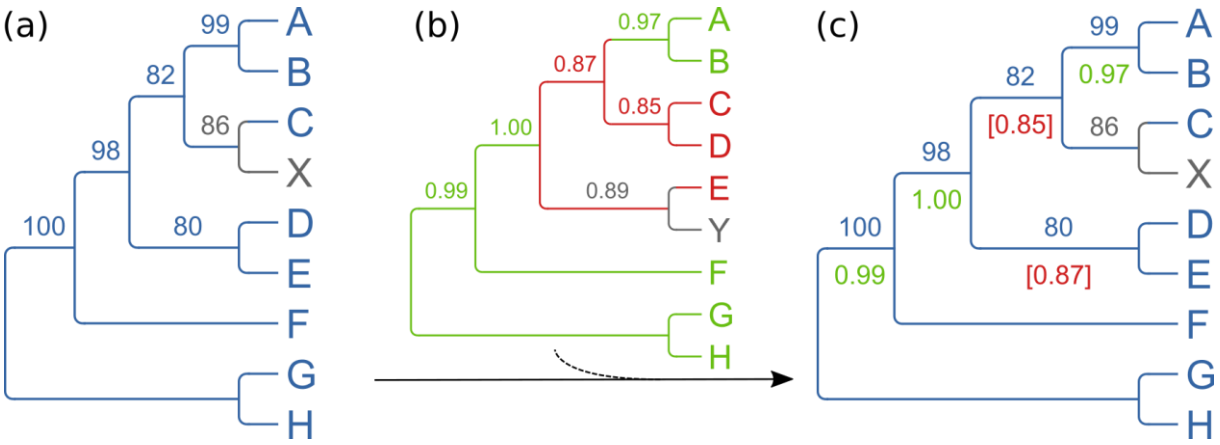


***Figure 9.1 The extended tree merging feature of* TreeGraph 2**

The first tree (a) is the initially opened tree onto which support values from another tree (b) should be mapped. The merged result is shown in (c). The example shown here is based on that in figure 1 of [76], but each tree contains an additional terminal node (X in (a) and Y in (b)) that is not present in the other tree respectively. Both additional nodes are grouped together in a subtree with one other node (which is present in both trees). The nodes unique to one tree and the parts of the topology and support values that would not exist without them are shown in gray.

When the topologies are compared to map the support values, only terminal nodes present in both trees are considered. For that reason, no support values were mapped onto the branch leading to the subtree of C and X in (c). Since X does not exist in (b) there is no annotation in (b) supporting or contradicting this node. The same is true for the support value 0.89 in (b) on the branch leading to the subtree containing E and Y. Since E is the only node in this subtree that is also present in (a), there is no respective subtree in (a) this support value could be mapped to.

As soon as the effects of all terminal nodes that are not present in all compared trees are filtered out, the tree merging still works as described in [76]. As a result, there are three supporting (green) and two conflicting (red in brackets) support values mapped in this example. The three supporting values are mapped since the subtrees they label are present in both topologies, e.g. (A, B) or (A, B, C, D, E). Note that the subtrees with the terminals A-E in both trees differ in their internal topology and in containing different additional terminal nodes unique for a single tree, but still are identical regarding the shared sets of terminal nodes their root branch separates and therefore this split is supported by both compared analyses.

The mapped conflicting values originate from topological differences between both trees. The value 0.85 in (b) is the support for the split between (C, D) and the rest of the terminals, while in tree (a) C is clustered closer together with A and B than with D. Therefore, the branch leading to (C, D) in (b) is in conflict with the branch leading to (A, B, C, X) in (a) and its support value is shown as a conflicting value there. The situation is similar for the branch leading to (D, E) in (a), with the difference that there are even two branches in (b) which define a split between D and E (the red branches carrying the support values 0.87 and 0.85). In such cases, TreeGraph 2 *chooses the highest conflicting support value and maps it onto the conflicting tree.*

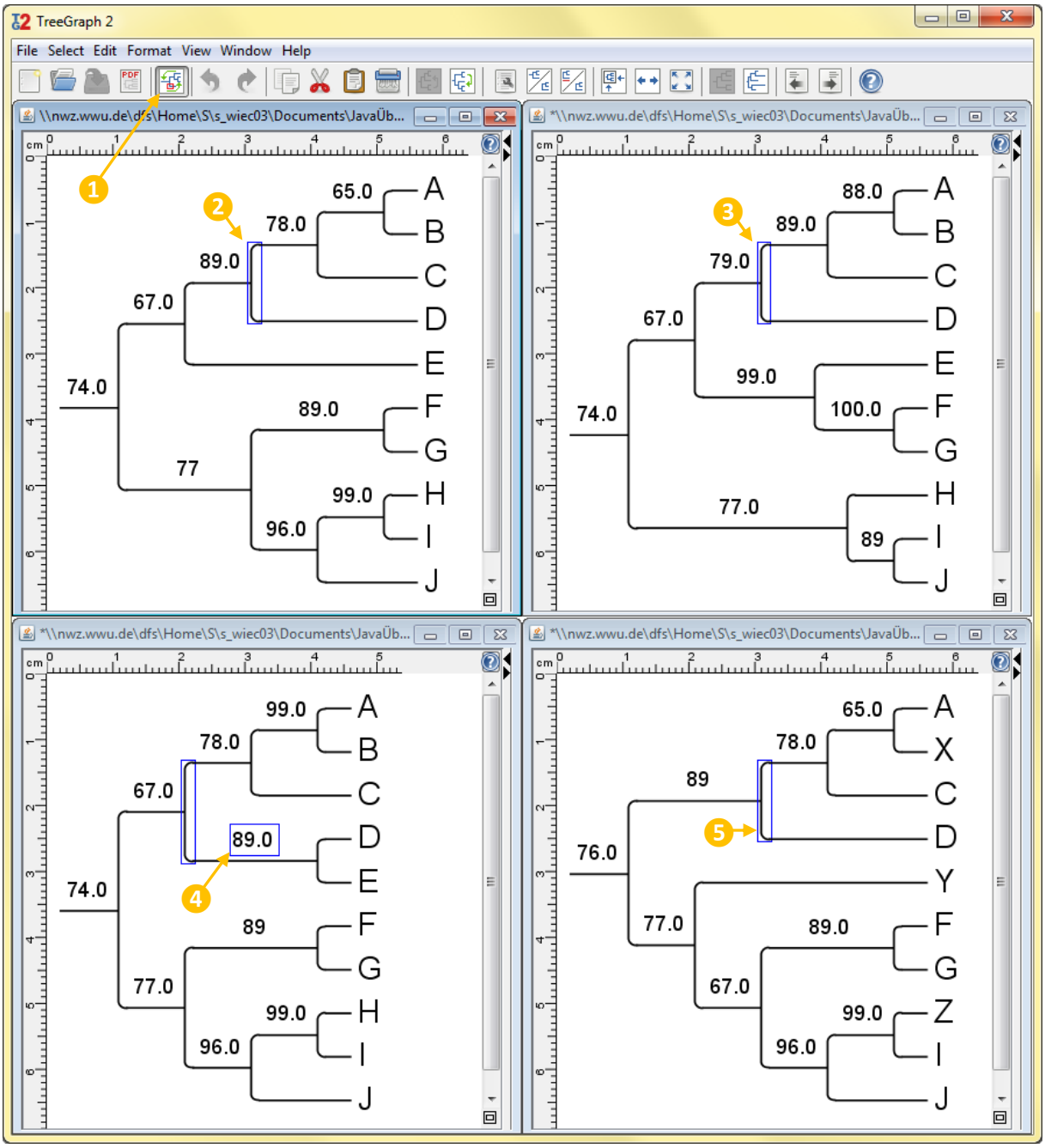


***Figure 9.2 The new interactive tree comparison feature in* TreeGraph 2**

After having opened a set of trees to be compared, the user activates the "selection synchronization" feature to start interactive tree comparison (❶). By selecting one node in any of the trees (❷) corresponding nodes and conflicting support values are automatically selected in all other trees. In this example, the top right tree contains a matching node that separates the same two sets of terminal nodes from each other (❸). The closest matching node in the bottom left tree separates the terminals A-E from the rest, but since that tree also contains a conflicting branch that separates A, B and C from D, its support value is highlighted as a conflicting support value (❹). The bottom right tree contains a different set of terminals, but TreeGraph 2 *is still able to select the closest matching node (❺) and shows that there are no conflicts, since no support values are highlighted.*

Besides the feature to create trees with merged annotations and its extension to support different sets of terminals, *TreeGraph 2* now additionally offers an interactive tree comparison feature. A user can open a set of trees and select one or more (internal or terminal) nodes in one tree and all topologically matching nodes, as well as support values on branches in conflict, will be automatically selected in all other trees. Figure 9.2 shows an example of this new interactive comparison feature. While creating a

single tree document with combined support values from different analyses is useful for producing figures that show e.g. a consensus topology and how strongly it is supported by different analyses, the new interactive comparison feature allows researchers to easily inspect each topological difference in all compared trees, while having all relevant topological information directly highlighted instead of having to focus on a single topology.

To make this comparison feature most convenient, *TreeGraph 2* now also allows to specify default node/branch data columns for node names and support values for each document. (Node/branch data column refers to a set of values (metadata) attached to a set of branches or nodes using the same string identifier.) If e.g. lab codes are stored as hidden branch data, these can be used to identify respective terminal nodes instead of the actual node names. Users can also choose between multiple sets of support values in a tree to be compared, by setting the default annotation column accordingly.

In addition to extensions of the core tree comparison features, further improvements and new features were added to make comparing tree topologies easier to use. Figure 9.3 shows a new rerooting feature, which allows to select a set of terminal nodes and automatically calculates a rooting branch that separates the selected nodes from as many of the remaining terminals as the topology allows. If trees resulting from different analyses are compared using the interactive comparison feature, differences are easier to inspect, when all compared trees have similar rooting points. Since the topologies from the resulting analyses may differ, similar manual rooting may not be trivial in some cases and an automatic feature to find the best matching rooting points in all trees improves the visualization of topological differences and to allows to distinguish them from “rerooting artifacts”.

Besides different rooting points, a different order of nodes within subtrees can also hinder the visual inspection of topological differences. (Note that changing the order of nodes under their direct parent node does not change the topology

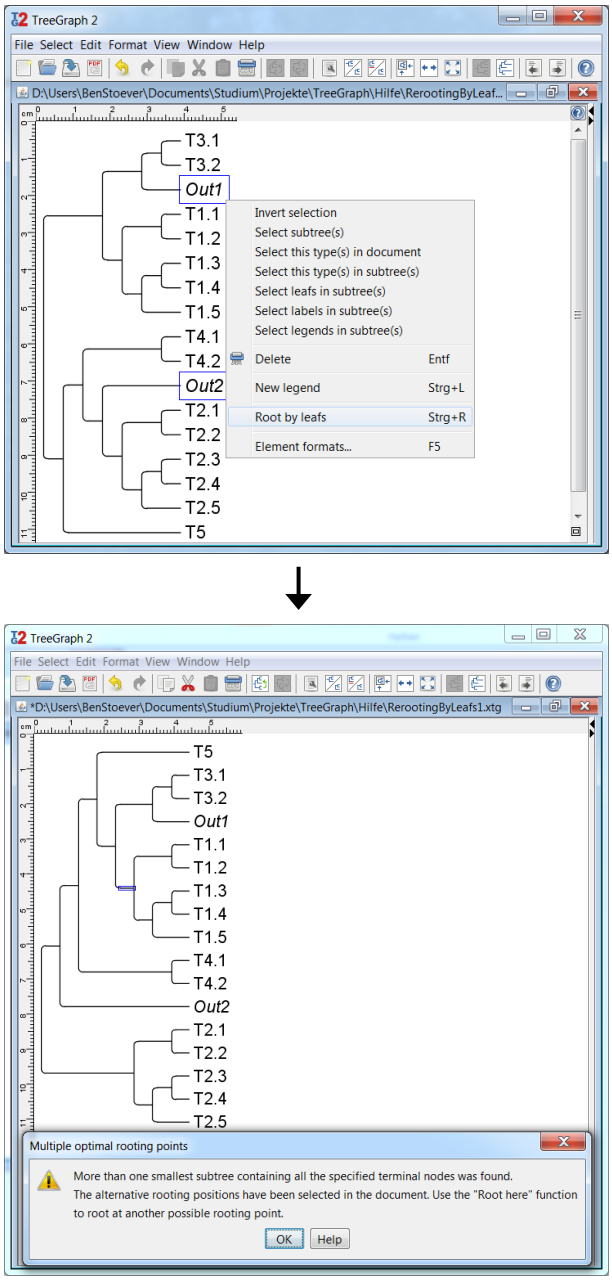


Figure 9.3 Rerooting by a set of terminal nodes

A set of terminals (e.g., an “outgroup” of a phylogenetic analysis) can be selected and TreeGraph 2 *provides the option to automatically reroot the tree in a way that would separate the selected taxa from the rest of the tree as far as the topology allows it.*

In some cases, multiple equally optimal rooting points may occur. TreeGraph 2 *will then chose one of these points for rerooting and select the alternatives so that the user can optionally reroot there using the usual rerooting feature at a specific node.*

(Note the example shown here was selected to show a situation with alternative rooting points. Real world examples will often have less difficult topologies, allowing a clearer separation between the outgroup and the other terminals, as achievable here.)

of a tree.) To address this, an additional new feature was added to *TreeGraph 2* that automatically sorts the terminal nodes of a tree by a defined order as far as topological restrictions allow this. The order can either be defined by another tree or a text file. Algorithm 9.1 describes how ordering is performed, while Figure 9.4 contains a concrete example, including a topological conflict. Both the new rerooting and the sorting leaf nodes features are especially useful for automatically processing trees from multiple alternative analyses to prepare them for interactive tree comparison (as described above).

Algorithm 9.1 Sorting terminal tree nodes by a defined order which may be in conflict with the tree topology

To sort terminal nodes as close as possible to a defined order, an average index is assigned to all internal nodes by calculating the arithmetic mean of the indices of their subnodes. (The index of terminal nodes is defined by the specified new order and the indices for internal nodes can then be calculated based in these recursively.) Both terminal and internal nodes are then sorted by their assigned new index. By calculating average index values for internal nodes, possible topological restrictions to the required positions of terminal nodes are balanced.

Figure 9.4 shows an example. See http://r.bioinfweb.info/TGSortLeafNodesImpl for the implementation of this algorithm.

Input:

- The root node of the subtree to be sorted: *r*
- A list of terminal nodes defining the favored order: *O*

Output:

- The number of nodes in the processed subtree: *n*
- The arithmetic mean of the indices the terminal nodes in the processed subtree have in *O*: $\overline{i}$

```
1  function [n, ī] := sortSubtree(r, O);
2    if r is a leaf node then
3      if r is contained in O then
4        n := 1;
5        ī := O.indexOf(r);
6      else
7        n := -1;  // Indicate that this node is node is not present in the favored order.
8      end if
9    else
10     for all children c of r do
11       [n_C, ī_C] := sortSubtree(c, O);  // Recursive call
12       if n_C ≠ −1 then
13         Attach the value (ī_C / n_C) to c as the average index;  // Each tree node in TreeGraph 2
14           // has an attribute map that allows to store feature specific information.
15         n := n + n_C;
16         ī := ī + ī_C;
17       else
18         Attach ∞ to c as the average index;
19           // Position nodes with undefined order at the end.
20       end if
21     end for
22   end if
23   Sort the children of r by their attached average index;
24   return [n_C, ī_C];
25 end function
```

The new tree comparison features are completed by an extended I/O functionality allowing to import support values from hot comments in *Newick* and *Nexus* [31] files and annotations from *TreeGraph*'s native *XTG* format or *phyloXML* [36]. (The version of TreeGraph published in 2010 was only able to import Newick and Nexus nodes names or branch lengths within the adding support values feature.)

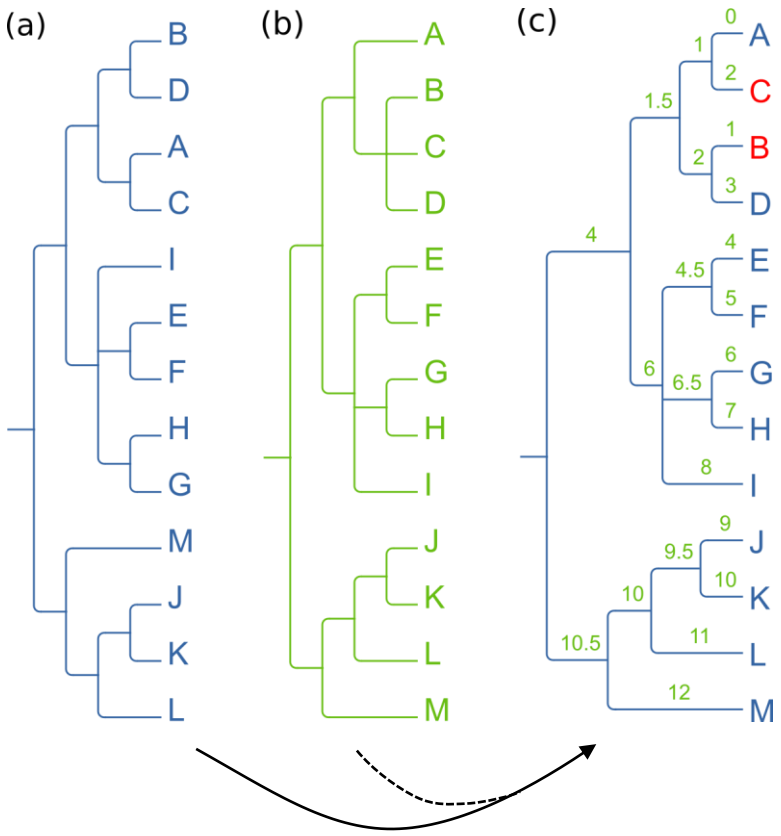


Figure 9.4 Example of sorting terminal tree nodes by a defined order using Algorithm 9.1

Trees (a) and (b) are alternatives that could result from different phylogenetic analyses and contain the same set of terminal nodes. The nodes of tree (a) were resorted using the new "Sorting leaf nodes" feature of TreeGraph 2 *to match the terminal node order of (b). The resulting tree is shown in (c). The indices as they are calculated by Algorithm 9.1 are shown in green. On the terminal nodes the indices just match the position of the respective node in (b), while the calculated average indices are shown in the internal nodes. Node that by sorting all subtrees by these indices, the closest possible leaf order to match (b) was applied. Only the positions of nodes B and C in (c) (shown in red) still differ from the positions in (b) due to topological differences.*

9.3.2 Handling data for ancestral state reconstruction

Since the last publication of *TreeGraph 2* in 2010, a set of new features was added to process and display data from ancestral character state reconstruction. To be able to display multiple support values from different analyses (see 9.3.1), *TreeGraph 2* always allowed to attach an unlimited number of labels to each branch, unlike most other tree editors. In addition to the initially supported text labels, pie chart labels have been added to display probability distributions among states of a reconstructed character. (Of course, pie chart labels can also be used to display any other type of additive data related to a tree node or branch.) Any annotations (node/branch data columns) imported into a *TreeGraph* document can be used as the source data of pie chart labels. Usually ancestral character state probabilities would be stored in hidden node data columns. (See [76] for details on node/branch data annotations.)

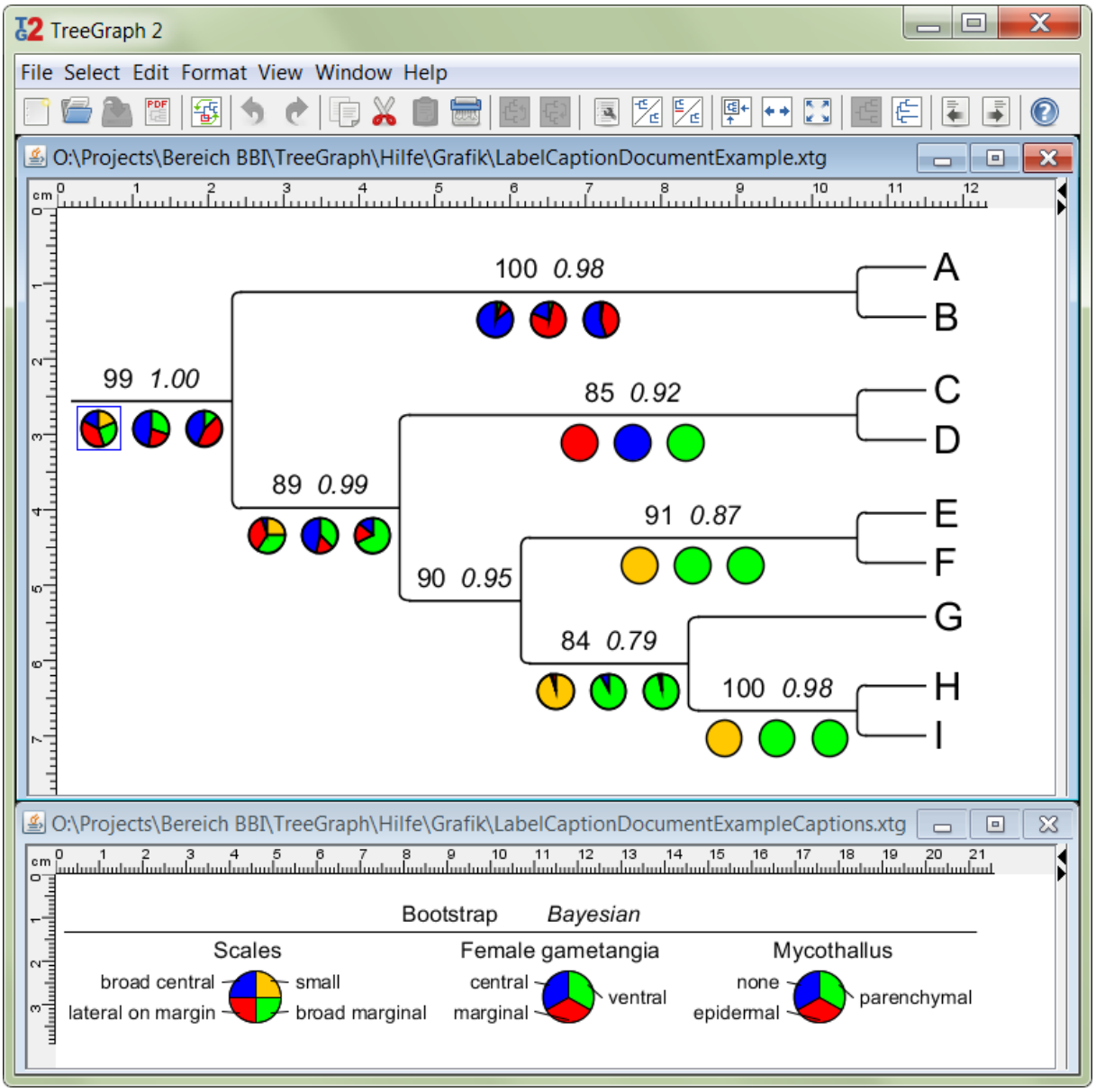


***Figure 9.5 New visualization options for ancestral character state data in* TreeGraph 2**

This screenshot of TreeGraph 2 *shows two opened documents. The upper one contains a tree carrying different kinds of labels on each node. Besides simple text labels displaying support values for the topology, there are also three pie chart labels attached to each internal branch, which display the probabilities for the states of three different morphological characters, which have been reconstructed. If respective ancestral state probabilities are loaded (e.g., as hidden node data),* TreeGraph 2 *allows to easily insert pie chart labels that visualize the probability distributions for each character on each internal node.*

The second document is a caption document used to label the tree in the first document that can be created by TreeGraph 2 *automatically. It contains one branch carrying a description for each type of label. While above the branch the two types of support values are labeled, the pie chart labels below carry a title describing the character and captions for each section of the chart that label the character state. Different caption types are available as shown in Figure 9.6.*

Optionally the pie charts may contain a title and captions for each section, which can be used directly on pie charts within a document or in a separate caption document used to label a whole tree, as shown in Figure 9.5. A special feature allows to automatically create a caption document for any existing tree. Three alternative labeling strategies for pie chart sections are offered by *TreeGraph 2* (Figure 9.6). Some of them are more useful for labeling pie charts with only a few different sections, while

others are more useful to label pie charts with many small sections. Beyond that, *TreeGraph 2* offers an alternative feature to export a table with the names of all data columns used as source data for the pie charts together with their respective section colors to a text file. Such output files allow to conveniently create captions with other applications, if more advanced or custom captions for trees are required.

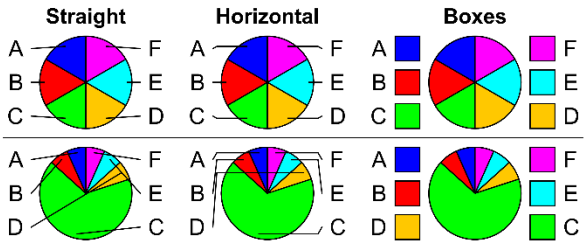


Figure 9.6 Different labeling options of pie chart labels

Pie chart labels in TreeGraph 2 *may optionally have a title and captions for all their sections. As illustrated here, three different labeling strategies are available, which have their advantages for different numbers and sizes of sections.*

While pie chart labels are the basis for displaying ancestral state data, other new features allow more convenient importing and exporting of ancestral state reconstruction related data. Besides an extended table import feature and support for additional tree formats and annotations (see 9.3.3), functionality for directly reading and writing data of *BayesTraits* [246] are now available. (See Figure 9.7.) *BayesTraits* requires all internal nodes for which ancestral character states are to be reconstructed to be defined by an input commands enumerating all nested terminal nodes. Especially for large trees where all nodes should be reconstructed, it can be very time consuming to generate such commands manually. *TreeGraph 2* offers a feature to generate terminal node definitions for selected or all nodes of a tree, which significantly simplifies running a *BayesTraits* analysis. Node definitions can either be copied to the clipboard to be pasted into the command line of *BayesTraits* or can be written to a file, if *BayesTraits* is run with a command file. If needed, *TreeGraph 2* also allows to generate the required tree and terminal state table input files for *BayesTraits* using its features to export trees to different formats and to export tree annotations to table text files. Both features now allow more customization options (see also 9.3.3, page 147) to create files in the exact syntax *BayesTraits* requires.

Another new feature allows to import the reconstructed character state probabilities from a *BayesTraits* output log file into tree annotations that can then be directly visualized in pie chart labels. This import feature is able to interpret the definitions of internal nodes in *BayesTraits* output format allowing to automatically map loaded data onto the correct internal node. If necessary, *TreeGraph 2* also automatically averages Bayesian samples of the reconstructed probabilities. Similar to generating *BayesTraits* commands, this feature saves the user a lot of time compared to importing such data by hand using manual node mapping and table reformatting.

Together, the command export and the log import features provide a seamless integration between *TreeGraph 2* and *BayesTraits*, as the workflow in the left of Figure 9.7 shows.

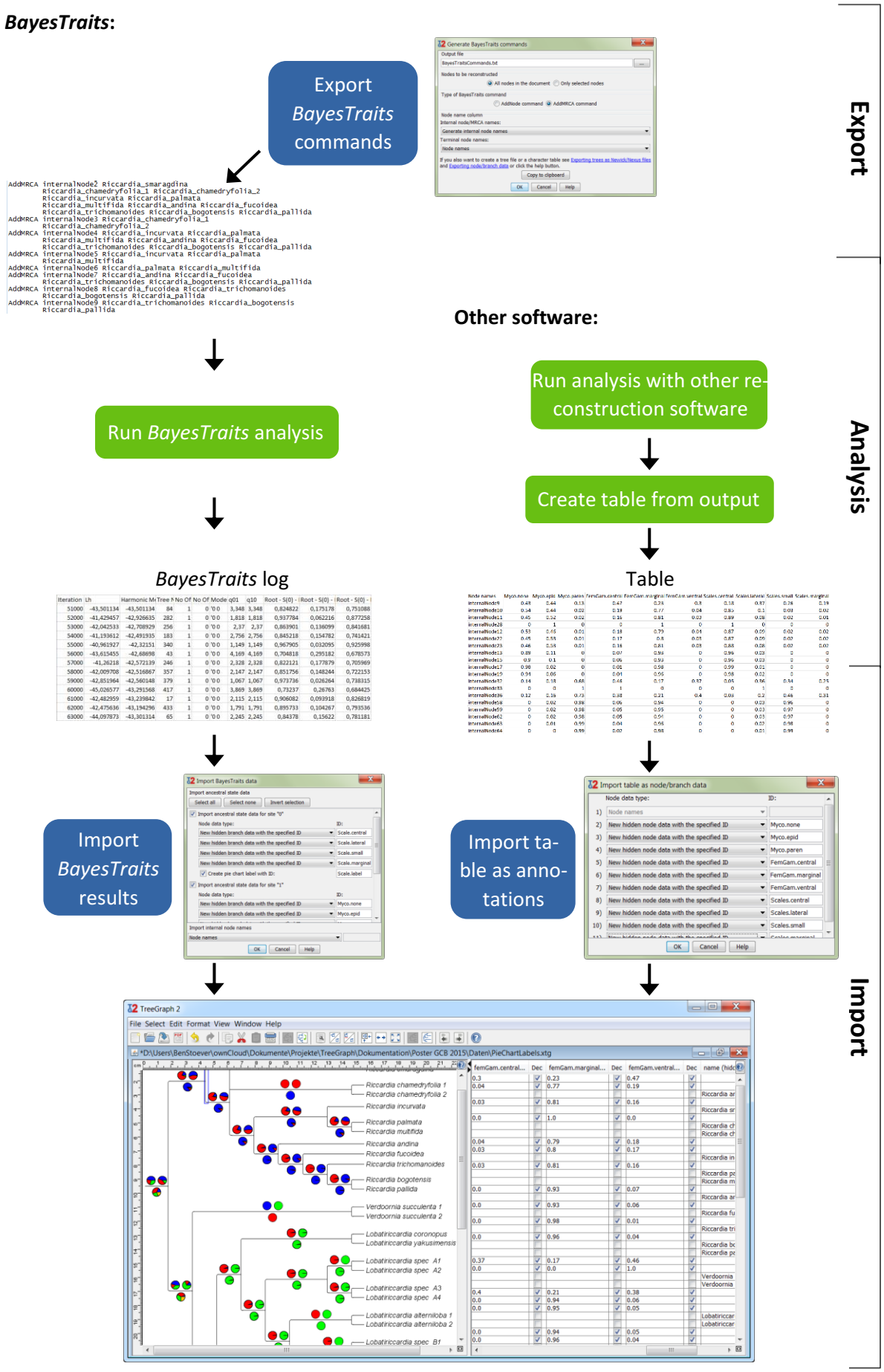


***Figure 9.7 New functionality to import and export ancestral character state data with* TreeGraph 2**
(caption on next page)

***Figure 9.7 New functionality to import and export ancestral character state data in* TreeGraph 2** *(continued)*

This diagram shows two workflows how data from ancestral character state reconstruction can be imported and visualized using new functionality of TreeGraph 2*. Actions of this workflow performed in* TreeGraph 2 *are shown in blue, while external actions are shown in green. The workflow on the left side of the figure shows how an ancestral state reconstruction using BayesTraits can be done. First* TreeGraph 2 *is used to generate node definition commands for* BayesTraits *and after running the reconstruction there the output log file is parsed again by* TreeGraph 2*.*

Although no application-specific functionality is currently present for other ancestral state reconstruction software, data can still be imported using the import table feature of TreeGraph 2, as shown on the right side of the figure. The table import feature has been extended to allow easier import as described in chapter 9.3.3 on page 147.

At the end of both workflows, the reconstructed ancestral character state probabilities are stored as annotations in the tree document and visualized using pie chart labels (see screenshot at the bottom of the figure).

9.3.3 Extended I/O functionality

TreeGraph 2 initially supported reading and writing phylogenetic trees and attached metadata in *Newick* [34], *Nexus* [31] and its native *XTG* format [247], as well as reading from *phyloXML* [36]. Current versions additionally allow to import trees and metadata from *NeXML* [35], using *JPhyloIO* (chapter 2). Unlike other phylogenetic file formats, *NeXML* includes an *RDF*-based metadata model allowing to use predicates from externally defined ontologies to link annotations, e.g., to tree nodes or branches. Such annotations may also be nested. (More details can be found in the chapters 2.2.4 or 9.3.6.) Currently, *TreeGraph 2* generates node/branch data column names from these predicates to import the linked annotations into its data model. Future versions will replace the current node/branch data column model by this *RDF*-based metadata model. This way the application will support linking metadata using predicates from externally defined ontologies directly within its interface and make full use of the functionality for format-independent reading and writing of trees with such metadata provided by *JPhyloIO*. (See chapter 9.3.6 for more information on the currently ongoing refactoring of *TreeGraph 2*'s metadata model.)

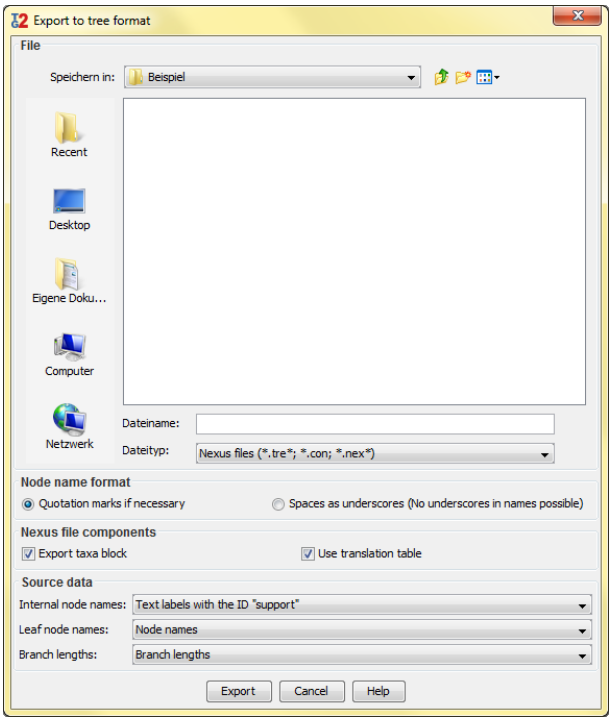


***Figure 9.8 The export trees dialog of* TreeGraph 2**

This dialog is used to export phylogenetic trees to Newick *and* Nexus *files. Below the file chooser are the new options for the format of node names and optional* Nexus *elements.*

To further improve the metadata support of *TreeGraph 2*, unnamed hot comments (as used in *Newick* and *Nexus* files by some applications) are now supported in addition to named hot comments (as used by *TreeAnnotator* of *BEAST* [237] or *MrBayes* [73]) which were already supported before. (Since *Newick* strings do not support metadata in their general definition but allow comments at any position, so-called hot comments were introduced, which contain metadata within a comment. See chapter 2.2.4 for further details.)

For exporting trees, a new option was introduced that allows to specify how spaces in node names should be treated when writing to *Newick* or *Nexus*. Some applications expect names with spaces to be within quotation marks, while others only support spaces replaced by underscores. For exporting

to Nexus, the user now additionally has the option to select whether a *Nexus* `TAXA` block or a translation table within the `TREES` block should be included in the exported file. (See [31] or chapter 2.2.3 for more details on *Nexus* blocks.) The new options increase interoperability, since other applications often support only one specific variant of *Newick* or *Nexus* and the user now has a choice and full control over which variant is exported by *TreeGraph 2*. Figure 9.8 shows the new options in the user interface.

To be able to store date related to new functionality like default node/branch data columns (chapter 9.3.1) or pie chart labels (chapter 9.3.2) the XTG format was extended multiple times. The current version 2.14.0 uses *XTG* version 1.5.

Besides reading and writing metadata from and to tree formats and handling ancestral character state data (chapter 9.3.2), *TreeGraph 2* is also able to import and export annotations from table files. The initially published version required each imported table to have a column containing special unique node names generated by *TreeGraph 2* in order to map rows of the table to nodes of the tree. If such a table was not previously created with *TreeGraph 2* and then edited, it was very time consuming to perform manual node mapping by adding the respective unique node names to a table. To solve this problem the import table feature does not require a column of unique node names anymore but allows any node/branch data column in the tree (e.g., node names, scientific taxon names or lab codes) to act as the key column, which can be selected on import. This way mapping table rows to tree nodes can be performed using any column of the table together with any node/branch data column in the tree containing the same values, which makes this feature much more flexible and easy to use.

Exporting annotations to table files now additionally allows including column headings and selecting columns to export from a large set of available columns has been made more convenient in the graphical user interface.

For exporting trees to the vector graphic format *SVG*, *TreeGraph 2* now allows to specify if texts should be exported as shapes or texts, allowing more flexibility when, e.g., fonts are used that may not be present on other systems.

9.3.4 New ways to calculate metadata

Managing metadata attached to nodes and branches in a phylogenetic tree is a key aim of *TreeGraph 2*. Besides the new functionality for visualizing, reading and writing metadata described in the previous chapters, functionality to calculate annotations from other annotations has also been extended. The version of *TreeGraph 2* published in 2010 already offered a feature to calculate numeric or textual values attached to a node or branch by a user-defined expression that allows references to other annotations of the same node or branch. A set of fundamental mathematical functions was available for use in such equations.

Various extensions of this “calculate node/branch data” feature have been made to date. Multiple new functions for text manipulation, like extracting or locating subsequences or converting to upper or lower case, have been added and allow to create and manipulate textual values depending on other attached metadata. Beyond additional fundamental mathematical functions that are now available, a new type of numeric functions has been introduced that takes a set of values as parameters. It can either be applied to a simple set of values directly expressed in the equation, a set of references to other attached values, or to a node/branch data column as a whole (one set of values attached to all nodes or branches of a tree). The available functions of this type allow, e.g., to calculate the mean value, the sum or the maximum of a set of values.

Topological functions, like `isRoot()` or `indexInParent()` allow to perform calculations that take the topological position of each node into account, while the new function `getParentValue()` allows

to make references to metadata attached to parent nodes instead of just allowing references to values attached to the node or branch that is currently calculated.

Another addition to the feature that enables new use cases is the dynamic calculation of the target column. The previously published version of *TreeGraph 2* allowed to statically specify one existing or new node/branch column into which the calculated values for each node or branch were stored. The current version alternatively allows to actually calculate the name (ID) of the target column. To achieve this, two expressions can be specified when the "calculate node/branch data" operation is executed; one is used to calculate the result and the other to calculate the name of the column to store it. The calculated name may refer to an existing column (to overwrite the current value) or a new column. The type of new columns (text label, hidden node data or hidden branch data) can be specified. The screenshots on the left of Figure 9.9 show the two text fields to enter both expressions and the options the select the type of new columns in the respective dialog. (Note that the option to calculate the target column is not used in the concrete example shown in the figure.) Dynamically calculating the target column for each node or branch allows to store values in different columns depending on the attached metadata or the topological position of a node or branch. This allows to solve new types of problems using this feature that could not be addressed before. (Follow the link given below for a concrete example.)

Figure 9.9 shows a usage example of some of the new functionality using `getParentValue()` and the maximum function applied to a whole node/branch data column to calculate the age of all internal nodes of a tree. An operation like this would not have been possible without the extensions made to the "calculate node/branch data" feature. The full documentation of all new functionality can be found at http://r.bioinfweb.info/TGCalc, including an additional usage example showing how to use the functionality to calculate the target column as described above to generate source data to be displayed with pie chart labels (cf. chapter 9.3.2).

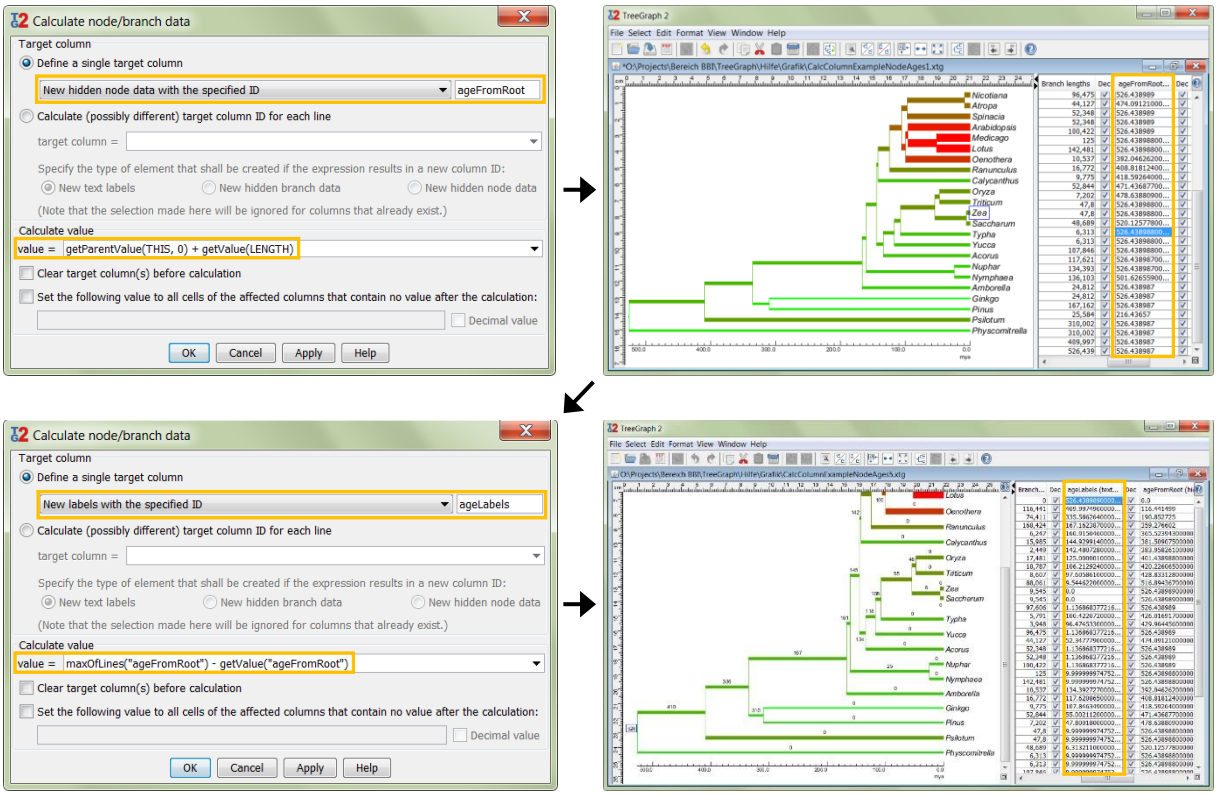


***Figure 9.9 Example usage of the extended calculate node/branch data feature to calculate the age of internal tree nodes with* TreeGraph 2**

This example shows how the extended feature for calculating node/branch data can be used to calculate and display the ages of internal nodes from branch lengths that define the time passed along them in two steps. Key elements of the screenshots are highlighted in orange.

In the first step (screenshots in the upper row) a new hidden node data column with the name "ageFromRoot" is calculated using the expression shown in the upper left screenshot (both highlighted in orange). For each node, TreeGraph 2 *calculates the sum between the value of the column "ageFromRoot" that was previously calculated for the parent node and the length of the branch leading to the current node. The function* `getParentValue()` *takes two parameters. The first is the name of the node/branch data column, while in this example the keyword* `THIS` *is used to indicate that the column that is currently calculated should be used. The second parameter is a default value that should be returned if no parent node is available, which happens when calculating the value for the root.* `getValue()` *is a similar function that returns the value of another node/branch data column on the currently calculated node. Its only parameter is again the name of the column, while the keyword* `LENGTH` *used here specifies that the respective branch length should be returned.*

In the second step the actual age of each node is calculated (screenshots in the bottom row). This time the target column to be created is a text label column with the name "ageLabels", as shown in the bottom left screenshot. The difference to the values calculated in step one is that the time should now be measured from terminal nodes and not from the root. (The root should be the oldest node, while the terminal nodes exist in the present, representing recent taxa.) To achieve this the previously calculated "ageFramRoot" value of each node is subtracted from the maximum value of all nodes using the expression shown in the dialog under (b). The maximum value for "ageFromRoot" in all nodes is the time that passed when moving from the root to any terminal node and is calculated by the `maxOfLines()` *function, which takes the source column as its only parameter. Since the target column was a set of text labels, the node ages are directly displayed above each branch as shown in the bottom right screenshot. Note that calculating the age in two steps was necessary, since calculations are always performed from the root to the terminals of a tree.*

The full description of this and other examples, including downloadable example files can be found under http://r.bioinfweb.info/TGCalcExamples.

9.3.5 Additional new features and improvements

Besides the new and extended features described above, some general improvements compared to the initially published version were made. The text output in tree documents on the screen and when exporting to pixel graphic files is now anti-aliased and more precise, avoiding artefacts resulting from different types of fonts.

A new feature allows to automatically collapse internal nodes to polytomies depending on their support value. This feature is useful when only nodes with a minimal support should be shown in a figure and saves time especially for large trees. Other topological features like manually collapsing single nodes, separating branches and rerooting now preserve branch length information.

Editing the text or captions of nodes or labels is now possible for multiple elements at the same time, which is, e.g., useful of the caption or title for a set of pie chart labels should be changed.

9.3.6 Ongoing extension of the metadata model

As already briefly mentioned in 9.3.3, future versions of *TreeGraph 2* will use a metadata model based on the *Resource Description Framework* (*RDF*) (as it is also supported by *NeXML* [35]) as a replacement for its current node/branch data column model (cf. explanation in chapter 9.3.1).

9.3.6.1 Status

Implementation of support for the new metadata model is currently ongoing in a separate development branch. The data structure of *TreeGraph 2* has already been fully adopted and a concept to combine the old column-based metadata attachment with the new nested *RDF* statements (see Figure 9.10) has been developed and implemented. The graphical user interface still needs to be extended to allow direct editing of *RDF* metadata attachments. Editing would currently only be possible directly in processed *NeXML* files. After the GUI-extension is completed as well, the new functionality will be incorporated into a regular release of *TreeGraph 2*. Details on that can be found in chapter 9.3.8.1. Source codes of the latest version with the new metadata model are available at http://r.bio-infweb.info/TGMetadataBranch.

9.3.6.2 Advantages of the new model

RDF allows formulating statements about resources in a standardized and machine-readable subject-predicate-object form. Resources modeled by *TreeGraph 2* are whole phylogenetic trees, their nodes or branches and attached metadata objects. A support value could be attached to a branch using a predicate, which unambiguously describes their relation. In this case, the branch would be the subject and the support value the object. Figure 9.10 shows a set of examples for such *RDF* statements. The branch leading to the subtree containing “Taxon 1” and “Taxon 2”, e.g., carries the support value “92” that is linked using the predicate `sup:maxLikelihood`.

So far, this may not seem very different from the previous metadata model of *TreeGraph 2* that just used any textual key (i.e., the node/branch data column heading) to describe the connection between a node or branch and a metadata value, but the new model has two important advantages. First, an *RDF* predicate is not a simple string, but a globally unique *URI* with an externally defined meaning. A set of predicates together with their meanings form an ontology. Different ontologies exist both inside and outside of life science and define terms relevant for a certain field or application. Table 9.1 (page 155) contains examples for ontologies useful to link metadata expected to be processed with *TreeGraph 2*. Compared to the current node/branch data column model, *RDF*-predicates from externally defined ontologies allow to unambiguously and machine-interpretable characterize the relationship between attached metadata and nodes or branches of the tree. This increases accessibility of data and reproducibility of respective studies, also by allowing automatically searching and interpreting files produced with *TreeGraph 2*.

The second advantage over using simple string keys is that the new *RDF*-based model allows nesting annotations, e.g., to group metadata. Figure 9.10 shows an example where different taxonomic information is nested under an anonymous internal resource metadata node. That internal metadata node is linked to a node of the phylogenetic tree using a predicate describing taxonomic information, while the nested genus and species names are linked to that internal node using specific genus and species predicates. (Refer to the figure description for further details.) The option to nest metadata allows to describe its relations more precisely and to present larger amounts of metadata in a clear way. (Note that we use the term phylogenetic tree here to differentiate the actual tree opened in *TreeGraph 2* from the *RDF* trees used to attach metadata to single branches or nodes, as shown in Figure 9.10. Anyway, everything described here is not limited to be used with phylogenetic trees but can also be applied to all other types of trees that may be edited with *TreeGraph 2*.)

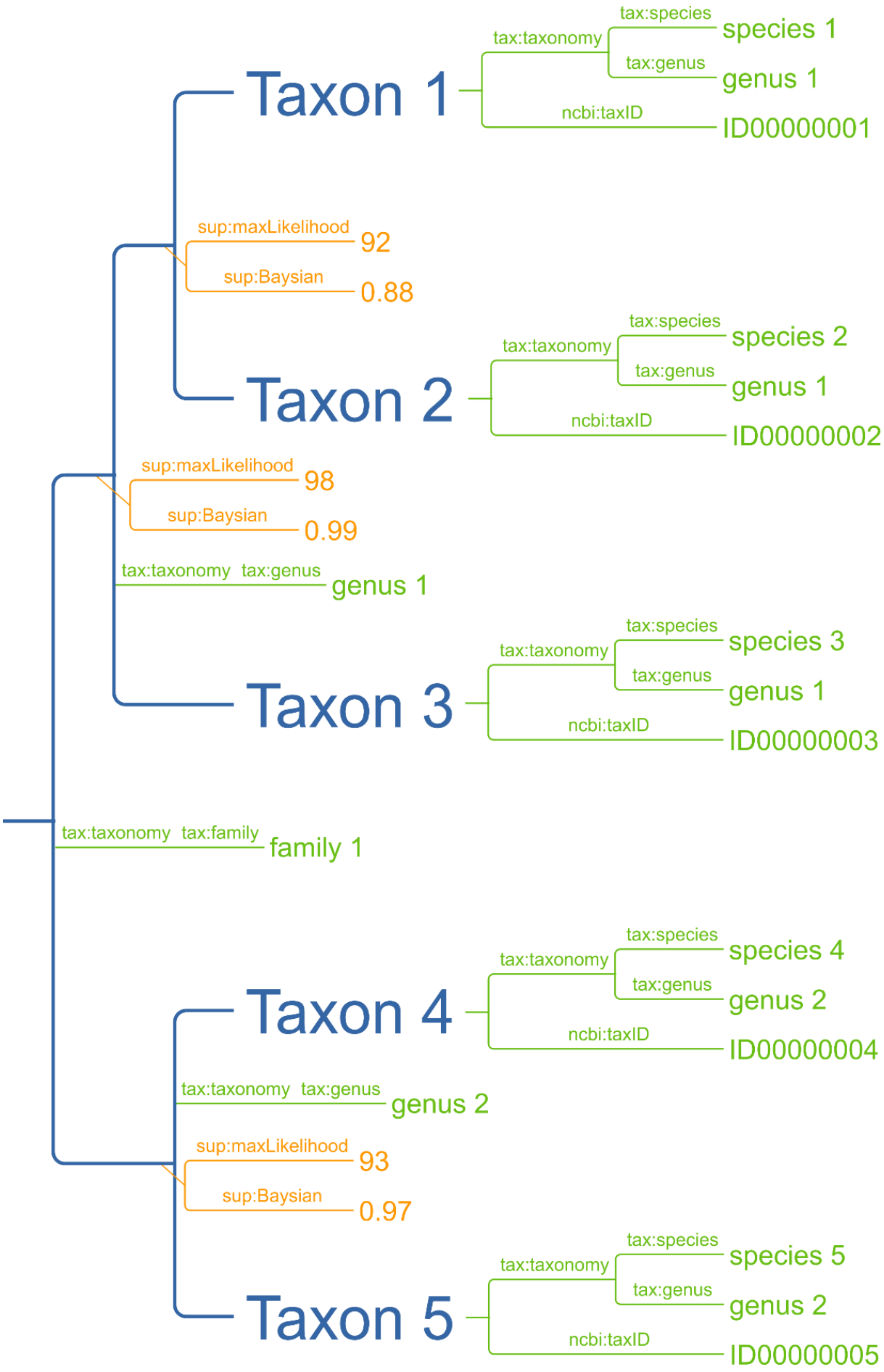
Although predicates from externally defined ontologies have the advantages described above and would allow to sufficiently describe any information on the relation between metadata elements and elements of a phylogenetic tree, the new metadata model of *TreeGraph 2* will still allow to additionally specify free strings as column names. This combination was chosen for the following two reasons. First, the new data structure must support loading *XTG* files created with previous versions of *TreeGraph 2* to ensure backward compatibility to the old node/branch data column model. There is no automatic way to interpret the meaning of freely user-defined column titles of the old model to convert them into predicates of existing ontologies. Second, users may need to describe relationships that are not yet modeled by any existing ontology. In fact, there is still a need for extending existing ontologies to offer predicates to attach common phylogenetic data to tree nodes and branches. In such cases, we cannot force users to first define a formal ontology in order to import data into our tree editor. That being said, it should be noted that data attached using free string keys instead of formally defined predicates will not benefit from any of the advantages regarding accessibility and reproducibility as described above. As the number of available ontologies in life sciences increases and the existing ones are extended, cases where user-defined string keys are the only way to describe a relation should become less frequent in the future.

This new metadata handling functionality is an important step to address demands of the age of omics, big data and the semantic web. Manually inspecting and collecting relevant data, e.g., for metastudies, may often be impossible for researchers, due to its sheer volume, which makes meaningful and unambiguously annotating biological data a central task for the future to allow automatic data retrieval and interpretation. To facilitate reuse of phylogenetic trees and attached data, user-friendly technology for annotating is required [22], which in our opinion needs to include tree editors. When the remaining GUI extensions are completed, *TreeGraph 2* would be the first tree editor to address these needs by offering full support for *RDF*-based annotations from externally defined ontologies and making full use of the *NeXML* format.

Table 9.1 Biological ontologies with potential use modeling the attachment of metadata using TreeGraph 2

This table lists examples for externally defined ontologies that can be relevant for users of future versions of TreeGraph 2*, since they provide predicates that can link data to nodes and branches or trees as a whole. It is an edited and extended version of Table 2 from the Bachelor thesis of Phoebe Brech, which was supervised by the author of this thesis [248]. In the area column G refers to "general" and B to "biological".*

Ontology	Purpose	Area	Citation	URL
Dublin Core	Can be used to, e.g., reference publications.	G		http://dublincore.org/documents/dc-rdf/
Biological Collections Ontology	Describes biodiversity data including museum collections and ecological surveys	B	[249]	https://github.com/tucotuco/bco
Comparative Data Analysis Ontology (CDAO)	Describes concepts and relations relevant to evolutionary comparative analysis	B	[250]	https://github.com/evoinfo/cdao
EDAM	Describes bioinformatics concepts, including types of data and file formats	B	[251]	http://edamontology.org
Environment Ontology	Describes biomes, habitats, environmental processes for genomic and microbiome-related studies	B	[252]	http://environmentontology.org/ https://github.com/EnvironmentOntology/envo
Gene Ontology	Describes genes and gene functions	B	[253,254]	http://www.geneontology.org/
Genotype Ontology	Describes the level of genetic variation in genotypes	B		https://github.com/monarch-initiative/GENO-ontology/
MIAPA Ontology	Formalizes annotation of phylogenetic data per the MIAPA metadata reporting standard	B	[40]	http://www.evoio.org/wiki/MIAPA
Phylogenetic Ontology (PHYLONT)	Ontology for phylogenetic analysis	B	[255]	https://bioportal.bioontology.org/ontologies/PHYLONT
Phylogenetics Ontology (PHAGE)	Generally models the steps of a phylogenetic analyses	B		https://bioportal.bioontology.org/ontologies/PHAGE
Plant Ontology	Links plant anatomy, morphology and development to genomics data	B	[256]	https://github.com/Planteome/plant-ontology
Protein Ontology (PRO)	Describes proteins and their relation to one another	B		http://proconsortium.org/


Taxon 1
tax:species
species 1
tax:taxonomy
tax:genus
genus 1
ncbi:taxID
ID00000001
sup:maxLikelihood
92
sup:Baysian
0.88
Taxon 2
tax:species
species 2
tax:taxonomy
tax:genus
genus 1
ncbi:taxID
ID00000002
sup:maxLikelihood
98
sup:Baysian
0.99
tax:taxonomy tax:genus
genus 1
Taxon 3
tax:species
species 3
tax:taxonomy
tax:genus
genus 1
ncbi:taxID
ID00000003
tax:taxonomy tax:family
family 1
Taxon 4
tax:species
species 4
tax:taxonomy
tax:genus
genus 2
ncbi:taxID
ID00000004
tax:taxonomy tax:genus
genus 2
sup:maxLikelihood
93
sup:Baysian
0.97
Taxon 5
tax:species
species 5
tax:taxonomy
tax:genus
genus 2
ncbi:taxID
ID00000005

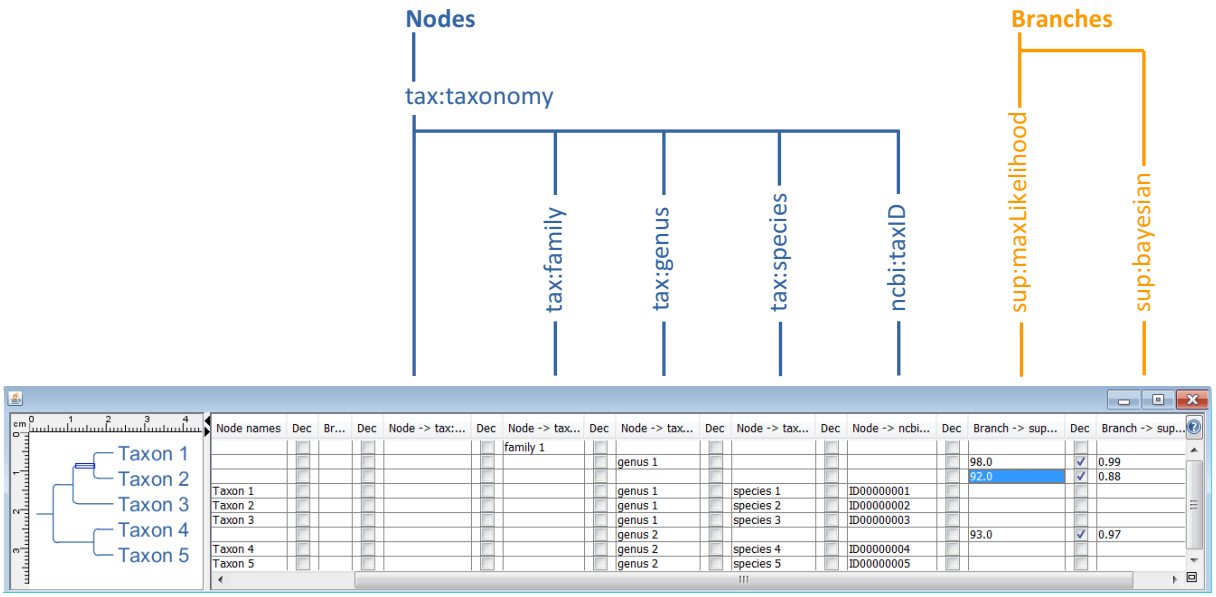


***Figure 9.10 The* RDF*-based metadata column model currently implemented for future versions of* TreeGraph 2**

The left part of this figure shows an example of a phylogenetic tree and its attached metadata as it is modeled in the current development of TreeGraph 2*. The right part shows how this data from the single* RDF *trees is combined and displayed in a node/branch data table.*

On the left, a phylogenetic tree is shown in blue, while the green trees attached to its nodes and the orange trees attached to its branches are the RDF *metadata trees. Attaching metadata to nodes or branches makes a topological difference if trees are rerooted. (Note that the green and orange trees are used in this figure to illustrate how metadata is modeled but it will not be displayed this way in by* TreeGraph 2*.) Metadata is attached using* RDF *predicates from three fictitious name spaces (`tax`, `sup` and `ncbi`) that are displayed above the branches. All predicate URIs in this figure are represented as CURIEs (compact representations of URIs), which consist of a namespace part and a local part. The predicate `sup:maxLikelihood`, e.g., has the local part `maxLikelihood` and its namespace is defined by `sup`, which is a local name for a namespace that needs to be declared within each document. If it would refer to http://example.com/supportValues/, the CURIE `sup:maxLikelihood` would be equal to the URL http://example.com/supportValues/maxLikelihood.*

The nesting of metadata is illustrated by the respective tree topologies. Each terminal node has genus and species names attached, which together are grouped within an anonymous resource metadata element. This resource metadata element is in turn connected to a respective tree node using the predicate `tax:taxonomy`. In general, the nodes and branches of the phylogenetic tree (blue) act as the RDF *subjects to which literal or resource metadata is linked using predicates. Resource metadata may then itself be the subject of a nested* RDF *statement as in the mentioned example of the taxonomic information.*

The screenshot on the right shows the tree and its metadata table from Figure 9.10 opened in the current development branch version of TreeGraph 2*. A column in the node/branch data table in uniquely identified by the path through a metadata tree. An element of such a path consists of a predicate and its index, if the same predicate is used multiple times below the same subject.*

9.3.7 Comparison to other tree editors

The following subchapters provide an overview on alternative available software that is able to perform tasks similar to the different new features of *TreeGraph 2* and how it differs. Comparing the general functionality of *TreeGraph 2* that was already available before the last publication in 2010 is beyond the scope of this chapter. Such a comparison can be found in chapter 8 (page 128, [76]) or in more recent publications of some other tree editors.

9.3.7.1 Interactive tree comparison

Multiple tools for comparing and visualizing differences between alternative phylogenetic trees have been developed in the past and focused on different aspects. *TreeJuxtaposer* [257] was published in

2003 and focused in the visual comparison of large phylogenies and displays branches of selected subtrees in different colors to illustrate the distribution of nodes in a subtree of a tree *A* over the topology of a tree *B*. It does not seem to be maintained anymore and requires an old version of the *Java* library *JOGL* to run, which is not included in the download and not available from the *JOGL* download page anymore. In addition, the recommended *Java webstart* application has not been adjusted to recent *Java* security requirements and is therefore difficult to start and then still missing the *JOGL* dependency.

Compare2Trees [258] was published two years later and can compare the topologies of two different trees, while the similarity of two internal branches is visualized by their width. The wider a branch is, the more differs its subtree from its most similar branch in the compared tree. Its accessibility suffers from the similar issues as described for *TreeJuxtaposer*.

ViPhy [259] allows to topologically compare more than two trees simultaneously as *TreeGraph 2* also does. In addition to highlighting selected corresponding nodes in the different trees, it provides additional comparison features like a global pairwise tree distance matrix and comparison score distribution diagrams. Its documentation is though very limited, leaving it unclear whether different sets of terminal nodes are supported.

Phylo.io [260] (not to be confused with the I/O library *JPhyloIO* described in chapter 2) is a recent tree comparison software that was developed in parallel with the interactive comparison feature of *TreeGraph 2*. It is a web application that focusses or comparing large phylogenies similar to *TreeJuxtaposer* but collapses subtrees to triangles and indicates their topological similarity by their color. It offers features for rerooting and sorting nodes to simplify visual comparison, similar to the functionality *TreeGraph 2* provides (see chapter 9.3.1).

Visual comparison of a very large set of topologically similar trees can be done using *DensiTree* [261]. It draws all compared trees (e.g. samples from a Bayesian phylogenetic inference) half transparent in top of each other to provide an overview of the distribution of different topologies in a set.

TreeVersity [262] was developed to compare changes in all types of hierarchical data visualized as a tree over time. Its main applications lie outside of phylogenetics and biology and its focus is to visualize the changes of numeric values attached to a tree node (e.g. funding of different departments and their substructures). Although comparing topologies of phylogenetic trees and maybe even attached numeric values of biological relevance would be principally possibly, at least with its first version, its capabilities of comparing topological changes are limited as it does only recognize deletions and insertions of nodes and does not model node movement, which is a key aspect when comparing alternative phylogenies and is the focus of the comparison functionality of *TreeGraph 2*. *TreeVersity 2* [263] focuses even more on comparing changes of attached values, rather than tree topologies and therefore has only very limited overlap with *TreeGraph 2*.

CompPhy [264] is an online workbench helping researchers to work together on phylogenetic trees. Different users can add and modify trees simultaneously and the system displays the differences based on a consensus tree. Since the focus is on providing a shared working environment for distributed team members and not mainly in the tree comparison, the visualization options are more limited than in the other software mentioned here.

The main goal of the new interactive comparison features of *TreeGraph 2* is to incorporate its existing algorithm to combine support values from different analyses while showing topological conflicts into an interactive feature. None of the other software mentioned here takes support values into account when comparing trees. Beyond that, most of the other applications are specialized on tree comparison and none of them is a fully functional tree editor. As a result, other data attached to nodes and

branches and the different ways of visualizing it, which are available in *TreeGraph 2*, cannot be displayed by the other software together with the comparison. The variety of tree formats supported by *TreeGraph 2* also makes comparing different trees easier, while none of the other tools supports the same number of formats, most of them support only one. Most of the tools are able to compare only two trees as the same time, while *TreeGraph 2* allows multiple trees to be incorporated in an interactive comparison.

Software like *TreeJuxtaposer* and *Phylo.io* has advantages in comparing large trees, while using it with small or medium-sized trees can be inefficient in some cases, since less terminal nodes are shown on the screen at the same time than in *TreeGraph 2*, due to the dynamic collapsing functionality of *Phylo.io* or the focus on displaying the names of few selected taxa in *TreeJuxtaposer*. As mentioned in its 2010 publication already, *TreeGraph 2* does not focus on displaying very large trees on a small space, but on providing advanced functionality for visualizing trees in detail. A similar principle applies to *DensiTree*, which is very useful for getting an overview on topological differences within a large set of trees but is less useful to inspect differences between a smaller number of trees in detail.

9.3.7.2 Handling ancestral state reconstruction data

To the best of our knowledge, there is currently no other tree editor available that can read the output of an ancestral state reconstruction performed with *BayesTraits* [246] and display an unlimited number of character state distributions on every branch. *BayesTrees* [265] is an application designed to handle Bayesian samples of trees and is able to export node definitions for *BayesTraits*, but cannot read or visualize the results of a *BayesTraits* analysis. It also lacks functionality to export node definitions for a whole tree in one step, as *TreeGraph 2* offers them, but instead requires the user to double click every tree node to be reconstructed.

Mesquite [111] is able to visualize ancestral character state distributions as pie charts on every node if its “Balls and sticks tree form” is selected. Alternative “tree forms” exist only for visualizing discrete states for each node. Displayed character state data need to be reconstructed with another module of *Mesquite* before, while importing data from *BayesTraits* is not supported. Only a single pie chart can be displayed on each internal node at a time, while *TreeGraph 2* is able to display an unlimited number of different character state distributions in combination with other labels on each branch.

PhyD3 [266] is a recently developed tree viewer that now also supports displaying pie chart labels based on data stored in *phyloXML*[36], but does not support reading or writing *BayesTraits* data or importing general metadata from *NeXML* [35], which could be displayed using pie chart labels. (Like *TreeGraph 2*, *PhyD3* also provides a set of other visualization options for metadata other than ancestral character states, but provides, e.g., fewer editing options.)

WARACS [267] is collection of *Python* scripts that make use of *TreeGraph 2* to visualize ancestral state reconstruction data obtained using *Mesquite* and *BayesTraits*. It has been developed independently at the same time as the *BayesTraits* import feature of *TreeGraph 2*. It does not offer import functionality for *BayesTraits* that is not also offered by *TreeGraph 2* and is not able to create node definitions, since it has no graphical interface. It still is a relevant complement, since it allows to import data reconstructed using *Mesquite* into *XTG* files that can then be further processed using *TreeGraph 2*.

9.3.7.3 Extended metadata model

Although the standard version of *Mesquite* does not support *NeXML* [35], a plugin can be downloaded that allows to read and export the format. When a *NeXML* file is opened, it is first converted to *Nexus* by the plugin and then loaded into *Mesquite*, while literal metadata annotations of nodes and branches are converted into hot comments. There seems to be no way to display or edit the metadata within the tree viewer of *Mesquite* and when exporting an opened *NeXML* file again, the metadata is not

contained in the *NeXML* output anymore. Nested annotations seem to be ignored during the import already.

Dendroscope [87] uses *NeXML* as its main format, but does not model attached metadata in GUI. When writing files, it uses the meta tags of *NeXML* to store application-specific formats, but its output is not fully compliant to the *NeXML* standard as also stated in the *NeXML* website, e.g. because different format values are combined in a single string literal metadata annotation.

Other tree editors, like *Archaeopteryx* [268], *TreeViewJ* [244] or *iTOL* [269] support *phyloXML* [36] and are able to display and edit different amounts of its predefined annotations. *PhyD3* [266] additionally introduces a set of custom tags that it can visualize in addition.

Although APIs exist that allow to visualize phylogenetic trees and read *NeXML* (e.g. *DendroPy* [99], *jsPhyloSVG* [270] or *ETE* [122]) and can handle metadata with data structures of the respective programming languages, these are libraries to develop scripts and applications and are not a replacement for a stand-alone tree editor that directly allows biologists to process phylogenetic trees and respective metadata.

None of the currently available tree editors allows to freely attach metadata to nodes and branches of phylogenetic trees, making use of externally defined ontologies and the *RDF* standard. Applications supporting only metadata from *phyloXML* are limited to the types of data explicitly modeled by the format (or some defined extensions in *PhyD3*) and the few programs that support *NeXML* do not make significant use of available metadata or model it in their GUIs. Even the metadata model of currently available versions of *TreeGraph 2* goes already beyond the capabilities of most other tree editors as its *NeXML* import functionality is already able to handle metadata ignored by the other available applications. The extended metadata support, as described in chapter 9.3.6, will be a significant advantage for the phylogenetic community, as there is currently no software available that allows user-friendly annotation to facilitate reuse of phylogenetic trees and attached data for automatic data retrieval and interpretation.

9.3.8 Future development

We will continue to develop and maintain the application in the future, while the help of new developers is welcome, who may now also contribute to the source repository via *GitHub*. (See chapter 9.5.) Besides the completion of implementations related to the new metadata model described in chapter 9.3.6, the future development may additionally focus on two major aspects: Increasing scriptability and integrating the functionality of *TreeGraph 2* with that of *PhyDE* (chapter 6) to create a complete phylogenetic workbench. All three aspects will be described in further detail below.

9.3.8.1 Implementing remaining components to release the new metadata model

As described in chapter 9.3.6 the metadata model of *TreeGraph 2* is currently refactored to support meaningful predicates from externally defined ontologies to attach metadata to trees and their nodes and branches. In order to integrate the new functionality from the development branch into the main branch and release it, respective GUI components to visualize and edit metadata attachments remain to be implemented. As it was shown in Figure 9.10, the *RDF* based metadata model will still allow to use the previous node/branch data IDs (metadata column titles) in combination with *RDF* predicates to ensure backwards compatibility and to allow handling of metadata which cannot be described by currently available ontologies sufficiently.

We are of course aware of the risk that users may continue to use these custom column titles instead of meaningful predicates from available ontologies because they do not know about them or because they do not want to invest additional time into proper annotation, if both options are provided as

alternatives. While considering proper annotation (to increase data accessibility and reuse) as important is a general issue that would have to be addressed by the scientific community as a whole (e.g., by databases, journals or funding agencies requesting metadata annotations following certain minimal standard), we will try to address these problems by making the access to existing ontologies as easy as possible from within the GUI of *TreeGraph 2*. The key component for this is the node/branch data input component that will be extended in order to complete the basic feature set of the metadata model development branch. The current public release of our tree editor allows users to specify metadata columns in various dialogs (e.g., when importing tables or annotations of other trees) by selecting between text labels, hidden node or branch data and entering a column title (node/branch data ID). The extended GUI component for specifying metadata columns would additionally allow to enter one or more *RDF* predicates formally describing the relation of the metadata to the phylogenetic tree. (The set of predicates correspond to the path from the root to an attached value in an *RDF* tree as shown in the examples in Figure 9.10.) The key feature would be content assistance allowing the user to directly select predicates from available ontologies instead of entering them manually. These proposed predicates might already be filtered or ordered, e.g., by how good they fit to the data type of the respective value. E.g., predicates related to geographical species distributions could be proposed first for values formatted as coordinates. Such a feature can be implemented similar to, e.g., content assistance in programming environments that propose variable and method names that best match the current context. This way, users of *TreeGraph 2* would automatically be made aware of existing predicates in available ontologies and inhibitions to use them would be decreased. Such a feature will also require information on available predicates of relevant ontologies, like the ones listed in Table 9.1. *TreeGraph 2* will need to be shipped with respective data and functionality to read ontology definitions from respective formats and databases. In addition, the user could be offered a search field to query the *NCBO Ontology Recommender* [38] within *TreeGraph 2* to search for relevant ontologies, which would be very simple to implement but already significantly ease up finding appropriate ontologies for the user's data.

With the completion of support for the extended metadata model, full support for reading, writing and editing of phylogenetic trees in *NeXML* with all attached metadata will be possible. To maximize interoperability, *NeXML* will then become the main format of *TreeGraph 2* and replace its current *XTG* format. Reading *XTG* files will still be support for downwards compatibility, the format will not be further developed and become deprecated.

9.3.8.2 Increasing scriptability

Since *TreeGraph 2* is open-source, it is in principle possible to develop custom Java applications using its functionality, as it is done in a currently ongoing study (chapter 13.2.2, page 186). Additionally, a number of command line parameters to convert between different tree and image formats is available that allow *TreeGraph 2* to be integrated into automated workflows. What is lacking, is a defined and stable API that provides access to the individual features of the application, especially those that make it unique, like combining support values from alternative analyses (chapter 9.3.1) or visualizing ancestral character states (chapter 9.3.2). Scripting APIs to process phylogenetic trees have become increasingly popular among biologists in recent years, as the publication and usage of, e.g., *ggTree* [271] for *R* or the *ETE* API [122] for *Python* show. Since *TreeGraph 2* is written in *Java*, the obvious first step would be to provide access to all key features with a properly versioned (cf. [103]) and documented *Java* API. This could be done by separating the current code base into API classes designed to be easily accessed by other developers and to remain stable over time and application classes, which are not of much interest for other developers and can therefore be freely changed in the future development. The API would then also have a versioning that is independent of the application versioning. This can be achieved relatively easy since only slight changes to the existing code would be necessary.

To make the features of *TreeGraph 2* available for simple use in custom scripts, the second step would be to provide wrappers for the *TreeGraph Java* API in scripting languages like *Python* or *R*, which is possible using, e.g., *Py4J* [272] or *rJava* [273]. Such APIs would make *TreeGraph 2*'s functionality available for a larger group of users and addresses the trend that an increasing number of biologists also uses custom scripts to process and create phylogenetic trees instead of directly interacting with GUI-based tree editors, especially when working with large data sets and repeated tasks. Although extending *TreeGraph 2*'s scriptability would be an important advance, it should be noted that the further development of the GUI-based application remains equally relevant. On the one hand, many users will still not be able or willing to develop custom scripts to fulfill their needs in general, and on the other hand functionality like the interactive tree comparison (as described in chapter 9.3.1) requires a GUI-based application and cannot be performed by creating an output of a script.

9.3.8.3 Integrating *TreeGraph 2*'s features into a larger phylogenetic workbench or the *Taxonomic Editor*

Typical phylogenetic data files, e.g. in *Nexus* [31] or *NeXML* [35] format, often contain phylogenetic trees together with related multiple sequence alignments and taxon lists. Handling such combined phylogenetic data sets (as they are submitted to, e.g., *TreeBASE* [44]) would be a lot easier with a unified application that allows to edit all these data types. Together, *TreeGraph 2* and *PhyDE*/*PhyDE 2* (chapter 6)/*LibrAlign* (chapter 3) provide all necessary functionality and could be combined to a phylogenetic workbench. Such a unified application would not necessarily be a replacement for the existing single applications, since it might be more complex to use for people interested in just one of the data types, and require more resources or possibly longer start times, but it could be provided as an alternative. It should be implemented in a way that all functionality from *PhyDE 2* and *TreeGraph 2* (i.e. the document windows and all actions) is automatically available, ideally without the need of any adjustments in the combined application. This way, that workbench would automatically provide the same feature set as the single applications after every release and only special functionality, like a project explorer component allowing the handle the combination of alignments and trees within a document, would have to be developed specifically for the workbench. Since both *PhyDE 2* and future versions of *TreeGraph 2* are based on *JPhyloIO* (chapter 2) and use *NeXML* as their main format, the integration of both applications would be straightforward.

Although other combined applications to display and edit trees and alignments, like *Mesquite* [111] or *MEGA* [69] are already available, having a workbench that combines the functionality of *PhyDE* and *TreeGraph 2* would still be beneficial for the scientific community, since both applications offer functionality that is not offered by alternative products and users in need for these features would have new possibilities within a combined application. The development of such a combined application is not yet scheduled for a concrete data but could start after the first feature-complete of *PhyDE 2* was release. (See also chapter 6.3.4 on page 88.)

A further step after such a workbench is available could be the integration of that functionality also into the *Taxonomic Editor* or to provide interfaces that allow the easy combination of both applications. As described in chapter 5 and [24], *LibrAlign* and *JPhyloIO* are already used by the *EDITor* to handle alignments of single reads to combine them into a consensus sequence as part of modeling the alpha-taxonomic workflow. In general, the *EDITor* models the taxonomic workflow from sample collection to generating and processing different types of data derived from these, while consistently linking all data back to the individual specimen, instead of a taxon. It currently does not offer any functionality that models the downstream parts of the taxonomic workflow, when data (e.g. sequences) from different specimens are combined to, e.g., create multiple sequence alignments and infer phylogenies. Closing this gap would allow to model the complete workflow, which is actually circular, since inferred phylogenies may influence the taxon diagnosis in the next iteration of the workflow, e.g. if a

former taxon turns out not to be monophyletic. In order to combine the functionality of the *Taxonomic Editor* with that of *TreeGraph 2* (and also *PhyDE 2*), links to the source specimens of all processed sequences and tree nodes must be modeled, which would be relatively easy to achieve with the extended metadata model of *TreeGraph 2* and *NeXML* as the exchange format.

9.4 Conclusion

The new interactive comparison features of *TreeGraph 2* (chapter 9.3.1) and the extended functionality for combining statistical support values from different analyses enable researchers to consider alternative methods for phylogenetic inference, examine their influence on the results of a study, and present a combination of all results in one figure. With a variety of different phylogenetic inference methods available, this becomes an increasingly important feature that is to date not offered by other software.

The other new features that include processing and visualizing ancestral state reconstruction data (chapter 9.3.2), extended I/O functionality (chapter 9.3.3), feature-rich calculation of annotations (chapter 9.3.4) and the new metadata model (chapter 9.3.6) all contribute to a better support of handling metadata attached to elements of a phylogenetic tree. This has been a focus of *TreeGraph 2* since its first versions, which introduced the ability to visualize an unlimited number of annotations on branches and a GUI element that displays all metadata in the editable node/branch data table next to the tree. The recent extensions do not only allow easier import, export and calculation of more types of annotations, but with the introduction of the new metadata model, *TreeGraph 2* is enabled to stay a relevant and widely-used tool in the age of big data, omics and the semantic web. When the remaining implementations (as described in chapter 9.3.8.1) for the new metadata model are completed, *TreeGraph 2* will be the first phylogenetic tree editor that fully supports annotating trees with the help of meaningful *RDF* predicates from any externally defined ontology in a user-friendly way, which will hopefully help to increase accessibility and reuse of phylogenetic data and metadata.

9.5 Availability and Requirements

Project name: *TreeGraph 2*

Project web page: http://treegraph.bioinfweb.info/

GitHub Repository: https://github.com/bioinfweb/TreeGraph2

ResearchGate project page: http://r.bioinfweb.info/RGTreeGraph2

Operating system: Platform independent

Programming language: *Java*

Other requirements: *Java* Runtime Environment 8 (or higher)

License: *GNU General Public License* Version 3 (*GPL*)

Any restrictions on use by non-academics: The restrictions specified in the *GPL* apply. (See http://treegraph.bioinfweb.info/License.)

9.6 Declarations

9.6.1 Author Contributions

Ben Stöver developed the concept for the new features, implemented large parts of the new code, wrote the manuscript and supervised a master, a bachelor thesis and a research module that focused on implementations of some of the new features. Sarah Wiechers implemented the main parts of the *BayesTraits* data import and export features during her research module and master thesis. Phoebe

Brech created most of the currently available implementations for the extended metadata model and created the initial version of Table 9.1 in her bachelor thesis. Kai Müller contributed conceptually and to the manuscript.

9.6.2 Acknowledgements

We very much appreciate the feedback from numerous *TreeGraph 2* users that helped us in identifying bugs and extending the software to address the current needs of the scientific community and Lukas Aaldering for his code contributions during a practical course supervised by Ben Stöver. Thanks to Maik Bartelheimer and Carsten Kemena for the co-supervision of two thesis that contributed to the new features of *TreeGraph 2* and Dietmar Quandt for his input in the early development phase of features to visualize ancestral character state probabilities. The work of the contributors of the open projects used is highly appreciated (*Apache Commons*, *Apache Batik SVG Toolkit*, *FreeHEP Java Libraries*, *Java Math Expression Parser*, *Tango Desktop Project*, *JUnit*, *Hemcrest*).

Part III –
General purpose software libraries and the *bioinfweb* portal

10 The *bioinfweb* portal

Stöver BC[1]*, Wiechers S[1]

[1]Institute for Evolution and Biodiversity, WWU Münster, Hüfferstraße 1, 48149 Münster, Germany

*Author for correspondence: stoever@bioinfweb.info

Own contribution

I developed the concept for the portal, implemented the required web application components, wrote the contents of the websites and maintain the server infrastructure. Sarah Wiechers contributed to the implementation of the release manager that is part of the portal.

Abstract

bioinfweb is a central web portal, where all software developed in this thesis is made available to the scientific community. It consists of a central portal and different project pages that contain downloads of the applications or libraries as well as user and developer documentation and provide access to its source codes, including a synchronization with a *GitHub* repository for each hosted project. Behind the visible portal, a technical infrastructure has been developed that automates new release by compiling the software, managing its dependencies and packing the different downloads. Creating new projects and reusing available components is also significantly simplified by the developed infrastructure.

The portal allows convenient access for all users to the software and its documentation and therefore significantly contributes to the usability of all applications and libraries developed in this thesis. It allows applying best practices for scientific open-source software development. The recent integration of social media like *ResearchGate*, *GitHub* or *Twitter* provides a greater visibility of all hosted projects and enables users to give direct feedback and to get support.

10.1 Introduction

The *bioinfweb* software web portal provides public and long-term access to all software and data developed in one of its projects, which includes all projects described in this thesis. Figure 10.1 show its logo. It was initially established in 2008 with *TreeGraph 2* as the first available software. Since then numerous other projects have been hosted here and a shared infrastructure has been built around them. Figure 10.2 shows the start of the list of available projects. This includes providing and archiving versioned downloads of all software with appropriate license information, content management systems for documentations and *JavaDocs*, issue tracking, and private and public access to versioned source codes. While all projects have access to all available features and new ones can easily be created, the single project pages are still very flexible and may largely differ from each other and still benefit from the available modules.

Figure 10.1 The* bioinfweb *logo

10.2 Release manager

The *bioinfweb ReleaseManager* is a server-based software that has been developed during this thesis by Ben Stöver and Sarah Wiechers. It allows to easily create new versions of hosted software. The release process can be triggered from a web interface and first obtains the latest source code from the *bioinfweb* repository. After that compiling and *JavaDoc* generation is performed and binary and source downloads are packed. The download and documentation sections of the respective project page are then updated. By automating this whole process, *ReleaseManager* significantly reduces the time necessary to make a release, while at the same time allowing a more convenient presentation by providing different types of downloads for binaries, source codes and documentation of the different modules of different software. It also plays a role in quality control, by making sure that all types of downloads and the documentation are always up-to-date, and no part can be forgotten to be adjusted.

Figure 10.2 Screenshot of the software list at the* bioinfweb *portal in 2017

Figure 10.3 Screenshots of different parts of* bioinfweb *project pages

The download section with the available modules for bioinfweb.commons *(chapter 11, page 168) is shown in the left, while the screenshot in the middle shows the JavaDoc of* JPhyloIO *(chapter 2, page 33) and on the right the online source code view for* TreeGraph 2 *(chapter 7, page 90) is visible.*

10.3 Social media

The source codes of all *bioinfweb* software is mirrored at *GitHub* (https://github.com/bioinfweb) to increase its visibility and allow other developers to easily contribute to the development. In addition, news on the software and information on new releases is posted (in addition to the news section of the *bioinfweb* main page) to the *bioinfweb Twitter* account (https://twitter.com/bioinfweb) and respective *ResearchGate* project pages. Each software has a separate *ResearchGate* project page where users can ask support questions and follow the project development. (An example would be the project page of *TreeGraph 2* available at http://r.bioinfweb.info/RGTreeGraph2. Other *ResearchGate* pages are linked in the navigation of the respective *bioinfweb* project pages.)

Figure 10.4 Screenshots of the* bioinfweb *social media pages

On the left, the LibrAlign *project page on* ResearchGate *is shown as an example. Additional pages exist for other projects. The* Twitter *feed on* bioinfweb *is shown in the middle, while the* bioinfweb GitHub *organization page is shown on the right.*

10.4 Aims of the portal

The aim of the portal is to allow convenient and long-term access to all project resources for users, developers and contributors. By providing this information independent of a specific university or working group web page, stable URLs can be guaranteed. The portal is planned to be maintained in the future, also if project members leave the *WWU*. *bioinfweb* projects are by choice hosted on their own portal to have maximal flexibility in the ways different types of contents are provided and to be independent of a single open source hosting provider, which might exit the market in the future. Project mirrors on *GitHub* are anyway provided but in parallel to the *bioinfweb* project pages to have more visible in the community and use the repositories there as an additional backup.

10.5 Conclusion

The *bioinfweb* software portal and its social media representations ensures the (log-term) availability of the developed software, including its documentation, source codes and additional information and allows users to give feedback and ask questions to the projects. It is a central tool to apply best practices in open-source scientific software development (e.g. rules 8-10 in [274] or rules 6-8 in [275]).

11 *bioinfweb.commons*: Shared bioinfweb components made available in a library

Stöver BC[1]*, Wiechers S[1]

[1]Institute for Evolution and Biodiversity, WWU Münster, Hüfferstraße 1, 48149 Münster, Germany

*Author for correspondence: stoever@bioinfweb.info

Own contribution

See section 11.5.1 on page 170.

Abstract

The different applications and libraries developed in this thesis often require the development of both general and bioinformatical functionality that potentially can be reused by other software. To avoid redundant implementations, a shared *Java* library, called *bioinfweb.commons* was created that is hosted as a separate project at the *bioinfweb* portal. (Versions for other languages are also in preparation.) All functionality can be reused under the terms of *GNU the Lesser General Public License*.

The library is structured into different modules, which, e.g., include compressed collections that allow the efficient handling of, e.g., large sequence and alignment datasets, mathematical and logging functionality, tool classes to interpret sequence strings and ambiguity codes, convenience methods to process *XML* and to perform unit testing, or GUI components and functionality for the *Java* toolkits *Swing* and *SWT*. Most of these features are used by the software of this thesis and have been developed together with it. It was made sure that only functionality was developed that was not available in other general-purpose and bioinformatical libraries in a suitable form.

11.1 Introduction

The numerous *bioinfweb* software projects that have been introduced to date sometimes require shared functionality. To efficiently address these needs, *bioinfweb.commons* was created, which is a *Java* library that provides general and bioinformatical tool classes and methods. All functionality that is developed for a *bioinfweb* software is checked for being of general use also for other applications and libraries. If that is the case, the respective implementation is added to *bioinfweb.commons* instead of having it directly within the actual software. This way, it is directly available not only for all other bioinfweb projects but also for all other developers, since *bioinfweb.commons* is open-source and publicly available under the terms of version 3 of the *GNU Lesser General Public License* (*LGPL*).

11.2 Modules and provided functionality

bioinfweb.commons is distributed among multiple modules for different purposes. While the core module provides functionality of very general use, the other modules provide functionality for a specific domain, so that depending software only needs to include dependencies it actually needs. Table 11.1 gives an overview on the currently available modules.

***Table 11.1 The modules of* bioinfweb.commons**

Module	Description
core	Contains implementations of very general use for all types of applications. There are e.g. classes that offer version management of applications, I/O and *XML* utility classes, efficient collection implementations, mathematical and random number methods, logging classes, text manipulation as well as system utilities.
applet	Implementations for writing *Java* applets.
bio	Bioinformatical tool classes that provide functionality for modeling and manipulating different kinds of biological sequences.
servlet	A set of helpful classes to implement *servlets*, e.g., using the *JSP model 2 architecture*.
sql	Shared functionality for working with *SQL* databases.
swing	Provides additional components and model implementations useful in *Swing* GUIs.
swt	Small module providing some additional functionality to create *SWT* GUIs.
testing	Provides functionality useful for unit testing.
experimental	Contains implementations for different purposes that are not yet fully tested or might undergo significant changes in the future. Classes contained here are not guaranteed to be remain API-stable in future releases, even if the major version number does not change.

All implemented functionality is checked to have as small overlap as possible with functionality already available in other tool libraries like *Apache commons* [276], *Google Guava* [277] or *BioJava* [93,94], while related features are usually compatible with these other libraries so that users can easily combine them.

A significant part of the available tool classes has been developed within this thesis to be used by the applications and libraries that are part of it. *LibrAlign* (chapter 3) makes e.g. use of the compressed list implementations that are available in the *core* module, the sequence utilities of the *bio* module and GUI tool classes of the *swing* and *swt* modules. The enumeration model classes of the bio module are used in *JPhyloIO* (chapter 2) to represent sequence information in its events. Since the same model classes are also used by *LibrAlign*, the interaction between both libraries is simplified. *TreeGraph 2* makes use of the available *Swing* dialog implementations and the molecular components of the use the *SWT* tools. The *Taxonomic Editor* (chapter 5), *PhyDE 2* (chapter 6) or *AlignmentComparator* (chapter 9.6.2) also make use of *bioinfweb.commons*.

11.3 Conclusion

Externalizing shared functionality needed by different *bioinfweb* software avoids redundancy and simplifies code maintenance, since all bug fixes and improvements in *bioinfweb.commons* are automatically available for all depending software. The development of future software in bioinfweb is furthermore simplified, since needed functionality is often already present. By being open-source and compatible with other tool libraries, the use of *bioinfweb.commons* goes beyond bioinfweb and is also released as a service to the scientific and open-source community.

11.4 Availability and Requirements

Project name: *bioinfweb.commons*

Project web page: http://commons.bioinfweb.info/Java

GitHub Repository: https://github.com/bioinfweb/commons.java

Operating system: Platform independent

Programming language: *Java*

Other requirements: *Java* Runtime Environment 8 (or higher)

License: *GNU Lesser General Public License* Version 3 (*LGPL*)

Any restrictions on use by non-academics: The restrictions specified in the *LGPL* apply. (See http://commons.bioinfweb.info/Java/License/LGPL.)

11.5 Declarations

11.5.1 Author Contributions

Ben Stöver developed the concept, implemented the library, and wrote the manuscript. Sarah Wiechers contributed some implementations during her master theses that was supervised by Ben Stöver.

11.5.2 Acknowledgements

Smaller code contributions by Jonas Bohn during his research module supervised by the author are highly appreciated. We thank the contributors to the open-source projects used (*Apache Commons*, *Apache Lucene*, SWT, *BioJava*, *Google Guava*, *Hemcrest*, *JUnit*).

12 *Toolkit Independent Components*: Creating GUI components for both *Swing* and *SWT*

Stöver BC[1]*

[1]Institute for Evolution and Biodiversity, WWU Münster, Hüfferstraße 1, 48149 Münster, Germany

*Author for correspondence: stoever@bioinfweb.info

Own contribution

See section 12.7.1 on page 20.

Abstract

Toolkit Independent Components (*TIC*) is a library that allows to create GUI components for the two major *Java* GUI toolkits, *Swing* and *SWT*, in one step. Abstractions are provided for painting components, resizing and scrolling them and receiving and handling mouse and keyboard events. The goal of *TIC* is not to provide access to the full component libraries of both toolkits and enabling to use, e.g., buttons or text fields, in a toolkit-independent way. The focus instead lies on simplifying the development of self-painting components for both toolkits, by providing an abstract component class with one painting method and one set of event listeners that are interoperable with both toolkits. For scrolling, special classes are provided that overcome the size limitations for subcomponents in *SWT* and allow the greater maximal component dimensions from *Swing* for both toolkits. *TIC* is especially useful for library developers that need to support both toolkits to allow a maximal reusability of their GUI components. It has been used by *LibrAlign*, which is a *Java* library that provides GUI components to display and edit sequences and multiple sequence alignments together with associated raw and metadata.

12.1 Introduction

As mentioned in chapter 3.2.2, *LibrAlign* provides all its GUI components in a *Swing* and an *SWT* version to be usable in applications based on either toolkit. *Swing* is the build-in GUI framework shipped with the *Java* virtual machine. It contains implementations for a set of basic and more advanced GUI components like buttons, text fields, tree views or tables. These components are so-called lightweight components that do not rely on the GUI components provided by each operating system but are implemented in *Java* from the scratch. Although their appearance can be adjusted to closely match the respective operating system, *Swing* components may look slightly different then their native counterparts. The advantage of *Swing* is true platform independence, allowing also complex (custom) components to work exactly identical on all operating systems.

SWT [104], on the other hand, provides heavyweight components that make use of the respective operating system libraries, including some extensions to these. As a result, platform-specific features, like file dialogs that largely differ between operating systems, are accessible using *SWT*, which cannot directly be used in *Swing*. The downside here is that different binary libraries need to be shipped with an *SWT*-based application for every supported operating system, which to some extend undermines the “compile once, run anywhere” philosophy of *Java*. *SWT* is also the basis of the *Eclipse Rich Client Platform* (*RCP*) [105] and therefore all *RCP*-based applications require *SWT*.

If a custom GUI component (e.g., to display and edit multiple sequence alignments) is needed within an application, it could easily be developed for the toolkit that the application uses, but for the development of GUI libraries it is desirable that their components are useable for creating both *Swing* and *SWT* applications. Although functionality exists in *SWT* to embed *Swing* components into an *SWT* GUI (the *SWT-AWT*-bridge, [278]), interaction between these components is expensive to implement. The

main reason is that both toolkits use their own GUI thread and special thread management code is necessary for each interaction between components from the different toolkits.

For *LibrAlign*, toolkit-independence is a requirement since the start of the project, because among the applications in need of the functionality to be developed were both *Swing*- and *SWT*-based ones. In order to fulfill this requirement, functionality is needed that simplifies creating GUI components to be combined with both toolkits. Using the *SWT-AWT*-bridge available in *SWT* was not a suitable option due to the reasons described above. Instead of developing the required abstractions over the toolkits directly in *LibrAlign*, all respective implementations were made in a separate library, so that this functionality is available to the public and can easily be reused elsewhere. The new library is called *TIC*, which stands for *Toolkit Independent Components*.

12.2 Concept

The aim of TIC is to provide abstractions between the GUI toolkits *Swing* and *SWT*, allowing developers to create new components usable in both toolkits without having to make multiple toolkit-specific implementations. Although imaginable, *TIC* does not focus on providing abstractions between concrete available components, such as buttons or text fields, to allow to create toolkit-independent components that are made up of existing subcomponents. Its aim is instead to allow the creation of components that directly paint their contents using component-specific code. (Displaying multiple sequence alignments as done by *LibrAlign* is an example of a use case where content is painted directly, and no standard subcomponents are needed.) The library should include a component base class with an abstract painting method, which needs to be implemented only once and can then be used in both a *Swing* and an *SWT* version. Component developers only need to inherit a *TIC* component and *TIC* will take care of creating respective *Swing* and *SWT* versions.

In addition, the library should provide abstractions for user interaction and scrolling functionality, so that developers can implement listeners to user input only once in *TIC* without having to worry about the differences between both toolkits. In the same way, listening to scrolling events and programmatically setting the scroll position of a component should be abstracted.

12.3 Implementation

`TICComponent` is the key class of *TIC*. (See also Figure 12.1.) All toolkit independent components developed using *TIC* are inherited from this class and it contains the abstract toolkit-independent painting method, outlined in the concept above. This method needs to be implemented by inherited classes and uses an instance of `TICPaintEvent` as its parameter, which provides an instance of the *Swing* class `Graphics2D` as the graphics context to be used by method implementations. If the *TIC* component is used in *Swing*, the graphics context will directly be obtained from there. If used in an *SWT* GUI, *TIC* will first create a new `Graphics2D` object that draws into a buffered image and after the execution of the painting method is complete, the contents of that buffered image will be copied into the SWT graphics context (the `GC` instance) to paint the *SWT* component. To keep the memory consumption for buffering low, only the rectangular part of a component that is repainted by the current operation will be stored in the buffered image.

Besides allowing toolkit-independent painting of GUI components, their reactions to user inputs need to be abstracted over both toolkits as well. In order to further simplify creating toolkit-independent components, *TIC* provides a set of event listener interfaces and event classes in its package `info.bioinfweb.tic.input`. An implementation of the interface `TICKeyListener` can be registered at every class inherited from `TICComponent` to receive key events from the toolkit-specific version of a *TIC* component. The events passed to the listener methods of `TICKeyListener` are also *TIC*-specific objects and make use of *Swing*/*AWT* constants to define keys, key masks or keyboard positions. When a *TIC* component is used with *SWT*, the same events are used, and all information is converted from

the *SWT* representation (that uses different constants and strategies) to the *Swing* representation used as the toolkit-independent representation in *TIC*. Following the same principle, different types of mouse listeners and respective events are also provided and information is converted. Translating between the different ways of modelling mouse movement and dragging in both toolkits is also handled by *TIC*.

In addition to painting and reacting to user inputs, a third component characteristic abstracted by *TIC* between both toolkits is scrolling. Scrolling of components that have contents larger than their current size is handled in different ways in both toolkits. *TIC* provides the abstract base class `ScrollingTICComponent` that inherits from `TICComponent` (Figure 12.1) as a toolkit-independent scroll container. It provides methods for programmatically obtaining and changing the current scroll position of the nested component, including convenience methods like optionally scrolling to make a rectangle fully visible and an architecture for toolkit-independent scroll listeners, similar to the key and mouse listeners described above. As Figure 12.1 shows in blue, a set of interfaces for toolkit-specific scroll container to be used with classes inherited from `ScrollingTICComponent` is provided. These interfaces contain default method implementations that act as adapters from the *TIC* scrolling methods to the methods of the scroll containers available in the toolkit GUI libraries. For *Swing* `JScrollPaneToolkitComponent` can be used to easily delegate scrolling functionality to an instance of `JScrollPane`, while `JScrollBarsToolkitComponent` allows to delegate to two instances of `JScrollBar`. `ScrolledCompositeToolkitComponent` allows to do the same for `ScrolledComposite` in *SWT*. The *TIC* documentation contains a set of example applications that demonstrate the different ways toolkit-independent scrolling can be achieved under http://r.bioinfweb.info/TICScrollDemo.

A special functionality provided by the *swt* module of *TIC*, is direct scrolling of nested *TIC* components. *SWT* has greater limitations than *Swing* to the maximal component size within a scroll container on some operating systems. (This is due to the use of code working with 16 bit integers in the GUI libraries of these operating systems.) To overcome this limitation, developers of large scrolled *SWT* components (e.g. to display biological data like alignments or phylogenetic trees) have to create their components in a way that they can display their contents with a scroll offset and perform scrolling themselves directly. This is usually more expensive to do than just placing the content component into a scroll container. For *TIC* components, such direct scrolling functionality for *SWT* is directly included when using `DirectPaintingSWTScrollContainer` together with `ScrollingTICComponent`. This way scrolling large components becomes significantly easier when working with *TIC*, compared to implementing such components directly in *SWT*, because any directly painting *TIC* component can be scrolled this way without having to be adjusted. (Note that by definition, direct scrolling is only possible for *TIC* components painting their contents directly, which are the main focus of *TIC* and not for custom toolkit-specific components consisting of a set of other nested components from the toolkit component libraries.)

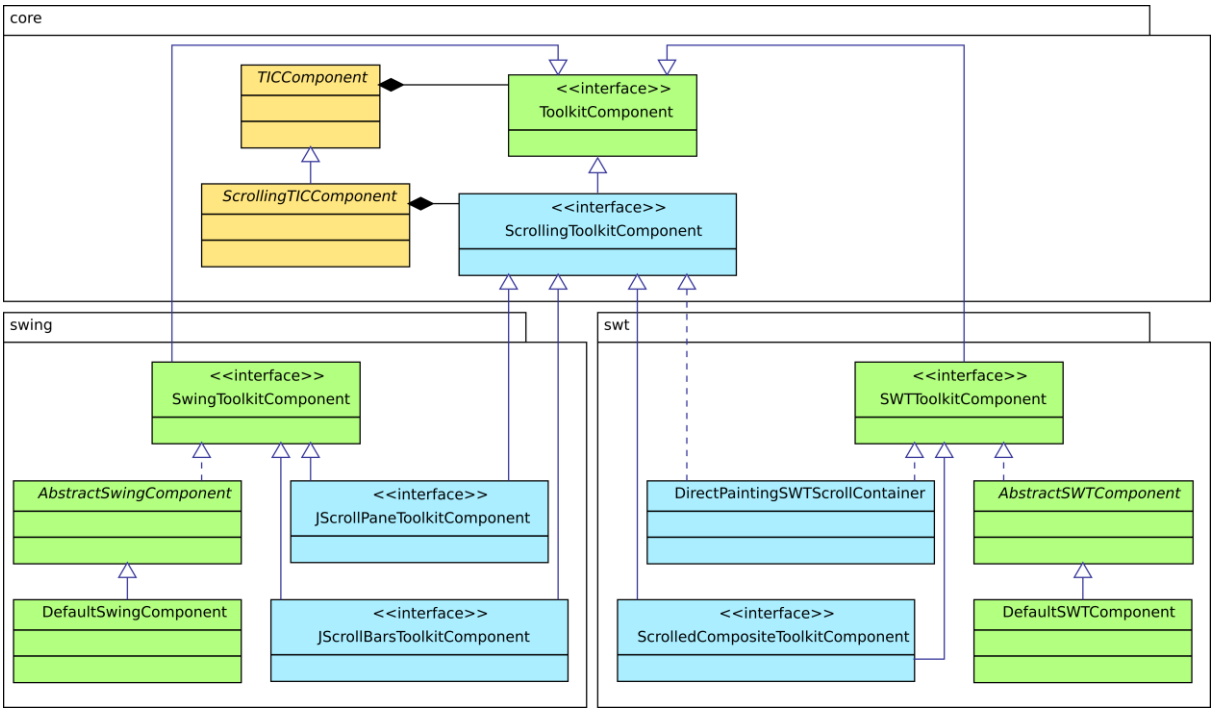


Figure 12.1 Overview on the toolkit-independent and toolkit-specific component classes in TIC

This UML class diagram shows the toolkit-independent classes of TIC *that model GUI components in orange.* `TICComponent` *is the parent class if all component classes implemented by users of* TIC *and will be associated with a toolkit-specific class implementing* `ToolkitComponent` *as soon as processed with a respective factory. (See chapter 12.4 for details.) All general toolkit-specific classes are shown in green here.*

If toolkit-independent scroll container components should be developed, these can be inherited from `ScrollingTICComponent`*, which in turn links a toolkit-specific class inherited from* `ScrollingToolkitComponent`*. All classes and interfaces that act as toolkit-specific scroll containers are shown in blue. The different interfaces contain default method implementations that delegate scrolling functionality to scroll containers available in* Swing *and* SWT*, while* `DirectPaintingScrollContainer` *is a special* SWT *implementation in* TIC *that allows to scroll larger components than* `ScrolledComposite` *of* SWT *does. (See chapter 12.3 for further details.)*

12.4 Results and discussion

TIC is available for download from http://bioinfweb.info/TIC/, including a *JavaDoc*, an example application and further documentation. Source codes and binaries are distributed under GNU Lesser General Public License Version 3.

The library is separated into three modules. The *core* module contains all *TIC* interfaces and classes that define the abstraction of GUI components – like painting, user interaction and scrolling – over both toolkits, while the *swing* and *swt* modules provide the toolkit-specific functionality. Depending on whether *TIC* is used in a *Swing* or an *SWT* application only one respective module is necessary as a dependency. Depending GUI libraries can be developed using the same module architecture to avoid dependencies between both toolkits. (*LibrAlign* would be an example for this.)

To embed a *TIC* component into a *Swing* or *SWT* GUI, a respective toolkit-specific component instance needs to be created from it. This is done by factory classes from the *swing* or *swt* modules of *TIC*. `SwingComponentFactory` creates a *Swing* component from a *TIC* component, while `SWTComponentFactory` allows to create an *SWT* component from it. Both resulting components will look and function identical as implemented in the toolkit-independent *TIC*-component. By default, instances of `DefaultSwingComponent` and `DefaultSWTComponent` (see Figure 12.1) will be created by the factories. These classes delegate to the implementations made in the toolkit-independent *TIC* component for, e.g., painting or user interaction as described in chapter 12.3. Figure 12.2 further illustrates this

principle and Figure 3.2 (page 50) contains code examples showing how to use the *TIC* factory classes to embed a *TIC* component in GUIs of both toolkits.

Alternatively, developers have the option to provide custom toolkit-specific components with different or additional functionality, which must implement the interfaces `SwingToolkitComponent` or `SWTToolkitComponent` respectively and may inherit basic functionality from `AbstractSwingComponent` or `AbstractSWTComponent`. Providing custom toolkit-specific classes is only necessary in special cases, while the main use case of *TIC* – which is providing direct painting components – does not require to provide any custom toolkit-specific components.

By using the conversion between the *Swing* and *SWT* graphics contexts described in chapter 12.3, *TIC* allows access to advanced *Swing* drawing functionality like platform-independent anti-aliasing or the *Swing* shape API also for *SWT*. This way, the contents of any *TIC* component implementation look exactly the same on both toolkits.

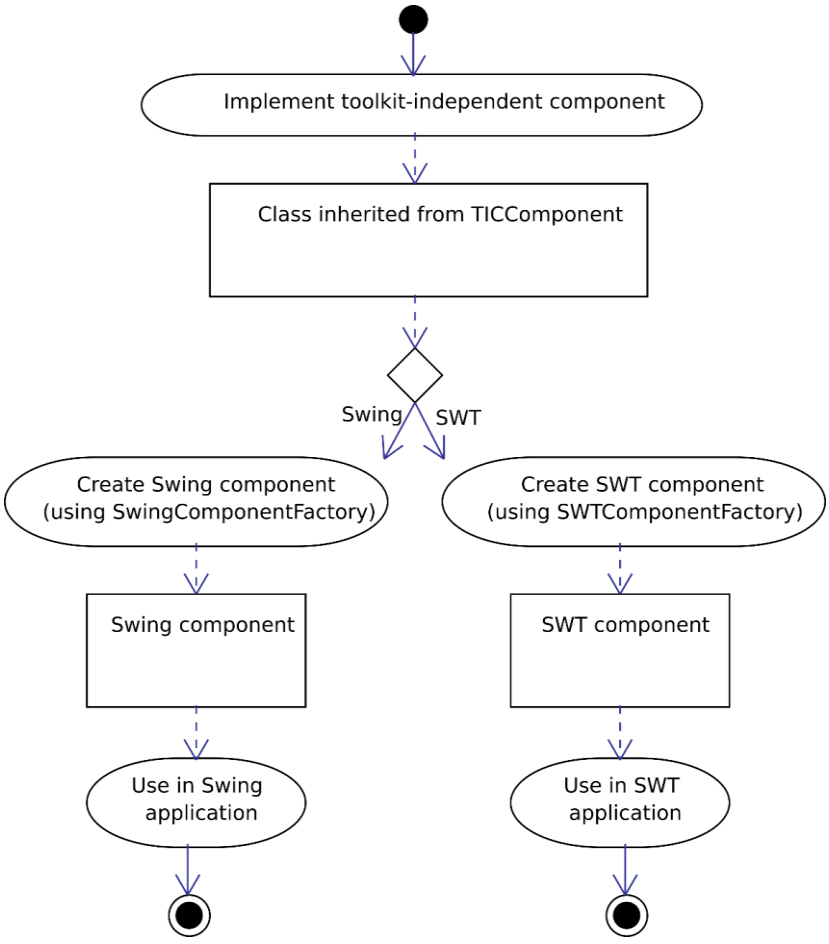


Figure 12.2 UML activity diagram showing the steps to create a toolkit-independent GUI component with TIC

Developers only need to implement a toolkit-independent TIC *component once and then are able to include it in both* Swing *and* SWT *GUIs by using one of the respective factory classes of* TIC *to create a toolkit-specific instance from it. (See also Figure 3.2 on page 50 for a code example on using the factories.)*

12.5 Conclusion

Directly painting GUI components developed with *TIC* can be used as fully functional native components both in *Swing* and *SWT* GUIs without the need for any toolkit-specific adjustments. In addition

to component painting, reacting to user inputs and scrolling larger components is abstracted by the library.

TIC was already successfully used in the development of *LibrAlign* that provides toolkit-independent GUI components for displaying and editing multiple sequence alignments and attached raw and metadata (chapter 3, page 46). The *TIC* components of *LibrAlign* have been successfully tested and used in multiple publicly available biological software, some based on *Swing* and some on *SWT*. (See chapter 3.3.5 on page 54 for further details.)

12.6 Availability and requirements

Project name: *Toolkit independent Components* (*TIC*)

Project web page: http://commons.bioinfweb.info/Java

GitHub Repository: http://bioinfweb.info/TIC/

Operating system: Platform independent

Programming language: *Java*

Other requirements: *Java* Runtime Environment 8 (or higher)

License: *GNU Lesser General Public License* Version 3 (*LGPL*)

Any restrictions on use by non-academics: The restrictions specified in the *LGPL* apply. (See http://bioinfweb.info/TIC/License.)

12.7 Declarations

12.7.1 Author Contributions

Ben Stöver developed the concept, implemented the library and wrote the manuscript.

12.7.2 Acknowledgements

Ben Stöver thanks Sarah Wiechers for her contributions to a *TIC* example application during a research module, which was supervised by Ben Stöver. The work of the contributors of *Apache Commons*, which is used as a dependency, is highly appreciated.

13 General discussion and outlook

The different software components introduced in this thesis contribute to achieving the goals outlined in the general introduction (chapter 1, page 20). They cover all major phylogenetic data types and facilitate large parts of phylogenetic and related workflows. Reading and writing as well as processing and visualizing different kinds of metadata closely together with the data in flexible and user-friendly ways has been a focus in the development of all projects. Large parts of the new functionality are available in separate libraries, so that all developers of bioinformatical software can benefit from it. Some developed applications are already in wide use in the scientific community and together, the applications and libraries open different new perspectives for future research and method development, as it will be discussed in this chapter.

13.1 Increasing data reuse and reproducibility

Increasing the reusability of phylogenetic data and the reproducibility of studies by making meaningful annotation and documentation more straightforward was an important goal of this thesis (goal 1 on page 20). All developed software contributes to this aim by providing different necessary functionality for both researchers and bioinformaticians.

13.1.1 Developed functionality

The software libraries *JPhyloIO* (chapter 2) and *LibrAlign* (chapter 3) provide the basis for all software developed in this thesis and are also available and free to use by other developers of current and future scientific software. By generalizing over most phylogenetic file formats, *JPhyloIO* provides access to a large part of the available phylogenetic data, including attached metadata, and allows extensive interoperability to other software in the domain. Providing this functionality in one single interface, allows all applications to support classical and widely used formats together with more recent and advanced formats in the same step.

The flexible model classes and GUI components to display and edit sequences and MSAs implemented in *LibrAlign* provided the basis for the development of the molecular components of the *Taxonomic Editor*, *PhyDE 2* and *AlignmentComparator*. The concept of data areas directly attached to sequences and whole MSAs is an important contribution to goal 1 (page 20), i.e., to foster meaningful annotations in order to improve reusability and increase reproducibility. Any software dealing with raw- and metadata associated with sequences or MSAs is now able to provide a user-friendly *Java* GUI by implementing an application-specific data area or may even be able to use an existing one either provided with *LibrAlign* or potentially from third party developers in the future. Beyond that, *LibrAlign* opens the perspective to handle externally defined types of metadata with externally implemented data areas, without having to explicitly model them in an application. (See section 13.1.3 for further details.)

The different applications cover all major types of phylogenetic data and allow to model complete workflows. The *Taxonomic Editor* increases data reusability and the reproducibility of taxonomic work by linking all types of modeled data back to a specimen, which is the only model that is invariant regarding future taxonomic revisions that change the assignment of specimens to taxa and makes alpha-taxonomic work significantly easier. Within this thesis, the functionality of the *EDITor* was extended to support molecular sequence data. Reproducibility and reusability of that data is therefore improved by associating it with its source specimen and linking all single read sequences to their Sanger sequencing pherograms and displaying that raw data directly within the contig editor. (See Figure 5.2 on page 81.)

An initial version of *PhyDE 2* that is now based on *LibrAlign* and *JPhyloIO* has been developed as a proof-of-concept and to open a perspective for the future development of the alignment editor. *PhyDE 2* has though not been a focus of this thesis and therefore it does not yet have extensive functionality

to model and display phylogenetic metadata, but it already uses *NeXML* as its main format and can easily be extended in the future, since necessary I/O functionality is already provided by *JPhyloIO* and extending the GUI to model different kinds of metadata is straightforward by inserting respective *LibrAlign* data areas. The new codebase even provides a perspective for *PhyDE* to be integrated into a larger phylogenetic workbench with a strong focus on flexible metadata modeling (see section 13.1.2) and to support any type of metadata attached using externally defined ontologies together with externally implemented data areas (see section 13.1.3).

AlignmentComparator mainly contributes to goal 2 (page 20), i.e. allowing researchers to take alternative methods into account and to document their workflows respectively (see section 13.2) and makes use of the functionality of *JPhyloIO* by using *NeXML* as its main format and storing all comparison metadata using its *RDF*-based model. Its development was simplified by *LibrAlign*'s extensibility using specific data area implementations to display comparison metadata and user annotations directly within the comparison of MSAs. Like *PhyDE 2*, *AlignmentComparator* can also easily be extended in the future to display externally defined metadata, it does not explicitly model and display it using externally implemented data areas. (See also section 13.1.3 and the its future application in MSA evaluation described in section 13.2.2.)

The phylogenetic tree editor *TreeGraph 2* focuses on displaying and combining different types of metadata attached to branches and nodes of a tree. It allows to attach an unlimited number of different types of labels to each branch and provides a table view to edit metadata annotations for each tree. Features to automatically set node colors, font sizes, branch widths any many other tree elements by respective numeric annotations, allows to visualize attached data in many ways and a flexible system for calculating textual and numeric attachments from each other, including topological features (chapter 9.3.4, page 148), further contributes to its capability to handle phylogenetic metadata. Import features for different tree formats, data tables (chapter 9.3.3, page 147) and ancestral character state reconstructions from *BayesTraits* (chapter 9.3.2, page 143) ensure easy access to the relevant data. The initial versions of *TreeGraph 2* (described in chapter 8, page 128) were using string keys to link metadata to nodes and branches as they are also used in hot comment annotations extending the *Nexus* standard. While still much of the widely-used phylogenetic analysis software uses the *Nexus* format with such annotations (e.g. [73,74]), which can then be further processed and visualized using *TreeGraph 2*, linking that data using *RDF* predicates would be more advantageous to unambiguously describe the relation between the tree elements and their metadata. The currently distributed versions of *TreeGraph 2* already support to import metadata from *NeXML* and additionally the metadata model is refactored to combine the use of string keys with the use of *RDF* predicates, allowing backwards compatibility and support for powerful forward-looking technologies at the same time. As described in chapter 9.3.6 (page 151), the conception and implementation of the new model has been performed in a development branch and adjustments of GUI components to the new model remain to be done. As soon as that is completed, *TreeGraph 2* would be the first phylogenetic tree editor to make full use of the flexible *RDF*-based metadata model of *NeXML* to increase the reusability of phylogenetic trees and to improve reproducibility.

The functionality developed in the libraries *JPhyloIO* and *LibrAlign* provides a strong basis for the development of new and the extension of exiting phylogenetic software to address the needs imposed by the age of big data and the sematic web allowing researchers to make optimal use the available technologies, like *RDF* or *NeXML* to increase the reusability of their data and the relevance and reproducibility of their studies. In addition to that, the applications developed in this thesis cover the most important datatypes of phylogenetics and therefore most phylogenetic workflows, support *NeXML*, and model a variety of different metadata. To make optimal use of the functionality provided by the libraries and to provide researchers with convenient tools that go one step further in the metadata

modeling for increasing reproducibility and reuse than most other available software, some further implementations that were beyond the scope of this thesis are still necessary, but the developed applications already provide a lot of useful features for that purpose and their further extensions is made easy by their maintainable and extendable architecture, also due to being based on the flexible libraries *JPhyloIO* and *LibrAlign*.

The following sections 13.1.2 and 13.1.3 discuss the future perspectives for the developed components in more detail.

13.1.2 Conserving relevant metadata throughout the whole taxonomic and phylogenetic workflow

Although flexible metadata modeling for unambiguous annotation of phylogenetic data is of key importance for making data reusable and studies reproducible (cf. chapter 1.1, page 20) and *JPhyloIO* and *LibrAlign* simplify developing software that addresses these needs, it would be unrealistic to expect that all software used in phylogenetic workflows is going to support the corresponding standards within the next years. Since the applications developed in this thesis model major datatypes used in phylogenetic workflows, they could be used to conserve any type of metadata during all of its steps, even if applications that do not support this are involved. Figure 13.1 illustrates this for the example of metadata annotations of sequences that link to the initial specimen the sequences are derived from, as implemented in the *Taxonomic Editor*. The same principle could be applied to all other types of metadata.

The information from the *Taxonomic Editor* can be exported using *NeXML* contig files that contain information on the single reads, a consensus sequence was constructed from, including links to the sequencing raw data of each read. To perform a multiple sequence alignment from a set of such consensus sequences, an input file for an MSA application containing the whole set needs to be created. Since none of the frequently used MSA software currently supports *NeXML* or conserving externally defined metadata in any way, the links to the specimens and sequencing raw data would already be lost at this step of a workflow and not be available in any subsequent analyses based on the MSA. (The metadata provided by the *Taxonomic Editor* is only used as an example here. Other metadata like database IDs or information on genomic regions or protein domains could be processed the same way.)

This problem of losing annotations could be avoided if the single reads were combined using a metadata-aware alignment editor. As mentioned above, *PhyDE 2* currently is only available in a basic version, but since it is based on *LibrAlign* and *JPhyloIO* and uses *NeXML* as its main format, it can be easily extended to model externally defined metadata, which is planned for future releases. As shown in Figure 13.1, *PhyDE 2* could then combine the imported set of contig files to one *NeXML* file that contains all sequences, including their specimen links and references to the initial contig files as resource metadata, preserving references to the raw data for each read. This *NeXML* file would then be the main instance to store the data of the current analyses. To align the sequences with an external tool, e.g., *MAFFT* or *PAGAN*, or any other MSA algorithm implementation, a file in the required format (e.g., *FASTA*) could be exported by *PhyDE 2* from the main sequence file in *NeXML*. Since it is not possible to include metadata in this step, all taxon names (or other names associated with the sequences) would be replaced by unique IDs by *PhyDE 2*, which can be unambiguously reassigned later. (This would also overcome format-specific limitations in the length or character set of sequence names, e.g., imposed by *Phylip*.) The exported file can then be processed by the aligner and the result (containing the aligned sequences labeled with their new IDs) can be opened with *PhyDE 2* again afterwards. Since the alignment editor is aware of the IDs and still has access to the main *NeXML* file, it can restore all metadata and produce a new *NeXML* file containing the aligned sequences with their initial names and

all associated metadata. This way, all metadata has been conserved over the workflow until now, although an application that does not support it was involved. The requirements for this, are that the user performs importing and exporting of the data with *PhyDE 2*, which has access to the main *NeXML* file at all time, e.g., in a user workspace folder, and that *PhyDE 2* supports the format required by the external application. Due to the variety of formats supported by *JPhyloIO*, that are automatically supported by *PhyDE 2* as well, nearly all available MSA applications can be integrated into such a workflow.

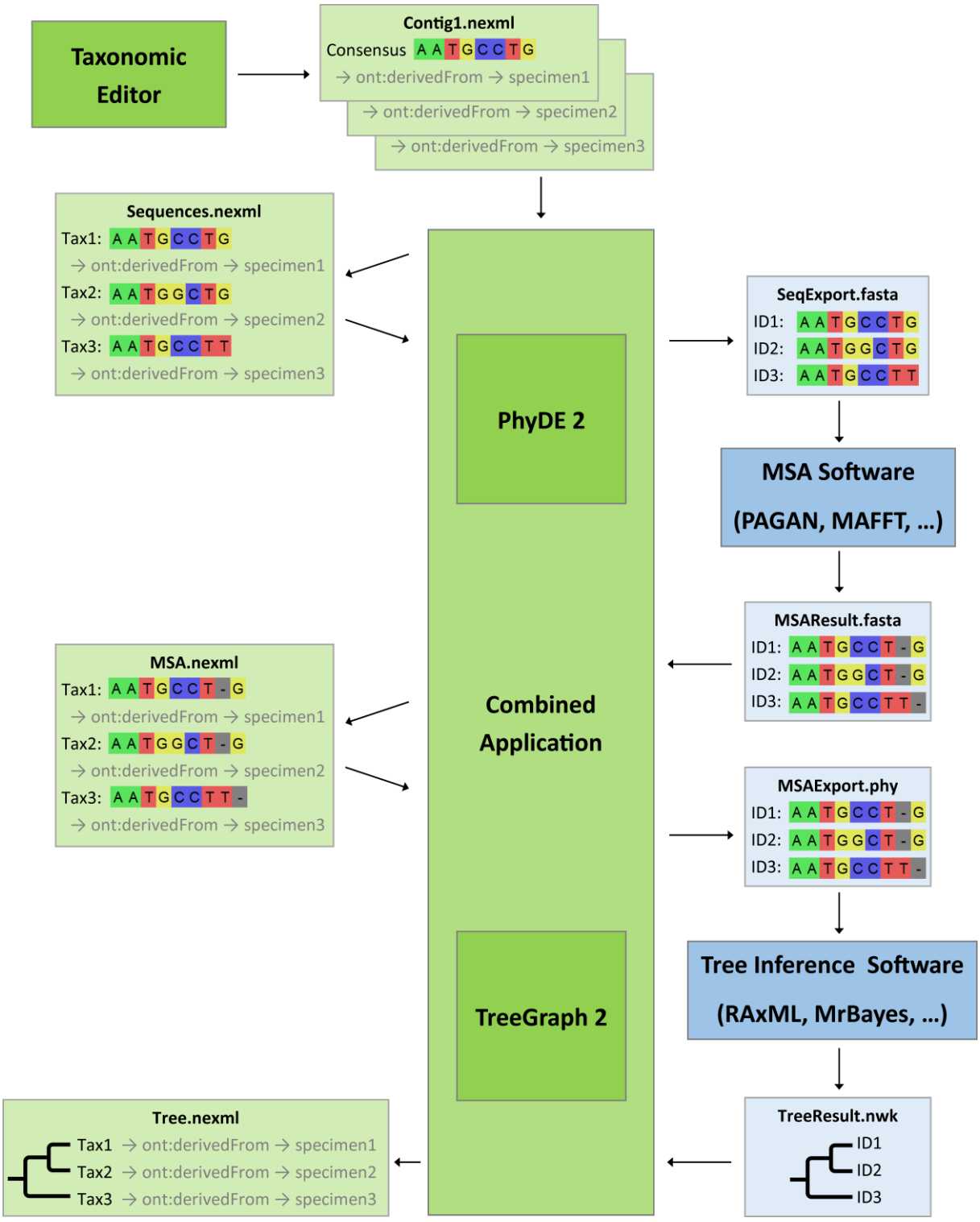


Figure 13.1 Possible future usage of* PhyDE 2 *and* TreeGraph 2 *to model and preserve relevant metadata across phylogenetic workflows

(Caption on next page.)

***Figure 13.1** (Continued.)*

The schematic workflow shows how metadata could be preserved across a phylogenetic workflow, even if software is involved that does not support that metadata. Documents and applications that model metadata are shown in green, while blue indicates external applications that are not aware of metadata in their input and output documents and would lose such information. The Taxonomic Editor *(chapters 4, page 59 and 5, page 79) is shown as an example for a producer of annotated data and exports a set of contig alignment documents with links to their specimens as metadata. (Note that the ontology* `ont` *to attach such data is hypothetical and not actually used by the applications. Not all functionality shown here is already implemented in the different applications. See text for details.) The contig files are imported by* PhyDE 2 *(chapter 6, page 84) in the next step and stored in a combined* NeXML *file preserving all metadata. A* FASTA *file is then exported, where all taxon names are replaced by unique IDs that can later be used to map the import back to the metadata. Since the external MSA software does not model the attached metadata, its resulting MSA is converted back to* NeXML *and all annotations are restored by* PhyDE 2*. In a subsequent step the same principle is applied for calling a phylogenetic inference software and* TreeGraph 2 *(chapters 8, page 128 and 9, page 137) restores all metadata after the import. The whole process could either be implemented with* PhyDE 2 *and* TreeGraph 2 *as separate applications that exchange data or by using a combined workbench application containing the functionality of both (as indicated by the light green box surrounding both applications).*

After an MSA has been created, the next step of phylogenetic workflows often is phylogenetic inference. The available applications for this, frequently expect their input sequences in *Phylip* or *Nexus* format, which are both supported by *JPhyloIO*, and therefore an MSA input file could be exported from *PhyDE 2* in the same way as for the previous step, again using the same IDs for all sequences. The phylogenetic inference software will then output one or more phylogenetic trees using, e.g., *Newick* or *Nexus* formats, which are also supported by *JPhyloIO*. The resulting tree could then be opened using *TreeGraph 2*, which supports all tree formats of *JPhyloIO*. Within the currently ongoing extension of *TreeGraph 2*'s metadata model (see 9.3.6, page 151), a feature could be added that allows combing the data from the *NeXML* MSA source file and the tree file to create a *NeXML* tree file (combined with the MSA or not, depending on the use case) that still contains all initially present metadata linked to the nodes of the tree. (In some cases, it may be necessary to select the metadata that is copied onto the tree, since not all types are still suitable for a tree node, e.g., detailed position information of mutational patterns. The metadata could either be filtered automatically, e.g., using knowledge about the *RDF* predicates or by prompting the user. What metadata to copy, also depends on whether the tree is stored in the same file, together with the MSA and its metadata and if a shared taxon list with annotations is used. Linking tree nodes and sequences would be an alternative in such cases for some types of metadata.) The implemented features of *TreeGraph 2* to combine support values from alternative analyses can be easily integrated into a metadata-preserving workflow modeling, since the same sequence/node IDs as for the first phylogenetic inference can be used for all analyses.

Beyond preserving initially present metadata, the proposed type of data management using *PhyDE 2* and *TreeGraph 2* holds the potential to automatically or semiautomatically collect information on the workflow itself and store, e.g., the used analysis software and its version and parameters as additional metadata. The output of some external applications contains information about their names and version numbers, e.g., in comments. *PhyDE 2* and *TreeGraph 2* could scan for such information when importing output files and automatically create respective metadata in the target *NeXML* document. In other cases, like *FASTA* output files from MSA software, no such information may be directly available, but the user could be prompted for it directly on the import. A convenient input dialog would simplify the workflow documentation and possibly motivate and remind more users to perform such a workflow documentation directly.

Calling external software directly from within *PhyDE 2* and *TreeGraph 2* (or from a combined application, see below) or a closer integration of both applications with available workflow managers are options to be considered in the future development, but providing functionality to document workflows

with manually executed external applications (also if documented by, e.g., batch files) will still be an important functionality to reach a maximal number of users, also those that are not planning to use workflow managers for whatever reasons.

Preserving metadata this way would be possible with few extensions to the current functionality of *PhyDE* 2 and *TreeGraph 2*. While *TreeGraph 2* already provides a large part of the necessary functionality (as described in chapter 9), more implementations will be necessary to extend *PhyDE 2*, but the easily extendable architecture of both programs and the use of *JPhyloIO*, would make the necessary steps relatively straightforward. Combining the data from the *NeXML* MSA file and the output of the external phylogenetic inference software would presumably more convenient and less error-prone for the user, if a workbench application combining the functionality of *PhyDE 2* and *TreeGraph 2* is created (cf. chapter 9.3.8.3, page 160). Such an application could be built around a workspace, where the user can store the contigs, the MSA, the tree and potentially additional files and would allow to access all features implemented in *PhyDE 2* and *TreeGraph 2*. Importing and exporting data from and to other applications, as described above would be possible for the user in a convenient way.

The applications developed in this thesis already form a substantial basis for the full implementation of phylogenetic workflow modelling and blaze the trail for complete metadata conservation and linking relevant raw data and specimens in all analysis steps. The functionality of *JPhyloIO* is an important contribution to achieve this goal, not only because it allows access to the full metadata model of advanced formats like *NeXML* and *phyloXML*, but also due to its simultaneous support for a variety of classical formats that provide the necessary interoperability with the great majority of external tools to be integrated into the modeled workflow.

13.1.3 Externally implemented GUI components for metadata attached by externally defined ontologies

As mentioned previously (e.g., in chapter 1.1.3 and 1.1.4, page 21), one of the advantages of the *NeXML* format is the possibility to link metadata to phylogenetic data elements using *RDF* predicates from ontologies that are externally defined. This provides great flexibility, since ontologies for specific purposes can be created without the need to change the format itself, which would affect the whole phylogenetic community. The same principle can also be applied to software that models, displays and allows to edit phylogenetic data together with different metadata. General-purpose software, like *PhyDE 2* or *TreeGraph 2*, could never explicitly model all types of metadata that might be attached to MSAs of phylogenetic trees for a variety of purposes. *TreeGraph 2* already offers functionality to display the textual representations of metadata as they are encountered in a *NeXML* document and *PhyDE 2* is planned to be extended to do so (cf. section 13.1.2 and chapter 6.3.4, page 88), but a more advanced graphical representation and editing functionality of discipline-specific metadata that goes beyond that is desirable for many use cases. Implementing specialized applications that model metadata of sequences or MSAs is already significantly simplified by *LibrAlign* that easily allows to extend its GUI components with any application-specific or third-party data area (cf. chapter 3.3.1, page 49). These data areas can then provide the specialized visualizing and editing functionality.

Beyond that, the architecture of *LibrAlign* holds the potential to provide such visualizing and editing capabilities even more easily. Applications like *PhyDE 2* or also more specialized ones like *AlignmentComparator* or the *Taxonomic Editor* could support the integration of custom third-party data areas at runtime. A *NeXML* document containing domain-specific metadata annotations of sequences (e.g., the position information of mutational patterns, as described in section 13.2.2, page 186) could additionally contain special annotations to unambiguously define the type(s) of data areas to display and edit each type of metadata. Data areas could either be referenced by unique identifiers, e.g., defined in a central data area repository, or directly by a download URL of a data area implementation.

If a central repository or database for data areas would be developed, it would also be thinkable, that applications like *PhyDE 2* query it with the *RDF* predicates encountered in an opened document and it returns a list of matching data area implementations. The user could then be prompted, whether to download and integrate one of these. (Possible security issues might, though, have to be addressed to prevent abuse of such a system. Certified download sources and restricting the access of data areas implementations to resources of the local machine would be options to be considered. *Java* already provides respective functionality and comparable problems have already been addressed in other modular frameworks, like *Eclipse*.)

Applying the proposed concept, developers could simply implement a data area for specific metadata they need to model and either upload or register their implementation in a respective database or provide a download URL with *NeXML* files containing respective data (depending on the implemented architecture). After that, no further steps would be necessary and their data area can directly be used in combination with *PhyDE 2* or any other *LibrAlign*-based application without the need to develop a whole new application. Accessing the relevant metadata required by such a data area implementation can easily be achieved using *JPhyloIO*, which is closely integrated with *LibrAlign*. It allows an unlimited number of data listeners (cf. chapters 2.2.1 on page 35 and 3.2.4 on page 48) so that all data areas can acquire necessary information directly when a document is loaded by an application. Similar techniques exist for writing data and metadata. Furthermore, *JPhyloIO* provides automatic format-independence and therefore data areas are enabled to also display and edit metadata provided, e.g., in *NeXML*, *phyloXML* or in *Nexus* hot comments.

As described for an example in section 13.2.2 (page 186 below), integrating externally implemented data areas not only makes developing separate *LibrAlign*-based applications unnecessary in more simple cases, it also enables software like *AlignmentComparator* to display metadata in new contexts. When alternative MSAs are visually compared, *AlignmentComparator* could use a technique as herein described to display relevant metadata within the visual comparison and a user can directly inspect the influence and relation of different types of (externally defined) metadata on the differences between MSAs. Using externally defined ontologies together with externally implemented data areas allows to include any type of metadata in the visual comparison, without the needs of explicitly modelling it in *AlignmentComparator*. The same principle can be applied for many other thinkable applications.

Data areas of *LibrAlign* cover the visual representation of all metadata attached to sequences, character matrices and MSAs, but to support all data modeled by *NeXML*, a visual representation of metadata attached to phylogenetic trees and their nodes and branches would additionally be needed. The different types of branch labels provided by *TreeGraph 2* to display various types of metadata directly within a phylogenetic tree, can be considered as the equivalents to *LibrAlign*'s data areas for phylogenetic trees. The components of *TreeGraph 2* are currently not explicitly exposed in a library comparable to *LibrAlign*, but the application is open source and products like the alignment evaluation software (introduced in section 13.2.2 below) already reuse its functionality. Creating a *TreeGraph API* as a counterpart to *LibrAlign* for phylogenetic trees is a planned future development. The different kinds of branch labels could be externally implemented in the same way as the data areas of *LibrAlign* already are and then be integrated into *TreeGraph 2* at runtime as well as into possible additional applications based on a *TreeGraph API*. Externally implemented labels could be referenced by metadata attachments, e.g., directly in *NeXML*, as described above or be provided using the same database as for data areas.

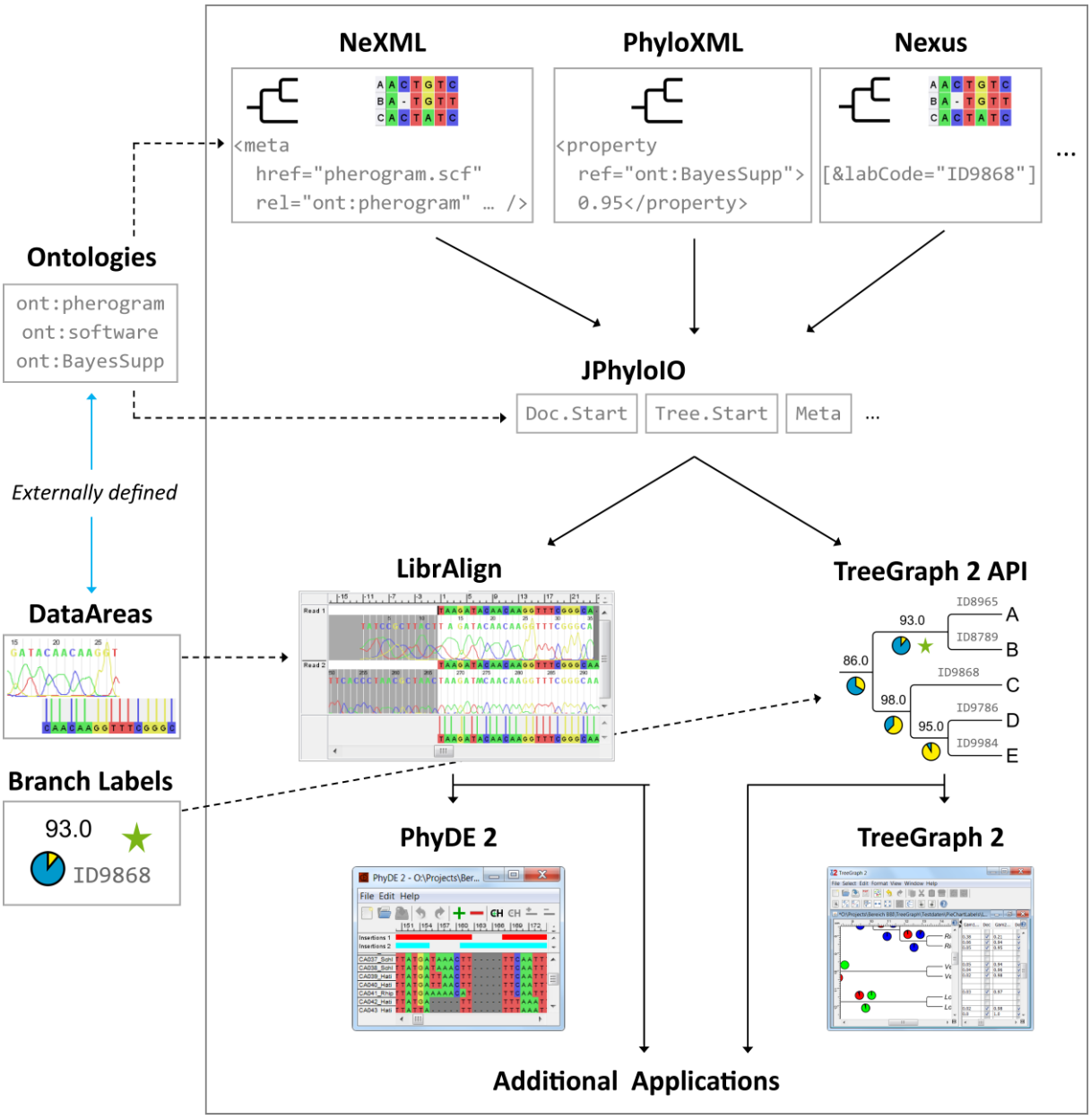


Figure 13.2 Externally implemented GUI components to handle metadata attached using externally defined ontologies for phylogenetic data

Externally implemented data areas of LibrAlign *and labels of* TreeGraph 2 *and its API represent the GUI counterparts of externally defined ontologies. Metadata available in different formats can be accessed using* JPhyloIO *though one common interface that models the metadata attachment in an* RDF*-like way. Such annotations can then be visualized and edited using matching data areas and labels that can be integrated into* PhyDE 2*,* TreeGraph 2*, or any other application that is based on* LibrAlign *or the* TreeGraph 2 API*.*

Together externally implemented data areas and *TreeGraph 2*'s labels can represent user interface counterparts of externally defined ontologies and would allow GUI-based applications to provide the same flexibility regarding the processing of metadata as *RDF* and *NeXML* allow for its storage and interchange. Figure 13.2 provides a schematic overview on this concept and the usage of the now available software components to implement it. The libraries and applications developed in this thesis provide a substantial basis for such a system. Having such a framework that is applied by bioinformatical software developers, could further foster the use of metadata annotations to improve and simplify

many workflows and data reuse not only in phylogenetics and its related disciplines and could open up new perspectives and opportunities for research and method development.

13.2 Comparing phylogenetic data

As mentioned in chapter 1.2 (and chapters 7 and 9), software to compare multiple sequence alignments and phylogenetic trees is essential in all studies involving these datatypes, since a variety of alternative methods are available both for constructing MSAs and for inferring trees. Downstream analyses are influenced by the selection of these methods and it is not always trivial to decide which algorithm to use and the results of multiple alternative ones must be compared. Furthermore, changes made to MSAs or trees during a workflow can be visualized and documented using comparison software.

13.2.1 Developed functionality

Applications for detailed comparisons of the results of constructing MSAs and inferring trees, which are the two main steps in many phylogenetic studies and deal with the main datatypes of the *Nexus* data model, have been developed in this thesis. Beyond that, MSAs and trees play a major role in numerous other fields of the life sciences and the software can be equally useful there.

AlignmentComparator (chapter 7, page 90) allows to visually compare alternative MSAs of the same or slightly different datasets by performing a superalignment between the alternatives and providing a visual comparison that takes differences between all sequences into account. With its alternative superalignment algorithms and graphical user interface, it allows researchers to identify differences and agreements of MSAs in a convenient way that is to date not offered by alternative software. (See chapter 7.4.4 on page 121 for details.)

Among many other features, *TreeGraph 2* provides functionality to map conflicting support onto one phylogenetic tree topology (chapter 8.3.1) and to interactively compare trees (chapter 9.3.1 and Figure 9.2, page 140). The comparison algorithm that takes support values into account (Figure 9.1, page 139) allows a detailed visual inspection of greater and smaller topological and support differences, which is a novelty compared to other software. (See chapter 9.3.7.1 on page 155 for further details.)

The convenient comparison functionality of both applications allows considering alternative methods and selecting the optimal tools in many use cases, instead of just relying on a single method, sometimes chosen somewhat arbitrarily. By significantly simplifying the comparison between different alignment and phylogenetic inference methods, the developed software can help to improve the quality of all kinds of scientific studies involving MSAs or trees. Researchers are also enabled to present results of alternative methods by providing the output of *AlignmentComparator* (both the graphical output or the comparison document in *NeXML* format) or showing different sets of support values on a single tree using the “Add support values” feature of *TreeGraph 2*. This way, variance between different methods can be presented and regions of conflict and agreement can be identified and documented. Which regions of an MSA or a phylogenetic tree topology are well supported among different methods can be more easily considered when applying downstream analyses, instead of having to choose one single result and ignoring variation.

Beyond that, the functionality of both applications can be used to inspect the single steps of a workflow. When MSAs are automatically postprocessed or manually edited or phylogenetic trees are changed, such changes can be documented by storing intermediate results (cf. rules 1 and 5 in [25]) and can be visualized using *AlignmentComparator* and *TreeGraph 2*. This way, the developed tools can also help to increase the reproducibility of studies, which is another important goal of this thesis.

Both applications allow visualizing metadata attached to elements of an MSA or a phylogenetic tree. *AlignmentComparator* can display *LibrAlign* data areas within the comparison, while *TreeGraph 2* offers a set of graphical labels attached to branches. (The integration of custom data areas into *AlignmentComparator* currently would require adjustments of the application, while *TreeGraph 2* already allows to visualize any type of metadata directly. See also chapter 13.1.3 above.) By combining graphical metadata representations with the visual comparisons, the influence of other data or metadata on multiple sequence alignment and phylogenetic inference is directly detectable using the developed applications.

13.2.2 Current and future applications in MSA evaluation and improvement for phylogenetic purposes

Beyond the general use in all studies involving MSAs or phylogenetic trees, one motivation for the development of functionality for fine-grained comparison of multiple sequence alignments and phylogenetic trees was their use in evaluating and improving automated multiple sequence alignment for phylogenetic purposes, especially regarding non-coding chloroplast DNA, which is often used for reconstructing plant phylogenies.

A study to address this is currently underway by the author of this thesis and Kai Müller, in cooperation with Dietmar Quandt, and already makes use of the comparison functionality provided by *AlignmentComparator* and *TreeGraph 2*. An alignment evaluation software is currently being implemented that takes a manually aligned dataset as its input and then realigns its sequences using various available methods for automated MSA. After that, *AlignmentComparator* is called from that software to compare the alternative MSAs. Subsequently, different phylogenetic inference methods are used to reconstruct trees from each alternative MSA and *TreeGraph 2* is used to compare the alternative results and how branches are supported by the different inference methods. Figure 13.3 shows a screenshot of the current version of the alignment evaluation software with integrated components from *TreeGraph 2* to compare alternative phylogenetic trees resulting from the analyses.

Figure 13.4 shows am overview on the structure of the study together with the components developed in this thesis and how they are linked. In blue, at the bottom of the figure, three important aspects on the way to improving automated sequence alignment for non-coding DNA are shown. The evaluation of existing MSA algorithms on respective datasets and comparing them to manual MSAs, as they are still frequently created in phylogenetic studies is one key part of the project. Another is the search for mutational patterns that might be relevant to be modeled in automated MSA to improve its results and to achieve a better homology assessment. Such patterns may include the addition of periods to tandem repeats in one step due to different mechanisms or inversion or relocation of subsequences. (Since the future application of the software developed in this thesis is the focus of this section, we will not go into further detail on the hypothesis regarding the origins and importance of such patterns here.) In the red application part of the figure, *MSMFinder* and *MSMDB* are listed as additional software components to be developed for the study. “MSM” stands for microstructural mutation. While *MSMFinder* is planned as an application to locate mutational patterns as mentioned above in sequences and MSAs (Some can only be identified within the context of an MSA.), *MSMDB* will be a database that stores the results of *MSMFinder* and allows queries to assess the frequencies of the different types of mutational patterns in different genomes, genomic regions and taxa. As depicted in Figure 13.4, *MSMFinder* will be based on *LibrAlign* and use its sequence and MSA visualization functionality combined with one or more custom data area implementations that display the identified mutational patterns.

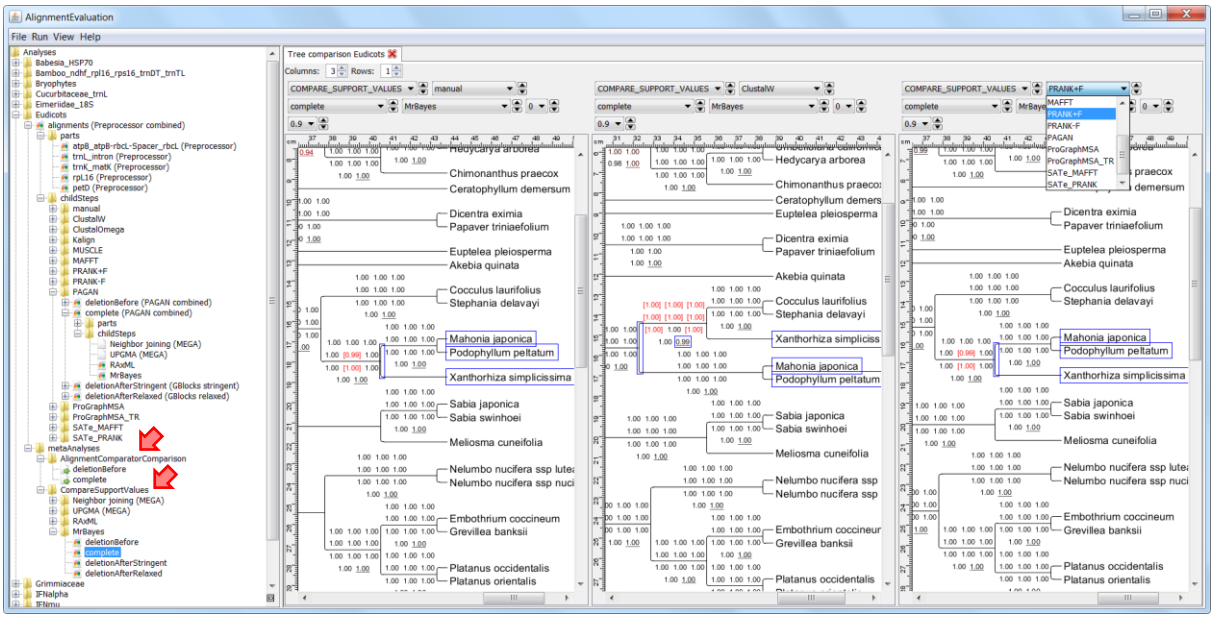


Figure 13.3 Screenshot of the alignment evaluation software used in a currently ongoing study that makes use of comparison functionality developed in this thesis

On the left is the analysis tree that contains all datasets from the current repository and the different MSA and tree inference steps nested in them. Additionally, each dataset contains a set of metaanalyses that compare the results. Highlighted by the red arrows are the steps `AlignmentComparatorComparison` *(which compares MSAs using* AlignmentComparator*) and* `CompareSupportValues` *(which compares phylogenetic trees using* TreeGraph 2.*)*

The tab opened on the right contains a GUI component that based on TreeGraph 2*. The number of columns and rows can be adjusted to determine the number of trees to be compared and the dropdown menus in each tile allow to choose a tree from the current dataset to be compared. As explained in chapter 9.3.1 (page 139) and Figure 9.2 (page 140), a node selected in one of the trees is automatically selected in all others and support values of conflicting branches are also highlighted. In addition, the trees displayed here contain the support values from all alternative trees, which have been mapped onto each branch as described in Figure 9.1 (page 139). Conflicting values are in brackets and all values are colored depending on how much they differ from the average support of that node in all trees. This has been achieved using* TreeGraph 2*'s features to calculate node annotations (chapter 9.3.4, page 148) and to automatically set formats by annotations (chapter 8.3.2.5, page 133), which are called using the* Java *API of* TreeGraph 2 *by the alignment evaluation software.*

In this concrete example the tree topology resulting from the ClustalW *MSA (in the middle) differs from the two others (left and right) and the support value in the* ClustalW *tree for the conflicting node is highlighted. This is the same support value (0.99) which is displayed in red and in brackets in the other two trees. As there is an additional value formatted in brackets, another tree (which is currently not opened in the comparison) conflicts with the manual and* PRANK *tree (left and right) and is in agreement with the* ClustalW *tree.*

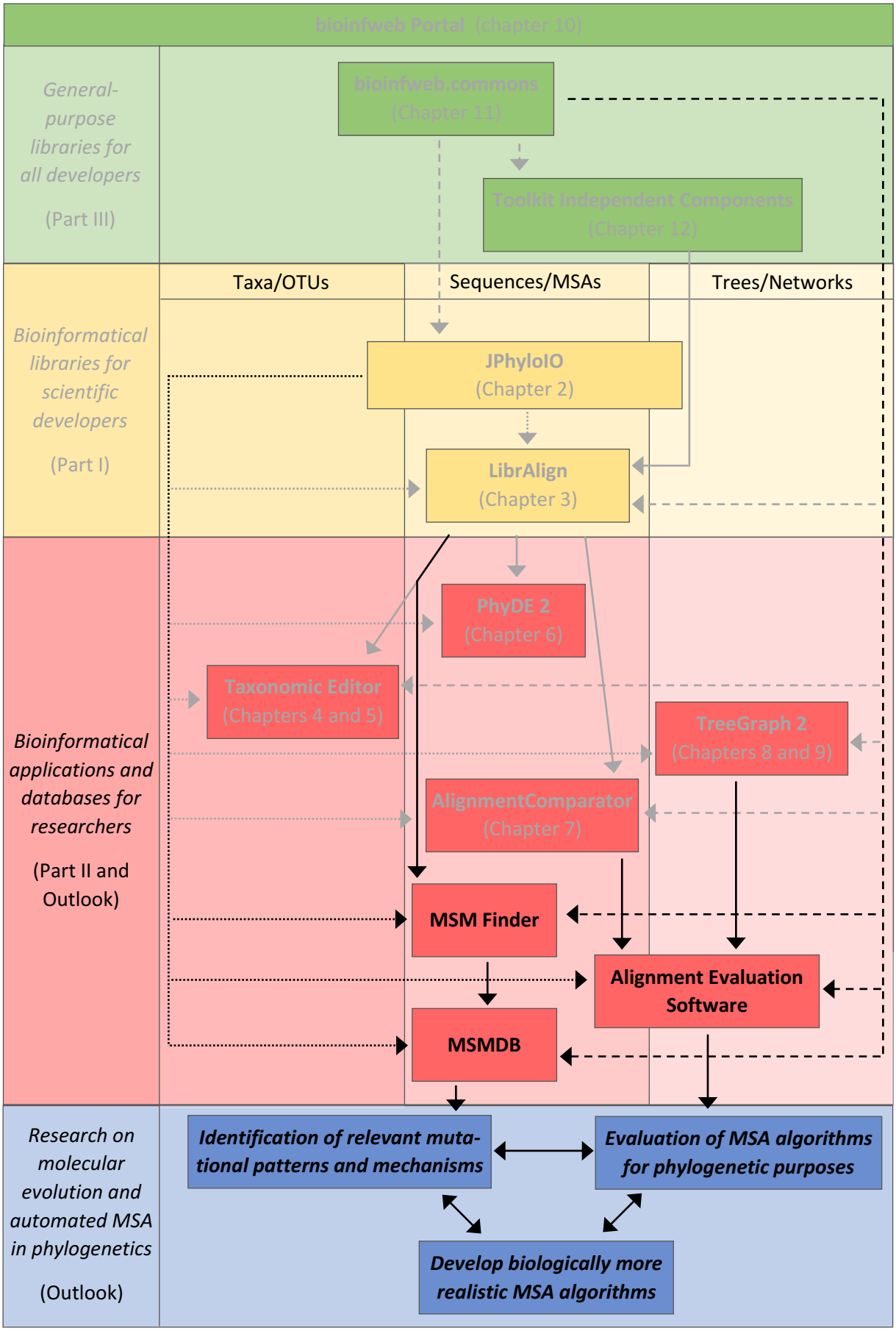


Figure 13.4 Ongoing and planned applications of the software developed in this thesis to investigate microstructural mutational patterns and improve multiple sequence alignment algorithms for phylogenetic purposes *(Caption on the next page.)*

Figure 13.4 *(Continued.)*

This figure shows the relation between software components developed in this thesis and additional ones that are planned or currently implemented. In the blue section, research aims to be addressed using this set of software components are shown. Components and relations that are a result of this thesis and were already contained in Figure 1.1 are shown in gray, while all additional components and relations that are currently developed or planned are shown in black.

("MSM" stands for microstructural mutations and MSMDB *for microstructural mutations database.)*

To decide which mutational patterns are of potential relevance for modeling in automated MSA algorithms, their frequency in different genomes and taxa (as it will be provided by *MSMDB*) is important, but a more fine-grained examination of their influence on the results of automated MSA and subsequent phylogenetic inference is necessary. *AlignmentComparator* with its *LibrAlign*-based architecture represents a very useful tool to perform such an analysis. The special data area(s) that are planned to be implemented in *MSMFinder* can directly be integrated into *AlignmentComparator*, as well, to indicate the positions of the identified mutational patterns in the evaluated data sets directly within the visual comparison. This allows a detailed and efficient inspection of the possible influence of such patterns on the results of different automated MSA algorithms and their possible correlation with manual alignment modifications performed in the investigated phylogenetic studies. Information gathered this way could be vital to improve automated MSA and new or extended algorithms could directly be tested on the relevant patterns. Therefore, the flexible data area architecture provided by *LibrAlign* plays a key role in enabling the combination of the two approaches (MSA evaluation and search for mutational patterns) to improve automated MSA for phylogenetic purposes and makes a study like this feasible.

14 List of abbreviations

ABCD: Access to Biological Collection Data (exchange format)

API: Application programming interface

AWT: Abstract Window Toolkit

BAliBASE: Benchmark Alignment dataBASE

BGBM: Botanical Garden and Botanical Museum (FU Berlin)

BHL: Biodiversity Heritage Library

BioCASe: Biological Collection Access Service

CDAO: Comparative Data Analysis Ontology

CDM: Common Data Model (of the EDIT platform for Cybertaxonomy)

CLD-CoW: Corvids Literature Database: Corvids of the World

DDBJ: DNA DataBank of Japan

DELTA: DEscription Language for Taxonomy

DFG: Deutsche Forschungsgemeinschaft (German Research Fundation)

DNA: Deoxyribonucleic acid

DP: Dynamic Programming

EDAM: EMBRACE Data And Methods

EDIT: European Distributed Institute of Taxonomy

EMBRACE: European Model for Bioinformatics Research and Community Education

ENA: European Nucleotide Archive

ETE: Environment for Tree Exploration

FASTA: Fast Adaptive Shrinkage Thresholding Algorithm (This abbreviation is used here for FASTA alignment format.)

FreeHEP: Free High Energy Physics (*Java* library)

GBIF: Global Biodiversity Information Facility

GFBio: German Federation for Biological Data

GGBN: Global Genome Biodiversity Network

GNU: Gnu's Not Unix (Recursive acronym. Uses a wildebeest (*Connochaetes*) as its icon, which is “Gnu” in German.)

GPL: General Public License

GUI: Graphical User Interface

ICN: International Code of Nomenclature for algae, fungi and plants

ID: Identifier

I/O: Input/Output

LGPL: Lesser General Public License

MIAPA: Minimum Information for A Phylogenetic Analysis

MSA: Multiple sequence alignment

MSM: Microstructural mutation

MSMDB: Microstructural mutations database

NaN: Not a Number

NCBI: National Center for Biotechnology Information

OTU: Operational Taxonomic Unit

OWL: Web Ontology Language (The letter reversal is intended by the authors.)

PDF: Portable Document Format

PhyDE: Phylogenetic Data Editor (described in chapter 6)

PNG: Portable Network Graphics

PRO: Protein Ontology

RCP: Rich Client Platform (*Eclipse*)

RAxML: Randomized Axelerated Maximum Likelihood (computer program)

RDF: Resource Description Framework

RNA: Ribonucleic acid

SDD: Structure of Descriptive Data standard

SEM: Scanning Electron Microscope

SP: Sum-of-Pairs score

SVG: Scalable Vector Graphics

SVN: Subversion (version control system)

SWT: Standard Widget Toolkit

TC: Total Column score

TDWG: Taxonomic Databases Working Group

TIC: Toolkit Independent Components (*Java* library, described in chapter 12)

UML: Unified Modeling Language

URI: Uniform Resource Identifier

URL: Uniform Resource Locator

W3C: World Wide Web Consortium

XML: Extensible Markup Language

XTG: Extensible TreeGraph format

15 References

1. Newman HB, Ellisman MH, Orcutt JA. Data-intensive e-Science Frontier Research. Commun ACM. 2003;46:68–77.

2. Kelling S, Hochachka WM, Fink D, Riedewald M, Caruana R, Ballard G, et al. Data-intensive Science: A New Paradigm for Biodiversity Studies. BioScience. 2009;59:613–20.

3. Michener WK, Jones MB. Ecoinformatics: supporting ecology as a data-intensive science. Trends in Ecology & Evolution. 2012;27:85–93.

4. Mora C, Tittensor DP, Adl S, Simpson AGB, Worm B. How Many Species Are There on Earth and in the Ocean? PLOS Biology. 2011;9:e1001127.

5. McKain MR, Johnson MG, Uribe-Convers S, Eaton D, Yang Y. Practical considerations for plant phylogenomics. Applications in Plant Sciences. 2018;6:e1038.

6. Geiger MF, Astrin JJ, Borsch T, Burkhardt U, Grobe P, Hand R, et al. How to tackle the molecular species inventory for an industrialized nation - lessons from the first phase of the German Barcode of Life initiative GBOL (2012–2015). Genome. 2016;59:661–70.

7. Ratnasingham S, Hebert PDN. bold: The Barcode of Life Data System (http://www.barcodinglife.org). Molecular Ecology Notes. 2007;7:355–64.

8. Kerr KC, Stoeckle MY, Dove CJ, Weigt LA, Francis CM, Hebert PD. Comprehensive DNA barcode coverage of North American birds. Molecular Ecology Resources. 2007;7:535–543.

9. Global Plants on JSTOR [Internet]. [cited 2018 Apr 28]. Available from: https://plants.jstor.org/

10. Smith V, Blagoderov V. Bringing collections out of the dark. ZooKeys. 2012;209:1–6.

11. Dickinson JL, Zuckerberg B, Bonter DN. Citizen Science as an Ecological Research Tool: Challenges and Benefits. Annu Rev Ecol Evol Syst. 2010;41:149–72.

12. Hochachka WM, Fink D, Hutchinson RA, Sheldon D, Wong W-K, Kelling S. Data-intensive science applied to broad-scale citizen science. Trends in Ecology & Evolution. 2012;27:130–7.

13. Joly A, Goëau H, Bonnet P, Bakić V, Barbe J, Selmi S, et al. Interactive plant identification based on social image data. Ecological Informatics. 2014;23:22–34.

14. Schröter M, Kraemer R, Mantel M, Kabisch N, Hecker S, Richter A, et al. Citizen science for assessing ecosystem services: Status, challenges and opportunities. Ecosystem Services. 2017;28:80–94.

15. Hang ST, Tatsuma A, Aono M. Bluefield (kde tut) at lifeclef 2016 plant identification task. Working Notes of CLEF 2016 Conference [Internet]. 2016 [cited 2016 Sep 14]. Available from: http://ceur-ws.org/Vol-1609/16090459.pdf

16. Wilf P, Zhang S, Chikkerur S, Little SA, Wing SL, Serre T. Computer vision cracks the leaf code. PNAS. 2016;113:3305–10.

17. Barré P, Stöver BC, Müller KF, Steinhage V. LeafNet: A computer vision system for automatic plant species identification. Ecological Informatics. 2017;40:50–6.

18. Poisot TE, Mounce R, Gravel D. Moving toward a sustainable ecological science: don’t let data go to waste! Ideas in Ecology and Evolution. 2013;6.

19. Parr CS, Guralnick R, Cellinese N, Page RDM. Evolutionary informatics: unifying knowledge about the diversity of life. Trends in Ecology & Evolution. 2012;27:94–103.

20. White EP, Baldridge E, Brym ZT, Locey KJ, McGlinn DJ, Supp SR. Nine simple ways to make it easier to (re)use your data. 1 [Internet]. 2013 [cited 2018 Jan 23];6. Available from: https://ojs.library.queensu.ca/index.php/IEE/article/view/4608

21. Whitlock MC. Data archiving in ecology and evolution: best practices. Trends in Ecology & Evolution. 2011;26:61–5.

22. Stoltzfus A, O'Meara B, Whitacre J, Mounce R, Gillespie EL, Kumar S, et al. Sharing and re-use of phylogenetic trees (and associated data) to facilitate synthesis. BMC Research Notes. 2012;5:574.

23. Federhen S. The NCBI Taxonomy database. Nucleic Acids Research. 2012;40:D136–43.

24. Kilian N, Henning T, Plitzner P, Müller A, Güntsch A, Stöver BC, et al. Sample data processing in an additive and reproducible taxonomic workflow by using character data persistently linked to preserved individual specimens. Database. 2015;2015:bav094.

25. Sandve GK, Nekrutenko A, Taylor J, Hovig E. Ten Simple Rules for Reproducible Computational Research. PLOS Computational Biology. 2013;9:e1003285.

26. Goble CA, Bhagat J, Aleksejevs S, Cruickshank D, Michaelides D, Newman D, et al. myExperiment: a repository and social network for the sharing of bioinformatics workflows. Nucl Acids Res. 2010;38:W677–82.

27. Köster J, Rahmann S. Snakemake—a scalable bioinformatics workflow engine. Bioinformatics. 2012;28:2520–2.

28. Wolstencroft K, Haines R, Fellows D, Williams A, Withers D, Owen S, et al. The Taverna workflow suite: designing and executing workflows of Web Services on the desktop, web or in the cloud. Nucleic Acids Research. 2013;41:W557–61.

29. Goecks J, Nekrutenko A, Taylor J, Team TG. Galaxy: a comprehensive approach for supporting accessible, reproducible, and transparent computational research in the life sciences. Genome Biol. 2010;11:R86.

30. Okonechnikov K, Golosova O, Fursov M, the UGENE team. Unipro UGENE: a unified bioinformatics toolkit. Bioinformatics. 2012;28:1166–7.

31. Maddison DR, Swofford D, Maddison WP. NEXUS: an extensible file format for systematic information. Systematic Biology. 1997;46:590–621.

32. Felsenstein J. Phylip input files [Internet]. 2013 [cited 2016 Nov 5]. Available from: http://evolution.genetics.washington.edu/phylip/doc/main.html#inputfiles

33. Miller M, Schwartz T, Pfeiffer W. Relaxed PHYLIP Format [Internet]. Relaxed PHYLIP Format documentation at CIPRES. 2016 [cited 2016 Nov 5]. Available from: http://www.phylo.org/index.php/help/relaxed_phylip

34. Felsenstein J. The Newick tree format [Internet]. 1993 [cited 2016 Nov 5]. Available from: http://evolution.genetics.washington.edu/phylip/newicktree.html

35. Vos RA, Balhoff JP, Caravas JA, Holder MT, Lapp H, Maddison WP, et al. NeXML: Rich, Extensible, and Verifiable Representation of Comparative Data and Metadata. Systematic Biology. 2012;61:675–89.

36. Han MV, Zmasek CM. phyloXML: XML for evolutionary biology and comparative genomics. BMC Bioinformatics. 2009;10:356.

37. RDF - Semantic Web Standards [Internet]. [cited 2018 Jan 29]. Available from: https://www.w3.org/RDF/

38. Martínez-Romero M, Jonquet C, O'Connor MJ, Graybeal J, Pazos A, Musen MA. NCBO Ontology Recommender 2.0: an enhanced approach for biomedical ontology recommendation. Journal of Biomedical Semantics. 2017;8:21.

39. Smith B, Ashburner M, Rosse C, Bard J, Bug W, Ceusters W, et al. The OBO Foundry: coordinated evolution of ontologies to support biomedical data integration. Nat Biotech. 2007;25:1251–5.

40. Leebens-Mack J, Vision T, Brenner E, Bowers JE, Cannon S, Clement MJ, et al. Taking the First Steps towards a Standard for Reporting on Phylogenies: Minimum Information about a Phylogenetic Analysis (MIAPA). OMICS: A Journal of Integrative Biology. 2006;10:231–7.

41. Wang X, Gorlitsky R, Almeida JS. From XML to RDF: how semantic web technologies will change the design of "omic" standards. Nat Biotech. 2005;23:1099–103.

42. National Evolutionary Synthesis Center, UNC-CH Metadata Research Center, Oxford University, The British Library. Dryad [Internet]. Dryad. [cited 2018 Jan 29]. Available from: http://datadryad.org/

43. McQuilton P, Gonzalez-Beltran A, Rocca-Serra P, Thurston M, Lister A, Maguire E, et al. BioSharing: curated and crowd-sourced metadata standards, databases and data policies in the life sciences. Database. 2016;2016:baw075.

44. Piel WH, Chan L, Dominus MJ, Ruan J, Vos RA, Tannen V. TreeBASE v. 2: a database of phylogenetic knowledge. e-BioSphere 2009 [Internet]. London; 2009 [cited 2017 Feb 28]. Available from: http://www.citeulike.org/group/894/article/4680867

45. Moore AJ, Mcpeek MA, Rausher MD, Rieseberg L, Whitlock MC. The need for archiving data in evolutionary biology. Journal of Evolutionary Biology. 2010;23:659–60.

46. Kenall A, Harold S, Foote C. An open future for ecological and evolutionary data? BMC Evolutionary Biology. 2014;14:66.

47. Magee AF, May MR, Moore BR. The Dawn of Open Access to Phylogenetic Data. PLOS ONE. 2014;9:e110268.

48. Laubach T, von Haeseler A, Lercher MJ. TreeSnatcher plus: capturing phylogenetic trees from images. BMC Bioinformatics. 2012;13:110.

49. Murray-Rust P, Smith-Unna R, Mounce R. AMI-diagram: Mining Facts from Images. D-Lib Magazine [Internet]. 2014 [cited 2018 Apr 30];20. Available from: http://www.dlib.org/dlib/november14/murray-rust/11murray-rust.html

50. Drew BT, Gazis R, Cabezas P, Swithers KS, Deng J, Rodriguez R, et al. Lost Branches on the Tree of Life. PLOS Biology. 2013;11:e1001636.

51. Rusher J. Triple Store [Internet]. 2004 [cited 2018 Apr 29]. Available from: https://www.w3.org/2001/sw/Europe/events/20031113-storage/positions/rusher.html

52. Stoltzfus A, Lapp H, Matasci N, Deus H, Sidlauskas B, Zmasek CM, et al. Phylotastic! Making tree-of-life knowledge accessible, reusable and convenient. BMC Bioinformatics. 2013;14:158.

53. Hinchliff CE, Smith SA, Allman JF, Burleigh JG, Chaudhary R, Coghill LM, et al. Synthesis of phylogeny and taxonomy into a comprehensive tree of life. PNAS. 2015;112:12764–9.

54. Pagel M, Meade A, Barker D. Bayesian Estimation of Ancestral Character States on Phylogenies. Syst Biol. 2004;53:673–84.

55. Tamura K, Battistuzzi FU, Billing-Ross P, Murillo O, Filipski A, Kumar S. Estimating divergence times in large molecular phylogenies. PNAS. 2012;109:19333–8.

56. Mellmann A, Bielaszewska M, Köck R, Friedrich AW, Fruth A, Middendorf B, et al. Analysis of collection of hemolytic uremic syndrome - associated enterohemorrhagic Escherichia coli. Emerging Infectious Diseases - CDC. 2008;14:1287.

57. Mayrose I, Otto SP. A Likelihood Method for Detecting Trait-Dependent Shifts in the Rate of Molecular Evolution. Mol Biol Evol. 2011;28:759–70.

58. Levy Karin E, Wicke S, Pupko T, Mayrose I. An Integrated Model of Phenotypic Trait Changes and Site-Specific Sequence Evolution. Syst Biol. 2017;66:917–33.

59. Holder M, Lewis PO. Phylogeny estimation: Traditional and Bayesian approaches. Nat Rev Genet. 2003;4:275–84.

60. Katoh K, Standley DM. MAFFT Multiple Sequence Alignment Software Version 7: Improvements in Performance and Usability. Mol Biol Evol. 2013;30:772–80.

61. Notredame C, Higgins DG, Heringa J. T-coffee: a novel method for fast and accurate multiple sequence alignment. J Mol Biol. 2000;302:205–17.

62. Do CB, Mahabhashyam MSP, Brudno M, Batzoglou S. ProbCons: Probabilistic consistency-based multiple sequence alignment. Genome Res. 2005;15:330–40.

63. Sievers F, Wilm A, Dineen D, Gibson TJ, Karplus K, Li W, et al. Fast, scalable generation of high-quality protein multiple sequence alignments using Clustal Omega. Molecular Systems Biology [Internet]. 2011 [cited 2012 Jun 6];7. Available from: http://www.nature.com/msb/journal/v7/n1/full/msb201175.html

64. Lassmann T, Frings O, Sonnhammer ELL. Kalign2: high-performance multiple alignment of protein and nucleotide sequences allowing external features. Nucl Acids Res. 2009;37:858–65.

65. Löytynoja A, Goldman N. Phylogeny-aware gap placement prevents errors in sequence alignment and evolutionary analysis. Science. 2008;320:1632–1635.

66. Löytynoja A, Vilella AJ, Goldman N. Accurate extension of multiple sequence alignments using a phylogeny-aware graph algorithm. Bioinformatics. 2012;28:1684–91.

67. Szalkowski AM, Anisimova M. Graph-based modeling of tandem repeats improves global multiple sequence alignment. Nucl Acids Res. 2013;41:e162–e162.

68. Edgar RC. MUSCLE: multiple sequence alignment with high accuracy and high throughput. Nucl Acids Res. 2004;32:1792–7.

69. Kumar S, Stecher G, Tamura K. MEGA7: Molecular Evolutionary Genetics Analysis Version 7.0 for Bigger Datasets. Mol Biol Evol. 2016;msw054.

70. Swofford DL. PAUP*: Phylogenetic Analysis Using Parsimony. (* and other methods). 2003.

71. Stamatakis A. RAxML version 8: a tool for phylogenetic analysis and post-analysis of large phylogenies. Bioinformatics. 2014;30:1312–3.

72. Yang Z. PAML 4: phylogenetic analysis by maximum likelihood. Mol Biol Evol. 2007;24:1586–91.

73. Ronquist F, Teslenko M, Mark P van der, Ayres DL, Darling A, Höhna S, et al. MrBayes 3.2: Efficient Bayesian Phylogenetic Inference and Model Choice Across a Large Model Space. Syst Biol. 2012;61:539–42.

74. Bouckaert R, Heled J, Kühnert D, Vaughan T, Wu C-H, Xie D, et al. BEAST 2: A Software Platform for Bayesian Evolutionary Analysis. PLOS Comput Biol. 2014;10:e1003537.

75. Müller J, Müller K, Neinhuis C, Quandt D. PhyDE - Phylogenetic Data Editor [Internet]. 2006 [cited 2016 Nov 5]. Available from: http://phyde.de/

76. Stöver BC, Müller KF. TreeGraph 2: Combining and visualizing evidence from different phylogenetic analyses. BMC Bioinformatics. 2010;11:7.

77. Thompson JD, Higgins DG, Gibson TJ. CLUSTALW: improving the sensitivity of progressive multiple sequence alignment through sequence weighting, position specific gap penalties and weight matrix choice. Nucleic Acids Res. 1994;22:4673–80.

78. Waterhouse AM, Procter JB, Martin DMA, Clamp M, Barton GJ. Jalview Version 2—a multiple sequence alignment editor and analysis workbench. Bioinformatics. 2009;25:1189–1191.

79. Löytynoja A, Goldman N. webPRANK: a phylogeny-aware multiple sequence aligner with interactive alignment browser. BMC Bioinformatics. 2010;11:579.

80. Huelsenbeck J, Ronquist F, Hall B. MrBayes: a program for the Bayesian inference of phylogeny. Version 3.0 b4. University of Rochester; 2003.

81. Höhna S, Landis MJ, Heath TA, Boussau B, Lartillot N, Moore BR, et al. RevBayes: Bayesian Phylogenetic Inference Using Graphical Models and an Interactive Model-Specification Language. Syst Biol. 2016;65:726–36.

82. Morell V. The Roots of Phylogeny. Science. 1996;273:569.

83. Maddison DR, Schulz K-S, Maddison WP. The tree of life web project. Zootaxa. 2007;1668:19–40.

84. Maddison WP, Maddison DR. Mesquite: a modular system for evolutionary analysis [Internet]. 2016 [cited 2016 Nov 5]. Available from: http://mesquiteproject.org/

85. Zmasek CM, Eddy SR. ATV: display and manipulation of annotated phylogenetic trees. Bioinformatics. 2001;17:383–4.

86. Troshin PV, Procter JB, Barton GJ. Java bioinformatics analysis web services for multiple sequence alignment—JABAWS:MSA. Bioinformatics. 2011;27:2001–2.

87. Huson DH, Scornavacca C. Dendroscope 3: An Interactive Tool for Rooted Phylogenetic Trees and Networks. Syst Biol. 2012;61:1061–7.

88. Gamma E, Helm R, Johnson RE, Vlissides J. Design Patterns. Elements of Reusable Object-Oriented Software. 1st ed., Reprint. Reading, Mass: Prentice Hall; 1994.

89. Ronquist F, Teslenko M, van der Mark P, Larget B, Donald S, Huelsenbeck J. The MrBayes Input Format [Internet]. 2016 [cited 2016 Nov 5]. Available from: http://mrbayes.sourceforge.net/Help/format.html

90. Cardona G, Rosselló F, Valiente G. Extended Newick: it is time for a standard representation of phylogenetic networks. BMC Bioinformatics. 2008;9:532.

91. Zmasek CM. NHX - New Hampshire eXtended, version 2.0 [Internet]. 2014 [cited 2016 Nov 5]. Available from: https://sites.google.com/site/cmzmasek/home/software/forester/nhx

92. Zmasek CM. forester: software libraries for evolutionary biology and comparative genomics research [Internet]. 2015 [cited 2016 Nov 5]. Available from: https://sites.google.com/site/cmzmasek/home/software/forester

93. Prlić A, Yates A, Bliven SE, Rose PW, Jacobsen J, Troshin PV, et al. BioJava: an open-source framework for bioinformatics in 2012. Bioinformatics. 2012;28:2693–5.

94. Holland RCG, Down TA, Pocock M, Prlic A, Huen D, James K, et al. BioJava: an open-source framework for bioinformatics. Bioinformatics. 2008;24:2096–7.

95. Hladish T, Gopalan V, Liang C, Qiu W, Yang P, Stoltzfus A. Bio::NEXUS: a Perl API for the NEXUS format for comparative biological data. BMC Bioinformatics. 2007;8:1.

96. Boettiger C, Chamberlain S, Vos R, Lapp H. RNeXML: a package for reading and writing richly annotated phylogenetic, character and trait data in r. Methods Ecol Evol. 2016;7:352–7.

97. Stajich JE, Block D, Boulez K, Brenner SE, Chervitz SA, Dagdigian C, et al. The Bioperl toolkit: Perl modules for the life sciences. Genome Research. 2002;12:1611–8.

98. Lewis PO. NCL: a C++ class library for interpreting data files in NEXUS format. Bioinformatics. 2003;19:2330–2331.

99. Sukumaran J, Holder MT. DendroPy: a Python library for phylogenetic computing. Bioinformatics. 2010;26:1569–71.

100. Vos RA, Caravas J, Hartmann K, Jensen MA, Miller C. BIO::Phylo-phyloinformatic analysis using perl. BMC Bioinformatics. 2011;12:63.

101. Vogt L. eScience and the need for data standards in the life sciences: in pursuit of objectivity rather than truth. Systematics and Biodiversity. 2013;11:257–70.

102. Berendsohn WG. Devising the EDIT Platform for Cybertaxonomy. Tools for Identifying Biodiversity: Progress and Problems Proceedings of the International Congress, Paris, September 20-22, 2010 [Internet]. EUT Edizioni Università di Trieste; 2010 [cited 2016 May 11]. Available from: http://www.openstarts.units.it/dspace/handle/10077/3737

103. Preston-Werner T. Semantic Versioning 2.0.0 [Internet]. 2013 [cited 2016 Nov 5]. Available from: http://semver.org/

104. Guindon C. SWT: The Standard Widget Toolkit [Internet]. [cited 2017 May 22]. Available from: https://www.eclipse.org/swt/

105. Rich Client Platform - Eclipsepedia [Internet]. [cited 2017 May 22]. Available from: https://wiki.eclipse.org/Rich_Client_Platform

106. Krasner GE, Pope ST, others. A description of the model-view-controller user interface paradigm in the smalltalk-80 system. Journal of object oriented programming. 1988;1:26–49.

107. Morrison D, Morgan M, Kelchner S. Molecular homology and multiple sequence alignment: an analysis of concepts and practice. Australian Systematic Botany [Internet]. 2015 [cited 2015 Jul 20]; Available from: http://www.publish.csiro.au/view/journals/dsp_journals_pip_abstract_Scholar1.cfm?nid=150&pip=SB15001

108. Spjuth O, Alvarsson J, Berg A, Eklund M, Kuhn S, Mäsak C, et al. Bioclipse 2: A scriptable integration platform for the life sciences. BMC bioinformatics. 2009;10:397.

109. The Collections Framework [Internet]. [cited 2017 May 22]. Available from: https://docs.oracle.com/javase/8/docs/technotes/guides/collections/index.html

110. Larsson A. AliView: a fast and lightweight alignment viewer and editor for large datasets. Bioinformatics. 2014;30:3276–8.

111. Maddison WP, Maddison DR. Mesquite: a modular system for evolutionary analysis [Internet]. 2016. Available from: http://mesquiteproject.org

112. Gille C, Frömmel C. STRAP: editor for STRuctural Alignments of Proteins. Bioinformatics. 2001;17:377–8.

113. Caffrey DR, Dana PH, Mathur V, Ocano M, Hong E-J, Wang YE, et al. PFAAT version 2.0: A tool for editing, annotating, and analyzing multiple sequence alignments. BMC Bioinformatics. 2007;8:381.

114. Gouy M, Guindon S, Gascuel O. SeaView Version 4: A Multiplatform Graphical User Interface for Sequence Alignment and Phylogenetic Tree Building. Mol Biol Evol. 2010;27:221–4.

115. Sanchez-Villeda H, Schroeder S, Flint-Garcia S, Guill KE, Yamasaki M, McMullen MD. DNAAlignEditor: DNA alignment editor tool. BMC Bioinformatics. 2008;9:154.

116. Bond CS, Schuttelkopf AW. ALINE: a WYSIWYG protein-sequence alignment editor for publication-quality alignments. Acta Crystallographica Section D: Biological Crystallography. 2009;65:510–512.

117. Yachdav G, Wilzbach S, Rauscher B, Sheridan R, Sillitoe I, Procter J, et al. MSAViewer: interactive JavaScript visualization of multiple sequence alignments. Bioinformatics. 2016;32:3501–3.

118. NCBI Multiple Sequence Alignment Viewer Embedding API [Internet]. [cited 2017 May 16]. Available from: https://www.ncbi.nlm.nih.gov/tools/msaviewer/embedding-api/

119. Veidenberg A, Medlar A, Löytynoja A. Wasabi: An Integrated Platform for Evolutionary Sequence Analysis and Data Visualization. Mol Biol Evol. 2016;33:1126–30.

120. Gille C, Fähling M, Weyand B, Wieland T, Gille A. Alignment-Annotator web server: rendering and annotating sequence alignments. Nucleic Acids Res. 2014;42:W3–6.

121. Java(TM) Web Start [Internet]. [cited 2017 May 31]. Available from: https://docs.oracle.com/javase/8/docs/technotes/guides/javaws/

122. Huerta-Cepas J, Serra F, Bork P. ETE 3: Reconstruction, Analysis, and Visualization of Phylogenomic Data. Mol Biol Evol. 2016;33:1635–8.

123. Stuessy TF, Crawford DJ, Soltis DE, Soltis PS. Plant systematics: the origin, interpretation, and ordering of plant biodiversity. Koenigstein: Koeltz Scientific Books; 2014.

124. Dallwitz MJ. User's guide to the DELTA system. A general system for coding taxonomic descriptions. 71 pp. CSIRO Division of Entomology Report; 1980.

125. Pullan MR, Armstrong KE, Paterson T, Cannon A, Kennedy JB, Watson MF, et al. The Prometheus Description Model: an examination of the taxonomic description-building process and its representation. Taxon. 2005;54:751–765.

126. Sokol RS, Sneath PH. Principles of numerical taxonomy. WH Freeman and Co; 1963.

127. Pullan MR, Watson MF, Kennedy JB, Raguenaud C, Hyam R. The Prometheus Taxonomic Model: a practical approach to representing multiple classifications. Taxon. 2000;55–75.

128. Borsch T, Hernández-Ledesma P, Berendsohn WG, Flores-Olvera H, Ochoterena H, Zuloaga FO, et al. An integrative and dynamic approach for monographing species-rich plant groups – Building the global synthesis of the angiosperm order Caryophyllales. Perspectives in Plant Ecology, Evolution and Systematics. 2015;17:284–300.

129. Redford AE. Vascular plant systematics. Harper & Row; 1974.

130. Berendsohn WG. The concept of" potential taxa" in databases. Taxon. 1995;207–212.

131. Franz NM, Peet RK. Perspectives: towards a language for mapping relationships among taxonomic concepts. Systematics and Biodiversity. 2009;7:5–20.

132. Franz NM, Cardona-Duque J. Description of two new species and phylogenetic reassessment of Perelleschus O'Brien & Wibmer, 1986 (Coleoptera: Curculionidae), with a complete taxonomic concept history of Perelleschus sec. Franz & Cardona-Duque, 2013. Systematics and Biodiversity. 2013;11:209–236.

133. Schoch CL, Robbertse B, Robert V, Vu D, Cardinali G, Irinyi L, et al. Finding needles in haystacks: linking scientific names, reference specimens and molecular data for Fungi. Database. 2014;2014.

134. Bachmann K. Species as units of diversity: an outdated concept. Theory in Biosciences. 1998;117:213–230.

135. Judd WS, Campbell CS, Kellogg EA, Stevensen PF. Plant systematics. A phylogenetic approach. Sunderland, Mass.: Sinauer Ass; 1999.

136. Stuessy TF. Paradigms in biological classification (1707–2007): has anything really changed? Taxon. 2009;58:68–76.

137. Stuessy TF, Lack HW. Monographic Plant Systematics: Fundamental Assessment of Plant Biodiversity [Internet]. Gantner; 2011 [cited 2018 Apr 23]. Available from: https://www.nhbs.com/monographic-plant-systematics-book

138. Marhold K, Stuessy T, Agababian M, Agosti D, Alford MH, Crespo A, et al. The future of botanical monography: Report from an international workshop, 12–16 March 2012, Smolenice, Slovak Republic. Taxon. 2013;62:4–20.

139. Samper C. Taxonomy and environmental policy. Philosophical Transactions of the Royal Society B: Biological Sciences. 2004;359:721–728.

140. Godfray HCJ, Clark BR, Kitching IJ, Mayo SJ, Scoble MJ. The Web and the Structure of Taxonomy. Systematic Biology. 2007;56:943–55.

141. Scoble MJ, Clark BR, Godfray HCJ, Kitching IJ, Mayo SJ. Revisionary taxonomy in a changing e-landscape. Tijdschrift voor entomologie. 2007;150:305–317.

142. Mayo SJ, Allkin R, Baker W, Blagoderov V, Brake I, Clark B, et al. Alpha e-taxonomy: responses from the systematics community to the biodiversity crisis. Kew Bulletin. 2008;63:1–16.

143. Smith VS, Rycroft SD, Harman KT, Scott B, Roberts D. Scratchpads: a data-publishing framework to build, share and manage information on the diversity of life. BMC bioinformatics. 2009;10:S6.

144. Diederich J. Basic properties for biological databases: character development and support. Mathematical and Computer Modelling. 1997;25:109–127.

145. Dallwitz MJ. A general system for coding taxonomic descriptions. Taxon. 1980;41–46.

146. Lucidcentral [Internet]. 1999 [cited 2018 Apr 23]. Available from: http://www.lucidcentral.org/

147. Hagedorn G, Rambold G. A method to establish and revise descriptive data sets over the Internet. Taxon. 2000;517–528.

148. Ung V, Causse F, Vignes Lebbe R. Xper^2: managing descriptive data from their collection to e-monographs. Tools for Identifying Biodiversity: Progress and Problems. EUT Edizioni Università di Trieste; 2010.

149. Hagedorn G, Thiele K, Morris R, Heidorn PB. The Structured Descriptive Data (SDD) w3c-xml-schema, Version 1.1 [Internet]. 2006 [cited 2018 Apr 23]. Available from: https://github.com/tdwg/wiki-archive/blob/master/twiki/data/SDD/CurrentSchemaVersion.txt

150. Dallwitz MJ. A comparison of formats for descriptive data. Institute of Botany, Chinese Academy of Sciences(2 August 2013. 2010;

151. Raguenaud C, Pullan MR, Watson MF, Kennedy JB, Newman MF, Barclay PJ. Implementation of the Prometheus Taxonomic Model: a comparison of database models and query languages and an introduction to the Prometheus Object-Oriented Model. Taxon. 2002;51:131–142.

152. Stevens PF. On Phylogenies and Data Bases: Where Are the Data, or Are There Any? Taxon. 1996;95–98.

153. McNeill J, Barrie FR, Buck WR, Demoulin V, Greuter W, Hawksworth DL, et al. International Code of Nomenclature for algae, fungi, and plants (Melbourne Code) adopted by the Eighteenth International Botanical Congress Melbourne. Regnum vegetabile [Internet]. 2012;154. Available from: http://www.iapt-taxon.org/nomen/main.php

154. Berendsohn WG, Anagnostopoulos A, Hagedorn G, Jakupovic J, Nimis PL, Valdés B, et al. A comprehensive reference model for biological collections and surveys. Taxon. 1999;511–562.

155. Berendsohn WG, Nimis PL. The complexity of collection information. Resource Identification for a Biological Collection Information Service in Europe (BioCISE). Berlin: BGBM; 2000. p. 13–8.

156. Berendsohn WG. MoReTax: handling factual information linked to taxonomic concepts in biology. Federal Agency for Nature Conservation; 2003. p. 1–115.

157. Berendsohn WG. Access to Biological Collection Data [Internet]. ABCD Schema 2.06 - ratified TDWG Standard. 2007 [cited 2018 Apr 23]. Available from: http://www.bgbm.org/TDWG/CODATA/Schema/default.htm

158. Robertson T, Döring M, Wieczorek J. Darwin Core Text Guide. Biodiversity Information Standards (TDWG) [Internet]. 2009 [cited 2018 Apr 23]. Available from: http://rs.tdwg.org/dwc/terms/guides/text/index.htm

159. Ciardelli P, Kelbert P, Kohlbecker A, Hoffmann N, Güntsch A, Berendsohn WG. The EDIT Cyberplatform for Taxonomy and the Taxonomic Workflow: Selected Components. Im Focus das Leben [Internet]. Bonn: Gesellschaft für Informatik; 2009. p. 625–638. Available from: http://subs.emis.de/LNI/Proceedings/ Proceedings154/article4943.html

160. Berendsohn WG, Güntsch A, Hoffmann N, Kohlbecker A, Luther K, Müller A. Biodiversity information platforms: From standards to interoperability. ZooKeys. 2011;71.

161. Common Data Model [Internet]. 2008 [cited 2016 May 11]. Available from: http://dev.e-taxonomy.eu/trac/wiki/CommonDataModel

162. Berendsohn WG. A taxonomic information model for botanical databases: the IOPI model. Taxon. 1997;283–309.

163. Berendsohn WG, Döring M, Geoffroy M, Glück K, Güntsch A, Hahn A, et al. The Berlin model: a concept-based taxonomic information model. Schriftenreihe für Vegetationskunde. 2003;39:15–42.

164. Geoffroy M, Berendsohn WG. The concept problem in taxonomy: importance, components, approaches. Schriftenreihe Vegetationsk. 2003;39:14.

165. Geoffroy M, Berendsohn WG. Transmission of taxon-related factual information. Schriftenreihe Vegetationsk. 2003;39:83–86.

166. Berendsohn WG, Geoffroy M. Networking taxonomic concepts—uniting without 'unitary-ism' In GB Curry & CJ Humphries (Eds.), Biodiversity databases: Techniques, politics, and applications (pp. 13–22). Boca Raton: CRC Google Scholar. 2007;

167. TDWG Secretariat. Biodiversity Information Standards [Internet]. TDWG. [cited 2018 Apr 23]. Available from: http://www.tdwg.org/

168. Hyam R, Kennedy J. The Taxon Concept Schema [Internet]. [cited 2018 Apr 23]. Available from: http://www.tdwg.org/activities/tnc/tcs-schema-repository/

169. GBIF. GBIF species API [Internet]. 2013 [cited 2018 Apr 23]. Available from: https://www.gbif.org/developer/species

170. Metawiki Contributors. Biowikifarm Metawiki [Internet]. 2011 [cited 2018 Apr 23]. Available from: http://biowikifarm.net/meta/Biowikifarm

171. Scratchpads [Internet]. 2006 [cited 2018 Apr 23]. Available from: http://scratchpads.eu/

172. Agosti D, Klingenberg C, Catapano T, Sautter G. Plazi [Internet]. [cited 2018 Apr 23]. Available from: http://www.plazi.org/

173. Vicario S, Hardisty A, Haitas N. Biovel: biodiversity virtual e-laboratory. EMBnet journal. 2011;17:pp–5.

174. Biodiversity Heritage Library [Internet]. 2014 [cited 2018 Apr 23]. Available from: https://www.biodiversitylibrary.org/

175. Ride W, Cogger HG, Dupuis C. International code of zoological nomenclature, 4th edn [Internet]. London: International Trust for Zoological Nomenclature, The Natural History Museum; 1999. Available from: http://www.nhm.ac.uk/hosted-sites/ iczn/code/

176. Hand R, Kilian N, von Raab-Straube E. International Cichorieae Network: Cichorieae Portal [Internet]. 2009 [cited 2018 Apr 23]. Available from: http://cichorieae.e-taxonomy.net/portal/

177. Droege G. CLD-CoW Portal— Corvids Literature Database: Corvids of the World [Internet]. 2013. Available from: http://dataportal.corvids.de/

178. Palmweb. Palmweb: Palms of the World Online [Internet]. 2014 [cited 2018 Apr 23]. Available from: http://www.palmweb.org/

179. Hand R, Hadjikyriakou GN, Christodoulou CS. Flora of Cyprus - a dynamic checklist [Internet]. 2011 [cited 2018 Apr 23]. Available from: http://www.flora-of-cyprus.eu/

180. Hamann TD, Müller A, Roos MC, Sosef M, Smets E. Detailed mark-up of semi-monographic legacy taxonomic works using FlorML. Taxon. 2014;63:377–393.

181. BioCASE. Biological Collection Access Service for Europe (BioCASE) [Internet]. 2005 [cited 2018 Apr 23]. Available from: http://www.biocase.org/

182. Droege G, Barker K, Astrin JJ, Bartels P, Butler C, Cantrill D, et al. The global genome biodiversity network (GGBN) data portal. Nucleic acids research. 2013;42:D607–D612.

183. Tschöpe O, Macklin JA, Morris RA, Suhrbier L, Berendsohn WG. Annotating biodiversity data via the Internet. Taxon. 2013;62:1248–1258.

184. Tamura K, Stecher G, Peterson D, Filipski A, Kumar S. MEGA6: Molecular Evolutionary Genetics Analysis Version 6.0. Mol Biol Evol. 2013;30:2725–9.

185. Campanula Portal. The Campanula Data Portal [Internet]. 2013 [cited 2018 Apr 24]. Available from: http://campanula.e-taxonomy.net/portal/

186. Güntsch A, Berendsohn WG, Ciardelli P, Hahn A, Kusber WH, Li J. Adding content to content, a generic annotation system for biodiversity data. Studi Trent Sci Nat. 2009;84:123–128.

187. Vogt L, Grobe P. Morph· D· Base—Eine online Datenbank für morphologische Daten und Metadaten. GfBS Newsletter. 2010;24:29–34.

188. Diepenbroek M, Glöckner FO, Grobe P, Güntsch A, Huber R, König-Ries B, et al. Towards an integrated biodiversity and ecological research data management and archiving platform: the German federation for the curation of biological data (GFBio). Informatik 2014. 2014;

189. Vines TH, Albert AY, Andrew RL, Débarre F, Bock DG, Franklin MT, et al. The availability of research data declines rapidly with article age. Current biology. 2014;24:94–97.

190. Güntsch A, Fichtmüller D, Kirchhoff A, Berendsohn WG. Efficient rescue of threatened biodiversity data using reBiND workflows. Plant Biosystems-An International Journal Dealing with all Aspects of Plant Biology. 2012;146:752–755.

191. Bach K, Schäfer D, Enke N, Seeger B, Gemeinholzer B, Bendix J. A comparative evaluation of technical solutions for long-term data repositories in integrative biodiversity research. Ecological Informatics. 2012;11:16–24.

192. Hyam R, Drinkwater RE, Harris DJ. Stable citations for herbarium specimens on the internet: an illustration from a taxonomic revision of Duboscia (Malvaceae). Phytotaxa. 2012;73:17–30.

193. Güntsch A, Hagedorn G. Stable identifiers for specimens–A CETAF ISTC initiative supported by proiBiosphere. 2013.

194. GFBio Terminology Server. Terminology Server of the German Federation for Biological Data (GFBio) [Internet]. 2015 [cited 2018 Apr 24]. Available from: https://terminologies.gfbio.org/

195. Müller J, Müller K, Neinhuis C, Quandt D. PhyDE - Phylogenetic Data Editor. 2006.

196. Plitzner P, Müller A, Güntsch A, Berendsohn W, Kohlbecker A, Kilian N, et al. The CDM Applied: Unit-Derivation, from Field Observations to DNA Sequences. Proceedings of TDWG. 2017;1:e20366.

197. Clamp M, Cuff J, Searle SM, Barton GJ. The Jalview Java alignment editor. Bioinformatics. 2004;20:426–7.

198. Liu K, Warnow TJ, Holder MT, Nelesen SM, Yu J, Stamatakis AP, et al. SATé-II: Very Fast and Accurate Simultaneous Estimation of Multiple Sequence Alignments and Phylogenetic Trees. Syst Biol. 2012;61:90–106.

199. Suchard MA, Redelings BD. BAli-Phy: simultaneous Bayesian inference of alignment and phylogeny. Bioinformatics. 2006;22:2047–8.

200. Li C, Medlar A, Löytynoja A. Co-estimation of Phylogeny-aware Alignment and Phylogenetic Tree. bioRxiv. 2016;077503.

201. Lyras DP, Metzler D. ReformAlign: improved multiple sequence alignments using a profile-based meta-alignment approach. BMC Bioinformatics. 2014;15:265.

202. Thompson JD, Thierry JC, Poch O. RASCAL: rapid scanning and correction of multiple sequence alignments. Bioinformatics. 2003;19:1155–61.

203. Morrison DA. Is Sequence Alignment an Art or a Science? Systematic Botany. 2015;40:14–26.

204. Hogeweg P, Hesper B. The alignment of sets of sequences and the construction of phyletic trees: An integrated method. J Mol Evol. 1984;20:175–86.

205. Feng D-F, Doolittle RF. Progressive sequence alignment as a prerequisite to correct phylogenetic trees. J Mol Evol. 1987;25:351–60.

206. Gusfield D. Efficient methods for multiple sequence alignment with guaranteed error bounds. Bulletin of Mathematical Biology. 1993;55:141–54.

207. Edgar RC. MUSCLE: a multiple sequence alignment method with reduced time and space complexity. BMC Bioinformatics. 2004;5:113.

208. Bayer R. Symmetric binary B-Trees: Data structure and maintenance algorithms. Acta Informatica. 1972;1:290–306.

209. Guibas LJ, Sedgewick R. A dichromatic framework for balanced trees. Foundations of Computer Science, 1978, 19th Annual Symposium on. IEEE; 1978. p. 8–21.

210. Needleman SB, Wunsch CD, others. A general method applicable to the search for similarities in the amino acid sequence of two proteins. Journal of Molecular Biology. 1970;48:443–453.

211. Levenshtein VI. Binary codes capable of correcting deletions, insertions, and reversals. Soviet physics doklady. 1966. p. 707–710.

212. Waterman MS, Smith TF, Beyer WA. Some biological sequence metrics. Advances in Mathematics. 1976;20:367–387.

213. Wang L, Jiang T. On the complexity of multiple sequence alignment. Journal of computational biology. 1994;1:337–348.

214. Saitou N, Nei M. The neighbor-joining method: a new method for reconstructing phylogenetic trees. Molecular biology and evolution. 1987;4:406–425.

215. Studier JA, Keppler KJ. A note on the neighbor-joining algorithm of Saitou and Nei. Molecular biology and evolution. 1988;5:729–731.

216. Myers EW, Miller W. Optimal alignments in linear space. Bioinformatics. 1988;4:11–7.

217. Thompson Julie D., Koehl Patrice, Ripp Raymond, Poch Olivier. BAliBASE 3.0: Latest developments of the multiple sequence alignment benchmark. Proteins: Structure, Function, and Bioinformatics. 2005;61:127–36.

218. Cline M, Hughey R, Karplus K. Predicting reliable regions in protein sequence alignments. Bioinformatics. 2002;18:306–14.

219. Blackburne BP, Whelan S. Measuring the distance between multiple sequence alignments. Bioinformatics. 2012;28:495–502.

220. Shafee T, Cooke I. AlignStat: a web-tool and R package for statistical comparison of alternative multiple sequence alignments. BMC Bioinformatics [Internet]. 2016 [cited 2017 Jan 31];17. Available from: http://bmcbioinformatics.biomedcentral.com/articles/10.1186/s12859-016-1300-6

221. Anderson CL, Strope CL, Moriyama EN. SuiteMSA: visual tools for multiple sequence alignment comparison and molecular sequence simulation. BMC Bioinformatics. 2011;12:184.

222. Shih AC-C, Lee D, Lin L, Peng C-L, Chen S-H, Wu Y-W, et al. SinicView: A visualization environment for comparisons of multiple nucleotide sequence alignment tools. BMC Bioinformatics. 2006;7:103.

223. Bawono P, Simossis VA, Heringa J. VerAlign [Internet]. [cited 2015 Jul 29]. Available from: http://www.ibi.vu.nl/programs/veralignwww/

224. Morgenstern B, Goel S, Sczyrba A, Dress A. AltAVisT: Comparing alternative multiple sequence alignments. Bioinformatics. 2003;19:425–6.

225. Sokal RR, Michener C. A statistical method for evaluating systematic relationship. University of Kansas Science Bulletin. 1958;28:1409–38.

226. Hirschberg DS. A linear space algorithm for computing maximal common subsequences. Communications of the ACM. 1975;18:341–343.

227. Herman JL, Novák Á, Lyngsø R, Szabó A, Miklós I, Hein J. Efficient representation of uncertainty in multiple sequence alignments using directed acyclic graphs. BMC Bioinformatics. 2015;16:108.

228. Whelan S. Spatial and temporal heterogeneity in nucleotide sequence evolution. Mol Biol Evol. 2008;25:1683–94.

229. Stamatakis A. RAxML-VI-HPC: maximum likelihood-based phylogenetic analyses with thousands of taxa and mixed models. Bioinformatics. 2006;22:2688–90.

230. Shimodaira H. An approximately unbiased test of phylogenetic tree selection. Syst Biol. 2002;51:492–508.

231. Müller J, Müller K. Treegraph: automated drawing of complex tree figures using an extensible tree description format. Mol Ecol Notes. 2004;4:786–8.

232. Barakat A, Müller KF, Sáenz-de-Miera LE. Molecular evolutionary analyses of the Arabidopsis L7 ribosomal protein gene family. Gene. 2007;403:143–50.

233. Sampedro J, Lee Y, Carey RE, DePamphilis C, Cosgrove DJ. Use of genomic history to improve phylogeny and understanding of births and deaths in a gene family. Plant J. 2005;44:409–19.

234. Zahn LM, Leebens-Mack JH, Arrington JM, Hu Y, Landherr LL, Depamphilis CW, et al. Conservation and divergence in the AGAMOUS subfamily of MADS-box genes: evidence of independent sub-and neofunctionalization events. Evolution & development. 2006;8:30–45.

235. Müller KF, Borsch T, Hilu KW. Phylogenetic utility of rapidly evolving DNA at high taxonomical levels: Contrasting matK, trnT-F, and rbcL in basal angiosperms. Mol Phyl Evol. 2006;41:99–117.

236. Jansen RK, Cai Z, Raubeson LA, Daniell H, dePamphilis CW, Leebens-Mack JH, et al. Analysis of 81 genes from 64 plastid genomes resolves relationships in angiosperms and identifies genome-scale evolutionary patterns. Proc Natl Acad Sci U S A. 2007;104:19369–74.

237. Drummond A, Rambaut A. BEAST: Bayesian evolutionary analysis by sampling trees. BMC Evolutionary Biology. 2007;7:214.

238. Richardson D, Simmons M, Reddy A. Comprehensive comparative analysis of kinesins in photosynthetic eukaryotes. BMC Genomics. 2006;7:18.

239. Zhang LB, Simmons MP. Phylogeny and delimitation of the Celastrales inferred from nuclear and plastid genes. Syst Bot. 2006;31:122–37.

240. Huson DH, Richter DC, Rausch C, Dezulian T, Franz M, Rupp R. Dendroscope: An interactive viewer for large phylogenetic trees. BMC bioinformatics. 2007;8:460.

241. Kumar S, Tamura K, Nei M. MEGA3: Integrated software for Molecular Evolutionary Genetics Analysis and sequence alignment. Brief Bioinform. 2004;5:150–63.

242. Jordan GE, Piel WH. PhyloWidget: web-based visualizations for the tree of life. Bioinformatics. 2008;24:1641–2.

243. Chevenet F, Brun C, Banuls A-L, Jacq B, Christen R. TreeDyn: towards dynamic graphics and annotations for analyses of trees. BMC Bioinformatics. 2006;7:439.

244. Page RDM. TreeView: An application to display phylogenetic trees on personal computers. Computer Applications in the Biosciences. 1996;12:357–8.

245. Cranston K, Harmon LJ, O'Leary MA, Lisle C. Best Practices for Data Sharing in Phylogenetic Research. PLoS Curr [Internet]. 2014 [cited 2018 Jan 23]; Available from: http://currents.plos.org/treeoflife/article/best-practices-for-data-sharing-in-phylogenetic-research/

246. Pagel M, Meade A. BayesTraits [Internet]. Reading: School of Biological Sciences, University of Reading; 2007. Available from: http://www. evolution. rdg. ac. uk/BayesTraits. html

247. Stöver BC. The XTG Format of TreeGraph 2 [Internet]. 2015 [cited 2016 Nov 5]. Available from: http://bioinfweb.info/xmlns/xtg

248. Brech P. Extending the metadata model of the phylogenetic tree editor TreeGraph 2 [Internet]. Münster, Germany; 2017 [cited 2017 Dec 22]. Available from: http://www2.ieb.uni-muenster.de/EvolBiodivPlants/en/Theses/Detail?id=21

249. Walls RL, Deck J, Guralnick R, Baskauf S, Beaman R, Blum S, et al. Semantics in Support of Biodiversity Knowledge Discovery: An Introduction to the Biological Collections Ontology and Related Ontologies. PLOS ONE. 2014;9:e89606.

250. Prosdocimi F, Chisham B, Pontelli E, Thompson JD, Stoltzfus A. Initial Implementation of a Comparative Data Analysis Ontology. Evol Bioinform Online. 2009;5:47–66.

251. Ison J, Kalaš M, Jonassen I, Bolser D, Uludag M, McWilliam H, et al. EDAM: an ontology of bioinformatics operations, types of data and identifiers, topics and formats. Bioinformatics. 2013;29:1325–32.

252. Buttigieg PL, Morrison N, Smith B, Mungall CJ, Lewis SE. The environment ontology: contextualising biological and biomedical entities. Journal of Biomedical Semantics. 2013;4:43.

253. Ashburner M, Ball CA, Blake JA, Botstein D, Butler H, Cherry JM, et al. Gene Ontology: tool for the unification of biology. Nat Genet. 2000;25:25–9.

254. Consortium TGO. The Gene Ontology in 2010: extensions and refinements. Nucl Acids Res. 2010;38:D331–5.

255. Panahiazar M, Ranabahu A, Taslimi V, Yalamanchili H, Stoltzfus A, Leebens-Mack J, et al. PhylOnt: A domain-specific ontology for phylogeny analysis. 2012 IEEE International Conference on Bioinformatics and Biomedicine. 2012. p. 1–6.

256. Jaiswal P, Avraham S, Ilic K, Kellogg EA, McCouch S, Pujar A, et al. Plant Ontology (PO): a controlled vocabulary of plant structures and growth stages. Comp Funct Genom. 2005;6:388–97.

257. Munzner T, Guimbretière F, Tasiran S, Zhang L, Zhou Y. TreeJuxtaposer: Scalable Tree Comparison Using Focus+Context with Guaranteed Visibility. ACM SIGGRAPH 2003 Papers [Internet]. New York, NY, USA: ACM; 2003 [cited 2018 Jan 17]. p. 453–462. Available from: http://doi.acm.org/10.1145/1201775.882291

258. Nye TMW, Liò P, Gilks WR. A novel algorithm and web-based tool for comparing two alternative phylogenetic trees. Bioinformatics. 2006;22:117–9.

259. Bremm S, Landesberger T von, Heß M, Schreck T, Weil P, Hamacherk K. Interactive visual comparison of multiple trees. 2011 IEEE Conference on Visual Analytics Science and Technology (VAST). 2011. p. 31–40.

260. Robinson O, Dylus D, Dessimoz C. Phylo.io : Interactive Viewing and Comparison of Large Phylogenetic Trees on the Web. Mol Biol Evol. 2016;33:2163–6.

261. Bouckaert RR. DensiTree: making sense of sets of phylogenetic trees. Bioinformatics. 2010;26:1372–3.

262. Guerra-Gómez J, Buck-Coleman A, Pack M, Plaisant C, Shneiderman B. TreeVersity - Interactive Visualizations for Comparing Hierarchical Data Sets. Transportation Research Record: Journal of the Transportation Research Board. 2013;2392:48–58.

263. Guerra-Gómez JA, Pack ML, Plaisant C, Shneiderman B. Visualizing changes over time in datasets using dynamic hierarchies.

264. Fiorini N, Lefort V, Chevenet F, Berry V, Chifolleau A-MA. CompPhy: a web-based collaborative platform for comparing phylogenies. BMC Evolutionary Biology. 2014;14:253.

265. Meade A, Organ C. BayesTrees [Internet]. Reading: School of Biological Sciences, University of Reading; 2017. Available from: http://www.evolution.rdg.ac.uk/BayesTraitsV3.0.1/BayesTraitsV3.0.1.html

266. Kreft Ł, Botzki A, Coppens F, Vandepoele K, Van Bel M. PhyD3: a phylogenetic tree viewer with extended phyloXML support for functional genomics data visualization. Bioinformatics [Internet]. 2017 [cited 2017 Aug 18]; Available from: https://academic.oup.com/bioinformatics/article/doi/10.1093/bioinformatics/btx324/3835380/PhyD3-a-phylogenetic-tree-viewer-with-extended

267. Gruenstaeudl M. WARACS: Wrappers to Automate the Reconstruction of Ancestral Character States. Applications in Plant Sciences. 2016;4:1500120.

268. Zmasek CM. Archaeopteryx: Visualization, Analysis, and Editing of Phylogenetic Trees [Internet]. Archaeopteryx: Visualization, Analysis, and Editing of Phylogenetic Trees. 2016 [cited 2018 Jan 22]. Available from: https://sites.google.com/site/cmzmasek/home/software/archaeopteryx

269. Letunic I, Bork P. Interactive tree of life (iTOL) v3: an online tool for the display and annotation of phylogenetic and other trees. Nucleic Acids Res. 2016;44:W242–5.

270. Smits SA, Ouverney CC. jsPhyloSVG: A Javascript Library for Visualizing Interactive and Vector-Based Phylogenetic Trees on the Web. PLOS ONE. 2010;5:e12267.

271. Yu G, Smith DK, Zhu H, Guan Y, Lam TT-Y. ggtree: an r package for visualization and annotation of phylogenetic trees with their covariates and other associated data. Methods Ecol Evol. 2017;8:28–36.

272. Py4J - A Bridge between Python and Java [Internet]. [cited 2018 Jan 22]. Available from: https://www.py4j.org/index.html

273. rJava - Low-level R to Java interface [Internet]. [cited 2018 Jan 22]. Available from: https://www.rforge.net/rJava/

274. List M, Ebert P, Albrecht F. Ten Simple Rules for Developing Usable Software in Computational Biology. PLOS Computational Biology. 2017;13:e1005265.

275. Prlić A, Procter JB. Ten Simple Rules for the Open Development of Scientific Software. PLoS Comput Biol. 2012;8:e1002802.

276. Apache Commons – Apache Commons [Internet]. [cited 2018 Jan 29]. Available from: https://commons.apache.org/

277. guava: Google core libraries for Java [Internet]. Google; 2018 [cited 2018 Jan 29]. Available from: https://github.com/google/guava

278. SWT AWT Bridge documentation [Internet]. Class SWT_AWT. Available from: https://help.eclipse.org/oxygen/topic/org.eclipse.platform.doc.isv/reference/api/org/eclipse/swt/awt/SWT_AWT.html

16 Acknowledgements

Many people contributed to making this thesis possible in multiple different ways. I'm very thankful to…

- Kai Müller for supervising this thesis, making it possible over the long time, letting me develop my own profile between biology and informatics and for being with me on our numerous trips to Bonn and Berlin.
- Dietmar Quandt for being my cosupervisor, his ideas on many projects, important feedback to the developed software and always welcoming me at my old institute in Bonn.
- Wojciech Makałowski for being my third supervisor and always providing another perspective on the thesis and bioinformatics in general.
- my students, Lukas Aaldering, Jonas Bohn, Phoebe Brech and Sarah Wiechers for their interest in my projects and contributing to them during their theses and modules.
- Patrick Plitzner for coordinating the development of the *Taxonomic Editor* between Münster and Berlin and for always being there, when I had questions regarding the *EDIT platform* or working with *Eclipse RCP*.
- Thomas Borsch for his cooperation in the *EDIT* project, frequent feedback and beta-testing of *TreeGraph 2* and for inviting us to stay in his house, when we visited the *BGMG*.
- Walter Berendsohn, Anton Güntsch, Tilo Hennig, Norbert Kilian, Andreas Kohlbecker, Andreas Müller and the whole biodiversity informatics team at the *BGBM* for working with me on the *Taxonomic Editor* and helpful advice and fruitful discussions on many different topics.
- Joachim Kurtz for crucial advice and support at the end of my thesis.
- Maik Bartelheimer and Carsten Kemena for cosupervising bachelor and master theses related to by work.
- Scott Kelchner and David Morrison for helpful ideas in the early phase of the thesis that influenced the overall concept.
- Jim Leebens-Mack, Sebastien Roch and Tandy Warnow for organizing the MSA workshop at *IPAM* in 2015, which was of key importance for me and gave me many ideas for the thesis and other projects.
- the developers and contributors to all open-source projects used in this thesis. Without so many other people believing in the idea of free software, this thesis would not have been possible.
- the users of our software for important feedback and suggestions.

- Katharina Schäper for sharing the office with me for many years and always being there when I needed someone to talk to.
- Maik Bartelheimer, Jonas Bohn, Phoebe Brech, Ortrun Lepping, Thomas Rabe, Joachim Röschenbleck, Susanne Sangenstedt, Katharina Schäper, Michelle Thönnes, Susann Wicke, Sarah Wiechers for being my group at the *IEB* and helpful input and discussions and many topics.
- the groups at the *Nees*, the *IoB* and the whole *IEB* staff for a great time, especially Michael Krug for always sharing his space in the office, when I was at the *Nees*.
- the members of the IEB game club for many fun game nights, especially Nicolle Demandt, Jasmin Kurafeiski and Andreas Lange for organizing it.
- the members of the Mittelbauinitiative Münster for working with me towards a better way of working in science.
- my friends on Bonn, The Hague and many other places for the great time.
- my parents for always supporting me and listening when things did not go well.
- Ruth Reinders for motivating me, when I nearly gave up, for great times on numerous hiking trips and for being my best friend for so many years now. In could not have made it until here without you.

Funding for parts of this thesis and travel grants were provided by the *German Research Foundation* (*DFG*), the *National Science Foundation* (*NSF*), the *International Society for Computational Biology* (*ISCB*), the *Gesellschaft für Chemische Technik und Biotechnologie e.V.* (*DECHEMA*) and the *Gesellschaft für Biochemie und Molekularbiologie e.V* (*GBM*).

17 Appendix

17.1 Curriculum vitae

Name: Ben Christoph Stöver

E-mail: stoever@bioinfweb.info

17.1.1 Education

17.1.2 Higher education

17.1.3 Ph. D. studies

17.1.4 Compulsory community service

17.1.5 Professional occupation

Münster, den _____

17.2 List of peer reviewed articles

Barré P, **Stöver BC**, Müller KF, Steinhage V: LeafNet: A computer vision system for automatic plant species identification. *Ecological Informatics* 2017, **40**:50-56
http://dx.doi.org/10.1016/j.ecoinf.2017.05.005

Grimm J, Hoffmann M, **Stöver BC**, Müller KF, Steinhage V: Image-Based Identification of Plant Species Using a Model-Free Approach and Active Learning. In Friedrich G, Helmert M, Wotawa F: KI 2016: Advances in Artificial Intelligence, 2016, Springer International Publishing; 169-176
http://dx.doi.org/10.1007/978-3-319-46073-4_16

Kilian N, Henning T, Plitzner P, Müller A, Güntsch A, **Stöver BC**, Müller KF, Berendsohn WG, Borsch T: Sample data processing in an additive and reproducible taxonomic workflow by using character data persistently linked to preserved individual specimens. *Database* 2015, **2015**:bav094
http://dx.doi.org/10.1093/database/bav094

Stöver BC, Müller KF: TreeGraph 2: combining and visualizing evidence from different phylogenetic analyses. *BMC Bioinformatics* 2010, **11**:7
http://dx.doi.org/10.1186/1471-2105-11-7

17.3 List of conference contributions

Plitzner P, Müller A, Güntsch A, Berendsohn WG, Kohlbecker A, Kilian N, Henning T, **Stöver BC**: The CDM Applied: Unit-Derivation, from Field Observations to DNA Sequences. TDWG 2017 Annual Conference; Ottawa, Canada; 2017
http://dx.doi.org/10.3897/tdwgproceedings.1.20366

Wiechers S, Müller KF, **Stöver BC**: Increasing data accessibility and reuse in phylogenetics by employing externally defined ontologies. 6th annual Symposium of the Münster Graduate School of Evolution; Münster, Germany; 2017
http://go.wwu.de/e4m7n

Stöver BC, Wiechers S, Müller KF: JPhyloIO - A Java library for event-based reading and writing of different alignment and tree formats through one common interface. European Conference on Computational Biology (ECCB); The Hague, The Netherlands; 2016
http://go.wwu.de/po5ac

Wiechers S, Müller KF, **Stöver BC**: New comparison and annotation methods of the phylogenetic tree editor TreeGraph 2. 5th annual Münster Graduate School of Evolution Symposium; Münster, Germany; 2015
http://go.wwu.de/u2gkf

Stöver BC, Wiechers S, Müller KF: Recent development of the phylogenetic tree editor TreeGraph 2. German Conference on Bioinformatics (GCB); Dortmund, Germany; 2015
http://go.wwu.de/cs9xk

Stöver BC, Müller KF: LibrAlign - A powerful Java GUI library for MSA and attached raw and meta data. IPAM Multiple Sequence Alignment Workshop; Los Angeles, USA; 2015
http://go.wwu.de/8gw4j

Stöver BC, Müller KF: AlignmentComparator - Comparing and annotating alternative alignments of the same data set. IPAM Multiple Sequence Alignment Workshop; Los Angeles, USA; 2015
http://go.wwu.de/lekf6

Stöver BC, Müller KF: LibrAlign - A Java library with powerful GUI components for multiple sequence alignment and attached raw and meta data. German Conference on Bioinformatics (GCB); Bielefeld, Germany; 2014
http://go.wwu.de/1xk0t

Stöver BC, Müller KF: AlignmentComparator - A GUI application to efficiently visualize and annotate differences between alternative multiple sequence alignments. 4th annual Münster Graduate School of Evolution Symposium; Münster, Germany; 2014
http://go.wwu.de/1okl3

Stöver BC, Müller KF: LibrAlign - A GUI library for displaying and editing multiple sequence alignments and attached data. BioDivEvo 2014; Dresden, Germany; 2014
http://go.wwu.de/xvda1

Stöver BC, Müller KF: Software in the BioInfWeb project. 2nd annual Münster Graduate School of Evolution Symposium; Münster, Germany; 2012
http://go.wwu.de/p1ivv

Stöver BC, Quandt D, Müller KF: Complex mutations and multiple sequence alignment - Example: Hairpin-initiated repeats (HIRs). 2nd annual Münster Graduate School of Evolution Symposium; Münster, Germany; 2012